UNIVERSIDADE
BEIRA INTERIOR

Performance Analysis of End-To-End Autonomous Driving Systems in Varying Simulated Scenarios
Versão Final Pós-Defesa

Ângelo Miguel Rodrigues Morgado

Dissertação para obtenção do Grau de Mestre em
Engenharia Informática
(2º ciclo de estudos)

Orientador: Prof. Doutor Nuno Gonçalo Coelho Costa Pombo

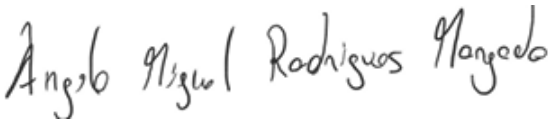
fevereiro de 2025

Declaração de Integridade

Eu, Ângelo Miguel Rodrigues Morgado, que abaixo assino, estudante com o número de inscrição 12306 de/o Engenharia Informática da Faculdade de Engenharias, declaro ter desenvolvido o presente trabalho e elaborado o presente texto em total consonância com o Código de Integridades da Universidade da Beira Interior.

Mais concretamente afirmo não ter incorrido em qualquer das variedades de Fraude Académica, e que aqui declaro conhecer, que em particular atendi à exigida referenciação de frases, extratos, imagens e outras formas de trabalho intelectual, e assumindo assim na íntegra as responsabilidades da autoria.

Universidade da Beira Interior, Covilhã 28/02/2025



Funding

This work was supported by FCT - Fundação para a Ciência e Tecnologia, I.P. by project reference UIDB/50008/2020, and DOI identifier https://doi.org/10.54499/UIDB/50008/2020.

Agradecimentos

Gostaria de agradecer a toda a gente que tornou este feito possível. Sem estas pessoas, seria impossível atingir este marco na minha vida.
Primeiramente gostaria de agradecer à minha família, que sempre me apoiou em todas as minhas decisões e me ajudaram a tornar os meus sonhos reais. À minha mãe que sempre procurou ajudar-me da melhor maneira que pôde e que me ajudou a ter a paciência de santo que tenho hoje. Ao João que sempre me deu conselhos e pelo qual tenho uma dívida que nunca irei conseguir pagar. Ao meu pai que me apoiou nesta jornada; ao meu irmão Luís que sempre me mostrou o amor de um irmão mais velho; ao meu irmão mais novo Hugo, que sabe que pode sempre contar com o seu maninho mais velho. Ao resto da família quero agradecer de igual forma mesmo que não os nomeie aqui (o relatório ia ficar com mais umas 20 páginas ;p), desde o apoio que me deram, ao amor que sempre demonstraram por mim.
À minha namorada, Rita quero mandar um beijão e agradecer por estar sempre lá para mim, seja no melhor ou no pior. Por todas as vezes que me apoiava quando eu mais precisava. Ensinando-me o que é o verdadeiro amor. Não existe uma pessoa neste planeta que eu ame mais do que a minha cara-metade. Amo-te IM.
Quero também agradecer aos meus amigos com quem passei muitas aventuras durante todos estes anos. Quero agradecer à maltinha com quem já me dou há 10 anos, por sempre tornarem cada momento mais alegre e por darem-me na cabeça quando eu precisava. Quero agradecer também aos meus companheiros de informática, com quem caminhei durante 5 anos sempre lado a lado, e com quem superei os maiores desafios que este curso mandou na nossa direção, sempre em equipa. Tenho obrigatoriamente que agradecer a todos os outros amigos que, mesmo não estando no mesmo curso que eu, sempre me deram força e muitas gargalhadas, seja dentro ou fora da biblioteca. A todos os meus amigos quero agradecer pelas memórias que tenho com eles, memórias estas que vou estimar até à velhice. E que estas amizades nunca acabem.
Sem estes 3 pilares jamais conseguiria superar os obstáculos colocados à minha frente durante este percurso. Fico muito grato por ter a sorte de os ter a todos.
Quero adicionalmente agradecer ao meu orientador, Nuno Pombo, por me dar a oportunidade de trabalhar neste projeto, que não só se mostrou ser um grande desafio mas também uma grande oportunidade de crescimento. E ao professor João Neves por providenciar o computador que usei neste projeto, sem ele não teria conseguido acabar a dissertação.

Resumo

Nos últimos anos, a importância da condução autónoma aumentou significativamente devido aos avanços tecnológicos e à evolução das necessidades da sociedade. A condução autónoma promete um transporte mais seguro e mais eficiente, com potencial para reduzir o congestionamento do tráfego e os acidentes. No entanto, conseguir uma condução totalmente autónoma continua a ser um desafio. Uma das principais limitações é que os atuais veículos autónomos não podem conduzir em segurança em todos os cenários, especialmente em condições meteorológicas adversas. Testar estes cenários no mundo real é inviável e muitas vezes inseguro, o que torna a simulação uma ferramenta essencial para o desenvolvimento de agentes de condução autónoma. Os ambientes de simulação permitem testar exaustivamente os veículos autónomos, fornecendo informações valiosas e ajudando a melhorar a fiabilidade e a segurança dos sistemas de condução autónoma. A maioria dos veículos autónomos comercialmente disponíveis seguem uma arquitetura modular, onde cada fase, desde a recolha de informações até ao controlo do veículo estão separadas em diferentes módulos preparados individualmente uns dos outros. Porém, têm aparecido estudos que discutem uma arquitetura diferente, *end-to-end*, onde todos esses módulos são juntos num só e treinados em simultâneo.
Este projeto tem dois objetivos principais, o primeiro é desenvolver uma ferramenta chamada CARLA-GymDrive, cuja funcionalidade é permitir ao utilizador treinar um agente de aprendizagem por reforço num ambiente compatível com a biblioteca *gymnasium* no simulador CARLA; o segundo é treinar agentes com as duas arquiteturas de veículos autónomos *end-to-end* e *modular*, juntamente com dois algoritmos de aprendizagem, DQN e PPO. Os agentes treinados são sujeitos a uma série de testes para avaliar de forma abrangente o seu desempenho e robustez. Este estudo não só realça o papel fundamental da simulação no desenvolvimento de sistemas autónomos fiáveis, como investiga o potencial da arquitetura *end-to-end*.

Palavras-Chave

Aprendizagem por Reforço, Condução Autónoma, *End-to-End*, Qualidade de *Software*, Simulador de Condução CARLA

Resumo alargado

Nos últimos anos, a importância da condução autónoma aumentou significativamente devido aos avanços tecnológicos e à evolução das necessidades da sociedade. A condução autónoma promete um transporte mais seguro e mais eficiente, com potencial para reduzir o congestionamento do tráfego e os acidentes. No entanto, conseguir uma condução totalmente autónoma continua a ser um desafio. Uma das principais limitações é que os atuais veículos autónomos não podem conduzir em segurança em todos os cenários, especialmente em condições meteorológicas adversas. Testar estes cenários no mundo real é inviável e muitas vezes inseguro, o que torna a simulação uma ferramenta essencial para o desenvolvimento de agentes de condução autónoma. Os ambientes de simulação permitem testar exaustivamente os veículos autónomos, fornecendo informações valiosas e ajudando a melhorar a fiabilidade e a segurança dos sistemas de condução autónoma.

A maioria dos veículos autónomos comercialmente disponíveis seguem uma arquitetura modular, onde cada fase, desde a recolha de informações até ao controlo do veículo estão separadas em diferentes módulos independentes uns dos outros, assim é mais fácil fazer com que cada um dos módulos fique apto para esta tarefa; no entanto, esta arquitetura tem um defeito, se um erro ocorre no início da *pipeline* de condução autónoma, este erro vai "sangrar" pelos outros módulos, tornando a deteção de erros mais complicada. Têm aparecido estudos que discutem uma arquitetura diferente, *end-to-end*, cuja premissa envolve juntar todos esses módulos num só, onde são treinados em conjunto. Na teoria, ao serem treinados em conjunto faz com que o modelo seja mais coeso nas suas decisões e impede a propagação de erros, no entanto, o tempo de treino é muito mais alto e é mais difícil perceber como o modelo toma decisões devido à sua natureza *black-box*, o que cria um problema de interpretabilidade.

Este projeto tem dois objetivos principais, o primeiro é desenvolver uma ferramenta chamada CARLA-GymDrive, cuja funcionalidade é permitir ao utilizador treinar um agente de aprendizagem por reforço, num ambiente compatível com a biblioteca *gymnasium*, no simulador CARLA; o segundo é treinar agentes com as duas arquiteturas de veículos autónomos, *end-to-end* e *modular*, juntamente com dois algoritmos de aprendizagem, DQN e PPO. Os agentes treinados são sujeitos a uma série de testes para avaliar de forma abrangente o seu desempenho e robustez.

Espera-se que este projeto contribua não só para avanços significativos na compreensão e aprimoramento de sistemas *end-to-end* de condução autónoma, como também fornecer uma ferramenta que possa ser usada por outros investigadores para conseguirem realizar as suas experiências abstraídos do código complexo do simulador. O projeto foi dividido em duas fases, com espaços de observação e funções de recompensa diferentes. Na primeira fase, devido ao mau balanceamento da função recompensa, os agentes tendiam a ficar parados e a receber um bom *score* por isso. Na segunda fase, devido à função recompensa ser melhor arquitetada, os veículos já estavam a ser incentivados a ter uma boa condução. No final do decorrer do projeto, uma terceira e última fase foi adicionada, com propriedades semelhantes à da segunda. A diferença é que um só agente foi treinado em apenas um cenário muito simples, de forma a reduzir a variância ao máximo, e consequentemente, o tempo de treino. Este foi

treinado durante 120,000 episódios. Esta fase revelou que esta arquitetura é promissora, já que o agente foi capaz de terminar o cenário 7 vezes, no entanto, em média, o agente falhava o cenário perto da linha da meta, revelando que ele precisava mais tempo de treino, trazendo à tona a principal desvantagem desta arquitetura e o porquê de ainda não ter visto nenhuma aplicação real.

Mesmo tendo conseguido obter resultados dignos de uma investigação, devido a restrições de uso da máquina de treino, o agente não teve tempo suficiente para conseguir se desenvolver por completo. Mesmo assim, este estudo relevou não só a importância de uma função recompensa equilibrada na construção de um agente de condução autónoma, como também realçou o potencial da arquitetura *end-to-end*, assim como as suas desvantagens.

Abstract

In recent years, the importance of autonomous driving has increased significantly due to technological advances and the changing needs of society. Autonomous driving promises safer, more efficient transportation, with the potential to reduce traffic congestion and accidents. However, achieving fully autonomous driving remains a challenge. One of the main limitations is that current autonomous vehicles cannot drive safely in all scenarios, especially in adverse weather conditions. Testing these scenarios in the real world is impractical and often unsafe, which makes simulation an essential tool for developing autonomous driving agents. Simulation environments allow for extensive testing of autonomous vehicles, providing valuable information and helping to improve the reliability and safety of autonomous driving systems. Most commercially available autonomous vehicles follow a modular architecture, where each phase, from information collection, to vehicle control is separated into different modules arranged individually from each other. However, there are studies that discuss a different architecture, *end-to-end*, where all these modules are combined into one and trained simultaneously.

This project has two main objectives, the first is to develop a tool called CARLA-GymDrive, whose functionality is to allow the user to train a reinforcement learning agent in an environment compatible with the *gymnasium* library in the CARLA simulator; the second is to train agents with the two autonomous vehicle architectures end-to-end and modular, along with two learning algorithms, DQN and PPO. Trained agents are subject to a series of tests to comprehensively assess their performance and robustness. This study not only highlights the fundamental role of simulation in the development of reliable autonomous systems, but also acesses the potential of the end-to-end architecture.

Keywords

Autonomous Driving, CARLA Driving Simulator, End-to-end, Reinforcement Learning, Software Quality

Contents

List of Figures

List of Tables

List of Acronyms

AD	Autonomous Driving
AI	Artificial Intelligence
AV	Autonomous Vehicle
RL	Reinforcement Learning
TD	Temporal Difference
A2C	Advantage Actor Critic
ABS	Antilock Braking System
ADS	Automated Driving System
CAV	Connected Autonomous Vehicles
CAN	Controller Area Network
CNN	Convolutional Neural Network
DQL	Deep Q-Learning
DQN	Deep Q-Network
FPS	Frames Per Second
GAN	Generative Adversarial Network
GPS	Global Positioning System
HMI	Human-Machine Interaction
IOV	Internet of Vehicles
MDP	Markov Decision Process
MLP	Multilayer Perceptron
PPO	Proximal Policy Optimization
RGB	Red, Green and Blue
SAE	Society of Automotive Engineers
TOF	Time of Flight
UBI	Universidade da Beira Interior
ADAS	Advanced Driver-Assistance System
COCO	Common Objects in Context
MaaS	Mobility as a Service
SLAM	Simultaneous Localization and Mapping
YOLO	You Only Look Once
LiDAR	Light Detection and Ranging
SWOT	Strengths, Weaknesses, Opportunities and Threats
TORCS	The Open Racing Car Simulator
VANET	Vehicular Ad Hoc Network

Chapter 1

Introduction

1.1 Motivation

Autonomous driving (AD) is a topic that has been idealized for decades, with one of its first introductions being what were called the phantom autos in 1926 [1]. There were also movies depicting their own vision of what an autonomous vehicle (AV) is, such as "The Love Bug" (1969) and "The Knight Rider" (1982).

Nowadays, more than ever, AD is gaining increased attention, with multiple companies and researchers pouring tremendous amounts of effort to make this idea a reality. The last couple of years have seen a total investment in the field of billions of dollars, all companies together [2]. This shows that we're closer than ever before to conceiving an AV that can be driven anywhere and bought by costumers.

In a study conducted by the National Highway Traffic Administration [3], 94% of road accidents are caused by human error. Thus, AD represents the future of the automotive industry, poised to revolutionize transportation with the advent of new road vehicles. AVs have the potential to bring about significant changes in urban mobility and city planning, leading to a transformative impact. Beyond this, the adoption of AVs holds the promise of enhancing traffic safety by substantially lowering accident rates, alleviating traffic congestion, and offering solutions to economic challenges.

However, this area of research is facing difficulties, as there are fatal victims as a result of an accident involving an AV, which not only result in less investment being put into the field [2], but also proving that they're not ready for mass production. Another disadvantage is their high development cost [4], result of not only its production, but also its training and testing.

Physical testing on public roads or even on private ones are not only a big factor in the cost, but they're also non reproducible. Testing an ego vehicle inside a simulation is not only cheaper, but it is also possible to have more control over environmental variables [5]. According to the annual Autonomous Mileage Report[1], Waymo's vehicles, a very famous AD company in California, as of 2019, has clocked over 32000 kilometers on public roads in 25 cities and over 24 billion kilometers in simulations.

Nowadays, most AVs are developed in a modular way, that is, each task is handled by a different phase. However, there is the belief that by joining all driving tasks in a single phase has the potential to achieve better results, and a more robust driving. This methodology is known as end-to-end[6].

[1] https://www.dmv.ca.gov/portal/vehicle-industry-services/autonomous-vehicles/disengagement-reports/

1.2 Objectives

This research is guided by two overarching goals:

1. Develop and implement a versatile framework for training RL agents across diverse simulated scenarios. This framework will act as a bridge between the AD simulation and the RL algorithm, following the Gym convention[7]. The Gym standards are a must since most of the RL community follows them, making this framework more universal;

2. Analyze and discuss the potential of the end-to-end methodology in the AD field, by comparing it to the modular method.

To achieve these primary objectives, the following detailed objectives are identified:

1. Conduct a comprehensive study on the potential of end-to-end metods, on the behavior of an ego vehicle within simulated environments.

2. Design and implement a robust training framework capable of accommodating different RL techniques and simulating realistic driving scenarios.

3. Develop and train ego vehicles using various RL algorithms within the established training framework, exploring the efficacy of different approaches.

4. Establish a systematic testing regimen within the CARLA simulator to assess the performance and behavior of the trained AVs across diverse environmental conditions.

5. Perform an in-depth analysis of the agents' behaviors and decision-making processes and deduct the potential of end-to-end in AD.

1.3 Document's Structure

This document is organized into several chapters, each addressing specific aspects of the research on AD technology. The structure is designed to guide the reader through a logical progression of information, from the introduction to the conclusion. Below is an overview of the chapters and their respective contents:

1. **Introduction** (chapter 1) – Provides the background, motivation, and objectives of the research, setting the context for the study.

2. **State of the Art** (chapter 2) – This chapter offers a comprehensive overview of the current landscape in AD technology. Delving into fundamental concepts, environmental challenges, and architectural trends, it provides an in-depth exploration of the prominent open problems within the field. Additionally, the chapter elucidates the intricacies of the reinforcement learning domain, offering a detailed exposition. Furthermore, it delves into the most relevant datasets, simulators, and frameworks, providing a nuanced understanding of the technological foundations shaping the AD simulation paradigm.

3. **Methodology and Chronogram** (chapter 3) – Details the research methodology, outlining the steps involved in the study and presenting a graphical chronogram illustrating the project's timeline.

4. **Results** (chapter 4) – Presents the results of the conducted investigation and discussed these results in detail; it also conducts a Strengths, Weaknesses, Opportunities and Threats (SWOT) analysis in order to evaluate the developed framework.

5. **Conclusion** (chapter 5) – Finishes the report with a final conclusion, accompanied by this project's limitations and possible future paths it can take.

Chapter 2

State of the Art

This chapter offers a comprehensive exploration of the current state of AD technology. It covers foundational concepts, environmental challenges, and architectural trends influencing AVs such as end-to-end. The chapter begins by elucidating fundamental AV principles, navigating through environmental challenges, and exploring frameworks, methodologies, and open problems in the field. A focused examination of RL's role in decision-making and an overview of influential datasets, simulators, and frameworks conclude the chapter, providing a foundational understanding for our subsequent research contributions.

2.1 Autonomous Driving

Autonomous driving refers to the ability of a vehicle to operate and navigate without direct human intervention. Also known as self-driving or driverless technology, AD systems use a combination of sensors and advanced artificial intelligence (AI) algorithms to perceive the environment, make decisions, and control the vehicle's movements.
Recently, this area of research has seen growth, driven by, not only the accumulated knowledge on vehicle dynamics, but also the appearance of new sensors and state-of-the-art computer vision models that aim to mitigate environmental problems. This growth, consequently, is fueling the search for even higher levels of automation than those that exist today. However, AD's performance in a big variety of conditions (i.e., location, environment, etc.) is still suboptimal, making it not robust and trustworthy enough to be implemented in a global scale [8]. And thus, this field needs more researchers and automotive companies to put in the work and try to make this technology more reliable and trustworthy. A few examples of the currently uncertain nature of AVs are when Cruise AVs recalled 950 of its cars after an accident in October 2023 [9], or when a Tesla car failed to recognize a truck in the middle of the road and didn't stop, killing the driver in 2019 [10].
The Society of Automotive Engineers (SAE) calls any system capable of executing dynamic driving tasks (DDT) in a self-sustainable manner, an AD system (ADS).
The SAE defines six levels of automation[1]:

- **Level Zero:** At this level, the vehicle lacks automation, though it may feature assistant functionalities like emergency braking or blind spot warning. However, there is no driving assistance provided.

- **Level One:** This level signifies some capacity for automation, but individual features operate independently. Examples include antilock braking system (ABS), stability control, or adaptive cruise control.

[1] https://www.sae.org/blog/sae-j3016-update

- **Level Two:** At this level, the ego vehicle integrates its automation technologies, allowing simultaneous operation. This may encompass features like lane centering **and** adaptive cruise control, along with automatic braking and obstacle evasion.
- **Level Three:** At this level, the car operates autonomously, but the driver must remain attentive at all times and be ready to resume control in case of an emergency.
- **Level Four:** Level four signifies a car that no longer requires a driver, eliminating the need for pedals and a steering wheel. This level encompasses fully AD, though it may not be applicable everywhere or under all conditions. Typically, it is deployed for local driverless taxis, confined to operation within city limits.
- **Level Five:** At the pinnacle of automation, level five involves a vehicle fully capable of autonomous operation without any human intervention. In this scenario, the car handles all driving tasks in all places and conditions.

From level one to level two, the vehicle is considered an Advanced Driver-Assistance System (ADAS), only having driving support features, whereas, from level three to level five, it's considered an ADS, boasting automated driving features.
Contrary to speculation, Tesla's Autopilot does not meet the criteria for level three autonomy. The continuous requirement for the driver to hold the steering wheel underscores its inability to attain level three, as tragically evidenced by the incident [10]. Achieving level three automation necessitates a higher degree of trust in the autonomous system, eliminating the need for constant manual intervention. In a significant stride toward this goal, in 2023, Mercedes introduced Drive Pilot, the first level three ADS in the USA[2]. This milestone represents a substantial advancement toward true autonomous vehicles [11] and serves as an inspiration for increased investment in ADS by other automotive manufacturers.
This chapter provides a comprehensive, step-by-step exploration of how AD works. It begins with an introduction to the concept of AD, followed by a detailed presentation of various research areas within the field, highlighting key studies and the challenges inherent to these areas. The focus then shifts to a deeper examination of AD architectures, specifically end-to-end and modular approaches. Additionally, a concise yet thorough explanation of RL—an essential aspect of this work—is provided. The chapter concludes by presenting useful datasets and simulators employed in training and testing AD systems.

2.1.1 Impact and Key Areas of Autonomous Driving

2.1.1.1 Impact

The widespread adoption of AD is anticipated to have a significant impact on society. Firstly, perfected AD technology holds the potential to reduce the number of traffic accidents. Secondly, its influence on personal vehicle use versus public transportation could either increase or decrease traffic congestion, consequently affecting air pollution levels [12]. This shift also presents the opportunity for decreased commuting times, enabling people to utilize their time

[2]https://group.mercedes-benz.com/innovation/product-innovation/autonomous-driving/drive-pilot-nevada.html

more efficiently. Moreover, individuals with limited mobility would benefit from easier and more independent travel.
AD has the potential to significantly decrease fatal accidents caused by human errors such as distractions, psychological conditions, and speeding. The aging population, particularly those over 60 years old, is growing rapidly [13]. Improving the mobility options for this age group could enhance their quality of life and overall productivity.
Another area that can benefit from AD is Mobility as a Service (MaaS), including popular services like Uber and Bolt, which have gained widespread acceptance. The seamless integration of AD into MaaS models has the potential to reshape and optimize modern transportation systems, meeting evolving user preferences and expectations.

2.1.1.2 Key Research Areas in Autonomous Driving

AD encompasses a vast array of research topics, making it impractical to cover them all comprehensively in a single document. This subsection highlights the primary areas of research within the field of AD that fall outside the scope of this project. These areas are:

- **Localization and Mapping**: This area involves determining the position of an AV on a map, whether global or local [14, 15, 16, 17, 18]. Accurate localization is crucial for maintaining lane discipline and enabling global navigation capabilities for the AV.
- **Perception**: Perception aims to extract useful information from sensor data, allowing the AV to understand its surroundings [8]. Common techniques in perception include object detection [19], semantic segmentation, object tracking [20], and road lane detection [21].
- **Risk Assessment**: An ADS must constantly evaluate its environment to assess risks and predict the intentions of other road users. Risk assessment involves quantifying uncertainty and determining the level of risk in various driving scenarios [22, 23, 24, 25].
- **Planning and Decision Making**: Driving scenarios are highly dynamic and complex, requiring the AV to plan routes that maximize safety while considering factors like time of arrival and fuel consumption. The AV must also adapt to unexpected situations by making informed decisions, making planning and decision-making critical components of the AD process [8, 26, 27].
- **Human-Machine Interaction (HMI)**: This area focuses on how vehicles interact with drivers. Effective communication interfaces through the HMI module are vital for enhancing the user experience and ensuring the safe operation of AD systems [28].
- **Internet of Vehicles (IoV)**: IoV refers to a networked system where vehicles communicate with each other, infrastructure, and other devices to enhance driving safety, traffic management, and user experience through real-time data exchange and advanced analytics [29, 30, 31, 32]. A notable example is the Aveiro Tech City Living Lab[3] project by the Universidade de Aveiro, which implements a large-scale smart city infrastructure.

[3] https://aveiro-living-lab.it.pt/

- **Ethics**: Ethical considerations are critical in the design of ADS, presenting significant challenges to the field. Issues such as decision-making in life-threatening situations and data privacy must be addressed to ensure the responsible deployment of AVs [33, 34].

2.1.2 Sensors

An ego vehicle seeks comprehensive information about its surroundings to make optimal real-time decisions. With the decreasing cost and increasing computational power of sensors, both researchers and the industry have made significant advancements in the field of AD sensors [35, 36, 37]. The primary challenge for sensors lies in extracting maximum real-time information with minimal errors.
To achieve this, an ADS typically integrates a variety of sensors, ensuring redundancy for both robustness and reliability. There are two types of sensors[8]:

- **Exteroceptive sensors:** Capture information from the surrounding environment.
- **Proprioceptive sensors:** Monitor the internal conditions of the vehicle. It measures metrics such as velocity, acceleration, and yaw. These sensors can be accessed through the Controller Area Network (CAN) protocol in modern vehicles.

This report will primarily focus on exteroceptive sensors, and thus, the other sensor types will not be discussed. The Table 2.1 overviews the pros and cons of the most popular sensors in tabular form, for easier understanding.
One of the biggest challenges of the AD field are the effects of adverse weather conditions on an AV's sensors. Weather is a natural phenomenon that can compromise the ADS's ability to navigate confidently in all situations [8, 38]. Rain, fog [39, 40, 41, 42] and snow [43] are the most well studied weather conditions, even though they still lack sufficient study; however, there are other less investigated weather conditions such as the light's effect on the vehicle's sensors [38].
There are three primary approaches to mitigating these weather-related problems:

- **Mechanical Solutions**: Techniques such as using airflow to clean the camera lens can help maintain sensor functionality.
- **Sensor Fusion**: Combining data from multiple sensors allows them to compensate for each other's weaknesses, enhancing overall system reliability in adverse conditions [44, 45, 46].
- **ML Solutions**: Applying ML techniques, such as de-noising algorithms, can improve sensor data quality under challenging weather conditions [47, 48, 49].

2.1.3 Simulation in Autonomous Driving

Simulation is extremely useful and important in the field of AD for several reasons. First and foremost, simulation environments allow for the comprehensive training of AD agents. These agents can be subjected to a wide range of driving scenarios, including rare and dangerous

Sensors	Advantages	Disadvantages
Camera	• Cheap and cost-effective • Versatile and can capture a wide range of visual information • Only sensor capable of detecting color and texture	• Very susceptible to adverse weather conditions
LiDAR	• Provides high-resolution, 3D point cloud data • Effective in low-light conditions • Precise distance and depth measurements • Can provide a bird's-eye view	• Very susceptible to adverse weather conditions • Very expensive
Radar	• Very long range detection • Resilient to adverse weather conditions	• Lower resolution and precision compared to LiDAR • Sound waves can interfere with the detection process
Ultrasonic	• Cheapest sensor • Suitable for short-range applications • Ideal for close proximity obstacle detection (e.g., parking sensor)	• Limited range and coverage compared to LiDAR or radar • Sound waves can interfere with the detection process

Table 2.1: Overview of the pros and cons of each ADS sensor.

situations, without any risk to human life or property. This extensive training helps in refining the decision-making algorithms of AVs by exposing them to diverse conditions and challenges that would be impractical or unsafe to replicate in the real world. Moreover, simulations can be tailored to account for all functional requisites, ensuring that the agents are well-prepared to handle the various demands and constraints they will encounter on the road [50].

Additionally, simulation has a multitude of applications beyond training decision-making agents. It is invaluable for the development and testing of sensors, which are crucial for the perception and localization tasks of AVs. By simulating different weather conditions, lighting scenarios, and other environmental factors, researchers can evaluate and improve sensor performance in a controlled setting. This process not only accelerates the development cycle but also enhances the reliability and robustness of the sensors used in ADSs. Furthermore, simulation allows for iterative testing and validation of both software and hardware components, facilitating the integration of new technologies and ensuring that they meet all necessary safety and performance standards [51].

2.1.4 Autonomous Driving Architectures

ADSs are designed to work as an individual unit (i.e., ego-only), or even as a group, called the connected autonomous vehicles (CAVs) [52, 53]. The big difference is that ego-only AVs rely only on their sensors, while CAVs also rely on the information provided by other vehicles. However, the most interesting type of architectures for this report is regarding the different phases of the AD cycle. A general representation for this can be observed in Figure 2.1. Firstly, the ego vehicle collects outside information using sensors embedded in it; it also uses Global Positioning System (GPS) as well as sensors information to estimate its position in the world, in a process called localization. Then, the collected data will be processed so the car "understands" what is around it. Using this information, the AV can then plan its actions and trajectories, followed by the translation of these actions into mechanical outputs [54]. It is crucial to note that while the presented architecture serves as a reference, diverse perspectives among researchers exist, leading to varying interpretations and approaches within the AD community.

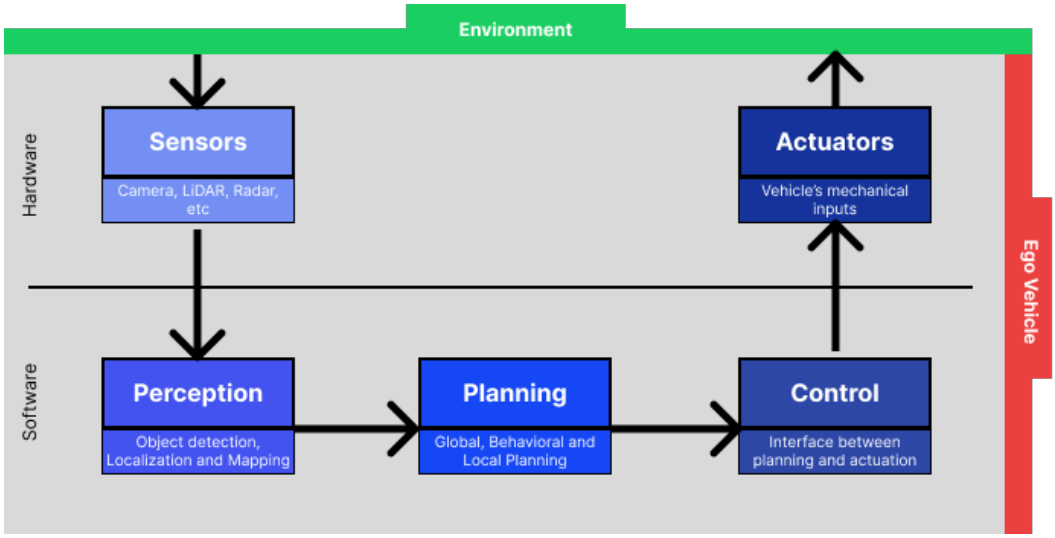


Figure 2.1: General architecture of an autonomous vehicle.

Nevertheless, to be more precise, an ADS's architecture can either be modular or end-to-end; these represent the two major types when designing an ADS [55].

A modular system, also known as the mediation perception approach, stands as the traditional architecture in AD. This architecture segments multiple components into separate modules structured in a pipeline, connecting sensor inputs to actuator outputs. This approach, similar to the structure in Figure 2.1, employs a "divide to conquer" strategy, breaking down the overarching AD problem into more manageable smaller problems. This white-box approach facilitates a clearer understanding of each step in the process. However, it is not without its downsides, notably error propagation, where a fault in one module can impact others, and the inherent complexity that comes with maintaining such a system. A big example of this method is Apollo[4], by the asian giant Baidu.

Conversely, a recent and very interesting architecture in AD is the end-to-end approach. By

[4]https://github.com/ApolloAuto/apollo

directly inferring the ego vehicle's actions from sensor inputs using a deep neural network, this methodology simplifies the entire ADS process, eliminating the need for intermediary components. The resulting actions can either be continuous (e.g., steering angle and speed) or a discrete set of actions (e.g., turn left, accelerate, stop). However, there are not commercially available ADS due to how hard it is to train one. This happens because in order to train one, there is also the need to train every phase of the process as well as their connection. Even though the final model should be more cohesive, the training times are also exponentially bigger. A graphical architecture for end-to-end can be found in the Figure 2.2, Both the sensor phase, and the actuator phase rely on hardware, while the end-to-end system is purely software. There are three primary implementations of end-to-end driving:

- **Supervised deep learning:** Also called imitation learning, it aims to mimic driving styles present in training data, providing a driving style akin to humans but lacking significant generalization power. This method raises the question of whether the agent should emulate human driving or discover an optimal driving strategy instead [56]. This method allows for offline training, this means that the agent doesn't need direct contact with the environment to train.
- **Deep RL:** Learns the optimal driving strategy through trial and error, requiring online interaction with the environment. Algorithms like Deep Q-Learning (DQL) and Proximal Policy Optimization (PPO) are prominent in this category. The objective is for the model to choose a set of actions in order to maximize a reward function.
- **Neuroevolution/Genetic Algorithms:** Learns through evolutionary processes, requiring online interaction with the environment. Although less recognized than the other methods, not even being referenced as a main way of learning in the survey [55], only being referenced as an idea, it eliminates the need for direct supervision, making the training process faster and easier to manage.

With the surge in deep learning, executing end-to-end with neural networks has become feasible and attractive to researchers due to its simplicity, becoming a hot topic among the community. Despite its appeal, end-to-end has a significant drawback. As artificial intelligence gains prominence, the need for trustworthiness and interpretability becomes crucial. Concerns about the black-box nature of neural networks, where decisions are made without clear understanding, present a notable drawback for the end-to-end approach [57, 58]. One example of an end-to-end implementation is OpenPilot[5] from Comma.AI.

2.2 Reinforcement Learning

This section provides an overview of the topic of RL and then explains how it can be used in the field of AD.
RL is an ML paradigm where the agent learns by interacting with the environment and then getting a positive or negative feedback for taking an action, called a reward, therefore, the agent learns through trial and error [59, 60, 61].

[5]https://www.comma.ai/openpilot

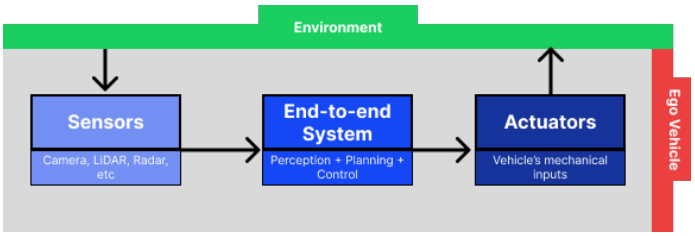


Figure 2.2: End-to-end architecture of an autonomous vehicle.

The RL process can be described in a loop, represented in the Figure 2.3; the agent, based solely on the current state S_t in the current instance of time t, takes an action A_t, the environment then returns to the agent the new state S_{t+1} followed by its reward R_{t+1}.

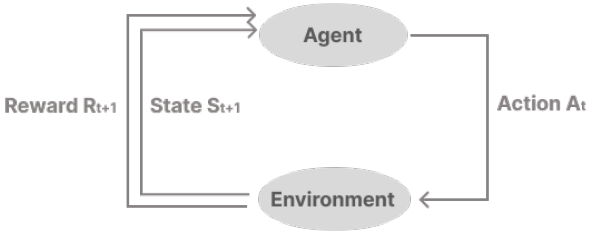


Figure 2.3: RL process loop. Based on the current state S_t, the agent takes an action A_t, in which the environment then returns the new state S_{t+1} and the reward for that state R_{t+1}.

In certain scenarios, the agent might be limited to observations rather than having access to the complete state, which encompasses all information about the environment. For instance, in video games, the agent may only perceive the current frame, or in financial settings, it might observe the stock graph. The key distinction lies in the comprehensiveness of information, where a state s encapsulates the entire environment details, while an observation o provides only a partial view. It's important to note that the terms "state" and "observation" are sometimes used interchangeably, allowing either to be referred to simply as states. The set of all states is called observation space or state space.
The action space is the set of all possible actions the agent can take in the environment. The environment can be discrete, i.e., the action space is finite, with a limited amount of possible actions (e.g., turn left, turn right, accelerate, stop), or continuous, where the action space is infinite (e.g., the steering angle and the speed). Understanding the nature of the action space is essential when choosing an appropriate RL algorithm.

2.2.1 Rewards and Discount

Rewarding is a crucial element in RL, since it is the way the agent knows if a determined action is correct or not. While a reward is associated with a state, the cumulative return $R(\tau)$, is the sum of all rewards, and is described in the equation 2.1, which is equivalent to the equation 2.2, where τ is the set of all instances of time, and r_t the reward at the instance of time t.

$$R(\tau) = r_{t+1} + r_{t+2} + r_{t+3} + ... \quad (2.1)$$

$$R(\tau) = \sum_{k=0}^{\infty} r_{t+k+1} \quad (2.2)$$

However, when using reward functions, the agent tends to prioritize long-term rewards, which are typically less predictable than short-term rewards. To address this, it is essential to discount later rewards, preventing the agent from excessively focusing on them.
This discounting process involves the introduction of a variable known as gamma γ, ranging from zero to one, which is multiplied to the rewards. Commonly, gamma values fall between 0.95 and 0.99. A higher gamma implies a smaller discount, causing the agent to prioritize long-term rewards more, while a lower gamma results in a greater emphasis on short-term rewards. The discounting mechanism is exponential, increasing with each time instance. Therefore, the return equation becomes as represented in equation 2.3 or equation 2.4.

$$R(\tau) = r_{t+1} + \gamma * r_{t+2} + \gamma^2 * r_{t+3} + \gamma^3 * r_{t+4} + ... \quad (2.3)$$

$$R(\tau) = \sum_{k=0}^{\infty} \gamma^k * r_{t+k+1} \quad (2.4)$$

2.2.2 Reinforcement Learning Tasks

A task in RL is an instance of a specific problem. There are two main types of tasks: episodic tasks and continuous tasks. Episodic tasks have a distinct starting point and endpoint, represented by a terminal state, which defines a complete episode. A classic example of an episodic task is a video game with levels. On the other hand, continuous tasks lack a clear starting or ending point, with the agent continuously performing actions until explicitly instructed to stop, e.g., a factory robot.
However, not all problems have a distinct task types and the task can vary depending on how the problem is approached. In AD, normally the problem is episodic, the starting point being where the agent starts the trip and the terminal state is when it arrived in its destination. But, it can also be a continuous task, when the problem only involves the act of driving and adapting to the different road conditions.

2.2.3 Exploration vs Exploitation

Exploration and exploitation are a crucial part of the learning process of an agent. If, for example, the agent could only go for the path it knows it is the best, maximizing its reward, it would never know new paths that, while random, could wield more rewards in the future. Therefore, there needs to be a trade-off between choosing the best known path (exploitation), and choosing new random paths (exploration).
One of the most famous algorithms to perform the trade-off is Epsilon-greedy. By using an epsilon ε variable, which is between zero and one and symbolizes the probability of explo-

ration, the agent can then know, based on the probability, if it is going to explore or exploit. The variable usually starts as one, because in the beginning exploration is mandatory, however, as the time passes, the variable gets decayed because the agent already knows most of the possible actions, and then it is the time to start exploiting.

2.2.4 Policies

A policy π is a function that determines which action the agent takes depending on the current state. The policy can be considered the agent's brain. Training the agent basically means finding the optimal policy π^*, in other words, a policy that maximizes the accumulated return when the agent follows it. In RL the symbol $*$ means optimal or ideal.

2.2.5 Deep Q-Learning

Q-Learning is a value-based algorithm[6] which uses temporal difference (TD) (i.e., the training only occurs after each episode) to train its action-value function. This function, called the Q-function[7], differs from a value function because it gives value to each action of each state, instead of only each state. The classical Q-Learning stores the values of each action-state pair in a table, and the function, given a state and an action, only has to retrieve the table value of them. When the training is complete, the result is an optimal Q-table, and, consequently, an optimal policy.
However, the classical approach has a critical limitation; while it is efficient for simple problems, scenarios with a considerable amount of states, a Q-table stops being able to contain them all. Therefore, an improved version is the solution to this problem, DQL.
Instead of a Q-table, DQL, has a Deep Q-Network (DQN), which receives a state and outputs all possible actions of that state with an inferred Q-value associated with them. This method is better at dealing with bigger problems than the Q-table because instead of outputting the stored value, the network infers the value based on its weights. Then, similarly to Q-Learning the action is chosen using an epsilon-greedy algorithm.
Rather than receiving one frame at a time, sending a batch of frames is more effective, providing the network with temporal context. Predicting trajectories becomes challenging when dealing with only one frame, a limitation known as temporal constraint.
Instead of updating the Q-values directly, a loss function (Q-loss), uses the difference between two values, the Q-value, which is the estimate of the current network, and the Q-target (also called TD target) which is the estimation of a less updated network to stabilize the training. The goal is to minimize this loss, which guides the learning process and improves the accuracy of the Q-value predictions; this process is called the gradient's descent. Ultimately, there will be two networks, the main network and the target network, and at every C steps, the target network is updated with the main network's weights.
An additional technique employed in this algorithm involves the utilization of a replay buffer. This buffer retains previous episodes experienced by the agent, preventing the agent from overlooking past experiences. The objective is to enhance the significance of each episode,

[6] https://huggingface.co/learn/deep-rl-course/unit2/two-types-value-based-methods
[7] The Q stands for quality.

ensuring that the agent retains them effectively. This technique serves to eliminate correlations and mitigate significant oscillations in gradient descent. The algorithm 1 represents the DQL process.

Algorithm 1 Deep Q-Learning training algorithm

```
Start
Initialize replay buffer
Initialize main Q-network and target Q-network with random weights
Initialize episode counter
Initialize exploration rate ε
while not converged or not reached predefined number of episodes do
    Initialize state
    while not end of episode do
        if with exploration probability ε then
            choose a random action
        else
            choose the action with the highest Q-value
        Execute the chosen action in the environment and observe the next state and reward
        Store the transition in the replay buffer
        Sample a random batch of transitions from the replay buffer
        Calculate the target Q-values using the target Q-network
        Calculate the Q-values using the main Q-network
        Update the main Q-network using the loss between Q-values and target Q-values
        Update the target Q-network periodically
        Move to the next state
    Increment episode counter
    Update exploration rate ε
```

2.2.6 Proximal Policy Optimization

PPO can be either a policy-based[8] or actor-critic[9], however in this report, the most prominent will be the actor-critic aspect of it. It works similarly to A2C, however, it avoids making big policy updates. To do that, it uses a clipping interval of $[1-\varepsilon, 1+\varepsilon]$. It is important to note that this algorithm doesn't use the epsilon-greedy, therefore, the symbol ε has a different meaning, it determines the maximum size of the step allowed. In OpenAI's paper [62], the established value is 0.2.

By taking smaller steps, it is more probable for the policy to converge into an optimal solution. A step too large could make the training lose its way (fall off the cliff), or even when it isn't lost, it could make it so it takes a long time until it recovers from the bad step.

PPO uses a special objective function, the clipped surrogate objective function, and it is represented by the equation 2.5; where $r_t(\theta)$ is the ratio function, represented by the equation 2.6, which is the current policy π (represented by the set of weights θ), divided by the old policy. This function is useful for checking how divergent both policies are, and to help clip

[8] https://huggingface.co/learn/deep-rl-course/unit4/what-are-policy-based-methods

[9] https://huggingface.co/learn/deep-rl-course/unit6/advantage-actor-critic

a very divergent policy. $\hat{A}_t$ is the estimation of the advantage function. The clip function is what makes sure $r_t(\theta)$ doesn't leave that range.

$$L^{\text{CLIP}}(\theta) = \hat{\mathbb{E}}_t[\min(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\varepsilon, 1+\varepsilon)\hat{A}_t)] \quad (2.5)$$

$$r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)} \quad (2.6)$$

The training algorithm can be observed in the algorithm 2.

Algorithm 2 Proximal Policy Optimization training algorithm

Start
Initialize policy network parameters θ and value function parameters w
Initialize exploration rate ε
Initialize episode counter
while not converged or not reached predefined number of episodes **do**
  Initialize state
  while not end of episode **do**
    if with exploration probability ε **then**
      choose a random action
    else
      choose the action with the highest $\pi_\theta(a_t|s_t)$
    Execute the chosen action in the environment and observe the next state and reward
    Store the transition in the replay buffer
    Sample a random batch of transitions from the replay buffer
    Calculate the advantage function $\hat{A}_t$
    Calculate the ratio function $r_t(\theta)$
    Calculate the clipped surrogate objective function
    Update the policy network parameters θ and value function parameters w
    Update exploration rate ε
    Move to the next state
  Increment episode counter

2.2.7 Reinforcement Learning in End-to-End Autonomous Driving

RL is a great perspective for an end-to-end system. While imitation learning is also very interesting, its heavy dependence on the training data makes the agent's driving more akin to a human's instead of the optimal way. Also, in Modular approaches to AD, the error propagation and redundant computations, makes them less interesting than their end-to-end counter-part, which, in the last couple of years, have been improving their interpretability by also having the network output attention maps and interpretable maps, which helps not only understand the model's decision, but also debug it more easily [6]. However, its reliance on the environment exploration makes luck a big variable during training, contrary to imitation learning, which can be deterministic and thus, more easily controllable.

To properly build an end-to-end RL model capable of achieving self-driving, a lot of consideration must be put into its structure. Firstly, which sensors should be used. As stated in the subsection 2.1.2, sensor fusion alternatives are more efficient since they cover each other weaknesses.

Another important thing to have consideration is also including navigational inputs, so the AV knows which way to go. There are plenty of ways of achieving this, PlanT's implementation [63], uses point-to-point navigation based on the target location's input, however, they use imitation learning and don't specifically imply its usage for RL. DriveNet's implementation [64] added a very unique way of navigation. They used a CNN based system to detect the turn indicator signals and then feed that information into the end-to-end model in order to perform navigation. These are two examples of the myriad of possible ways of adding navigation context to the end-to-end-model.

Designing a good cost function is essential to making the ego vehicle's performance more optimized. Because it needs to choose the best trajectory from a list of possible trajectories. Useful metrics to take into account when designing a cost function include safety, distance traveled, comfort, infractions, between others.

Depending on the problem (if it is continuous or discrete) it is important to take into account the output of the end-to-end model, in other words, the action space. In continuous models, most implementations use the steering angle and speed as outputs, while in discrete implementations, they use `[turn left, turn right, idle, accelerate, decelerate]` as outputs. However, the action space isn't a constant, and it always depends on the problem at hand.

Safety is a paramount concern in AD, necessitating meticulous attention to ensure the reliability and security of ADS. To mitigate the risk of accidents, it is crucial to expose the vehicle to rare and critical scenarios during training, enabling it to respond effectively to unforeseen real-world situations. This proactive approach contributes to enhancing the system's robustness and preparedness.

Moreover, ensuring the reliability of an AV becomes a pivotal aspect of Software Quality. The design and implementation of comprehensive testing procedures play a crucial role in evaluating the AV's performance under various conditions. This phase is arguably one of the most vital components of the entire development cycle, emphasizing the importance of these practices in delivering a safe and dependable AD experience.

Some interesting implementations with the two main algorithms, DQN and PPO include the study [65], which implemented a DQN model, and trained and tested it in the CARLA simulator. Instead of training only one individual vehicle, it trains five CAVs. The observation space it used was a 5x5 array, which represents the five vehicles and information about them, including the x and y parameters of both position and velocity. However, the authors don't state the fifth characteristic, not even in the result tables, which raises some doubts about the study. Nonetheless, the action space is composed of five actions, these being `[LANELEFT, IDLE, LANERIGHT, FASTER, SLOWER]`. Their reward function takes into account three things, if the car stays on the right lane, if it maintains a high speed, and if it avoids collisions. Their testing metrics are very interesting, being collision rate, speed, total reward, and number

of steps without any infractions or collisions. They test the ego vehicle in most CARLA town maps, with two different weather sets, and with three different traffic types. Their results were shown, but without any comment if they're good or bad. Overall, despite some questions, they present an interesting idea.
Another implementation [66] discussed a very interesting approach. Initially, their end-to-end model utilized PPO, which is a very rare algorithm in these articles despite its massive potential in the field. They also employed additional implementations, such as a random network distillation, which helps find more novel paths in the exploration, therefore making it much more efficient. By using actor-critic methods, they managed to avoid the impact of overestimation. This study shows good results, however they don't point out the observation space, action space and reward function they used. The AV was trained and tested in The Open Racing Car Simulator (TORCS).

2.3 Datasets and Simulators

When developing an ADS, selecting the appropriate tools is crucial. This includes choosing the right datasets tailored to the problem and the suitable simulator for testing the ADS, especially when real-world testing may not be feasible.
This section provides insights into some of the most renowned datasets and simulators, offering concise overviews. Additionally, it explores well-known open-source driving frameworks, comprising pre-trained ADSs ready for testing in simulated scenarios. Finally it delves into similar tools to CARLA-GymDrive, the framework developed in this project.

2.3.1 Datasets

Datasets serve as the foundation for machine learning models, providing the necessary diversity and complexity found in real-world driving scenarios. Here, we delve into a selection of prominent datasets widely utilized in the development and assessment of ADS.
The Common Objects in Context (COCO) dataset [67] stands as a cornerstone for training and evaluating object detection and segmentation models in the context of cameras. It encompasses a diverse collection of images captured across various environments, offering a comprehensive benchmark for computer vision tasks.
Designed specifically for semantic and instance segmentation tasks, Cityscapes [68] provides a rich dataset with 5,000 labeled images from 50 different cities. Its urban scenes offer a challenging yet realistic backdrop for evaluating algorithms in urban environments.
ApolloScape [69] is a multifaceted dataset offering an extensive array of tasks crucial for AD research. Encompassing scene parsing, car instance analysis, lane segmentation, self-localization, trajectory understanding, and LiDAR 3D detection/tracking, ApolloScape provides a holistic resource for algorithm development. Notably, it covers diverse weather conditions, contributing to the robustness of autonomous systems.
Known for its versatility, the KITTI dataset [70] covers stereo, optical flow, visual odometry, 3D object detection, and 3D tracking. Widely adopted in the research community, KITTI serves as a benchmark for algorithms in multiple aspects of AD.

Offering a real-world perspective, Comma2k19 [71] consists of over 33 hours of commute footage on California's 280 highway. This dataset, curated by Comma.ai, provides valuable insights into actual driving scenarios, making it an essential resource for testing and validating AD algorithms.
Lastly, ImageNet [72] is a large-scale dataset made of more than 14 million images, and more than 21000 labels, which are called synsets. This dataset's focus is computer vision and its labels cover pretty much everything that someone would want to identify using ML. It is commonly used to train backbones of ML architectures such as ResNet-50 and EfficientNet. These datasets collectively contribute to advancing the capabilities of ADS by providing diverse, real-world scenarios for training and evaluation. Their utilization ensures that autonomous systems are well-prepared to handle the complexities of dynamic environments.

2.3.2 Simulators

The choice of simulators plays a crucial role in providing a realistic and diverse environment for training and evaluation. In this subsection, several prominent simulators widely employed in the field of AD research are explored.
CARLA [73] stands out as a versatile simulator that offers a comprehensive set of options for sensors, allowing users to tailor their simulations to specific requirements. Users can dynamically adjust weather conditions, environmental parameters, and customize map settings. However, CARLA is noted for having a limited number of pre-made maps, potentially leading to overfitting in deep learning models; but as of version 0.9.15[10], CARLA introduced a procedural map generation tool as an experimental feature in order to prevent this problem. CARLA is well maintained and is regarded as a standard in AD simulation.
Built on top of CARLA, SUMMIT [74] enhances the simulation experience by introducing additional parameters that govern agent behavior, such as aggression and traffic attentiveness. Moreover, SUMMIT has the capability to generate simulations from real-world maps using OpenStreetMap, extending CARLA's map-making functionality. However, it seems that this tool has been discontinued since 2022.
Developed by LG and built on Unity, SVL [75] focuses on end-to-end ADSs, and provides a user-friendly interface with the flexibility to incorporate plugins for adding new sensors. Users can control various environmental variables, including time of day, weather, and road conditions. SVL allows the creation of maps from existing 3D environments and utilizes the SCENIC framework for simulation. Much like SUMMIT, SVL hasn't seen an update since 2022.
One very interesting simulator that exists since 2015, but its popularity spiked in recent times, is the game BeamNG.drive[11], which in its core is a physics engine with the best physics out of all simulators and with real vehicle destruction. Addressing the opportunity, BeamNG launched BeamNG.tech[12], which lets the users test their ADS with highly realistic physics. This library is constantly updated and offers a big variety of sensors (camera, LiDAR, IMU, ultrasonic, etc.) and even has ground truth support.

[10] https://carla.org/2023/11/10/release-0.9.15/
[11] https://www.beamng.com/game/
[12] https://beamng.tech/

These simulators collectively contribute to advancing the capabilities of ADS by providing realistic, configurable, and diverse environments for training and testing. The choice of simulator depends on specific research needs and the desired balance between customization and pre-defined content.

2.3.3 Similar Tools

CARLA-GymDrive is a very useful tool for researchers aiming to work on AD. As expected, there are similar tools. This subsection aims to present some similar tools as well as compare them to the tool developed in this project.
Farama's Gymnasium [7], previously OpenAI Gym, is an API that provides simulated training environments to train and test RL agents. It's one of the most famous environment structure and a standard in the area. CARLA-GymDrive was developed with this API in mind and is compatible with any library that also supports Gymnasium.
DeepDrive [76] is a platform developed for AD research, offering a self-developed simulation that is compatible with the Gymnasium library. While DeepDrive provides valuable functionalities, it has not received updates recently, leading to potential limitations in support and development. Our tool, CARLA-GymDrive, aims to deliver similar functionalities but leverages the CARLA simulator, known for its comprehensive and realistic simulation environment. CARLA continues to receive regular updates, ensuring ongoing improvements and support for cutting-edge AD research.
MACAD-Gym [77] is another tool designed for training RL agents within the CARLA simulator using Gymnasium. MACAD-Gym supports multi-agent training features, which are advantageous for complex AD scenarios. However, CARLA-GymDrive offers a more user-friendly interface for customizing scenarios and the ego vehicle's sensors. While MACAD-Gym requires manual coding for these customizations, potentially complicating setup and usage, CARLA-GymDrive's organized code structure and intuitive customization options simplify the process for researchers, enhancing usability and efficiency.
NVIDIA DRIVE[13] is an attempt from the giant NVIDIA to AD. It is one of the few major corporations working on the end-to-end architecture. Their models, trained on vast amounts of driving data, map raw sensor inputs to driving commands. However, these systems are still mainly research-focused and not yet robust enough for full-scale deployment, further solidifying the significant challenges associated with achieving fully reliable end-to-end AD systems.

2.4 Conclusion

In this chapter, an in-depth exploration of the current state of AD has been presented. The overview encompassed crucial aspects of the AD domain, including fundamental concepts of RL, and detailed insights into prominent datasets and simulators. By delving into these diverse facets, this chapter has provided a comprehensive guide to the multifaceted landscape of AD.

[13] https://www.nvidia.com/en-us/self-driving-cars/

The field of AD is one with multiple open problems, and even though it has been more relevant recently, it still lacks sufficient studies and thus, needs more research put into it.

2.4.1 Proposed Research Agenda

After extensive research into the overall AD (AD) field, this report's research agenda will focus on two key areas: the role of simulation in AD and the potential advantages and disadvantages of the end-to-end method in an AD system. Although there are numerous existing studies, the end-to-end approach has not yet achieved the same level of maturity as the modular counterpart, despite its clear advantages; this disparity provides an excellent opportunity for in-depth research on why that is.

Chapter 3

Methodology and Development

In this chapter, the methodology employed for the research will be detailed, outlining the step-by-step process involved. The chapter will provide a comprehensive explanation of the procedures and techniques used to address the research questions and objectives. The chapter also presents the software engineering that went towards the development of the framework. Additionally, a chronogram will be presented to outline the timeline for each phase of the research.

3.1 Project's Methodology

Both training and testing will be held in a simulator, since it doesn't require the acquisition of physical sensors or even a vehicle, and it provides better control over environmental and traffic variables. The project will be divided into two main tasks:

1. **Implementation of a training/testing framework:** This framework will act as an interface between the simulated environment and the RL agent; providing methods that the agent can call to interact with the environment, and from it, retrieve the necessary information to act.

2. **Training and testing RL agents:** Through the developed framework, an autonomous driving agent will be trained in urban scenarios. These scenarios will consist of either straight roads or curves, without additional complexities such as traffic lights, stop signs, or other obstacles. The focus of this thesis is solely on training an agent to learn how to drive. Multiple RL algorithms (PPO and DQN) will be explored, with variations in their structure (end-to-end and modular). After training, the agents will be tested and compared to analyze their performance.

3.1.1 Simulator Selection

Between the simulators presented in subsection 2.3.2, only CARLA and BeamNG.drive are interesting candidates. However, due to its intuitive nature and the array of features it offers, CARLA will be the simulator used not only to train the agents, but also test them.

3.1.2 Training Method and Conditions

Training an effective AD agent requires meticulous fine-tuning and, most importantly, extensive training time. The scale of this training can extend to weeks for a single agent. For instance, the study by Pérez et al. [78] trained their agent for 120,000 episodes, while Ekim et al. [79] trained theirs for 60,000 episodes. However, given the constraints of this project,

training multiple agents in such an extensive number of episodes is not feasible. Therefore, the agents in this work will be trained for 5,000 in the first phase and 15,000 episodes in the second phase. Only in the last phase of the project is a single end-to-end agent going to be trained for 120,000 episodes in order to get some relevant discussions surrounding it.
To mitigate overfitting, it is common to train an agent across various CARLA town maps. However, this approach significantly increases training time and can diminish results. Consequently, this project will utilize only one CARLA town map, specifically `Town01`, chosen for its simple yet urban structure; the Figure 3.1 shows a bird-eye's view of the map. Additionally, the number of weather configurations used during training significantly impacts the agent's performance. While multiple weather configurations can reduce overfitting, they also lengthen training time. In this project, the initial training phase will employ a random weather preset for each episode, while the second phase will use a single weather preset, sunny. The reason this project simplifies all the variables is that training an AD agent with such variance is impossible in the time frame of this project; multiple years would be necessary in order to train one.

Figure 3.1: Birds-eye view of the Town01 map.

The framework used to train the neural network will be PyTorch, since it's reliable, provides a set of tools to customize the network's architecture at will, and it uses the Python language, which is very practical to use.

3.1.3 Testing Parameters and Variables

To effectively evaluate the architectures and the trained agent, it is crucial to construct a comprehensive test suite. This falls within the domain of software quality, requiring meticulously designed tests to properly assess the ADS.
The developed framework already includes testing capabilities by providing the reward for each of the agent's actions. Testing will be based on the rewards accumulated by the agent under evaluation, with total rewards being normalized for better comparability.
Four distinct test scenarios, separate from the training scenarios, will be constructed within

the same map under sunny weather conditions. These scenarios are deterministic, ensuring that all agents are evaluated consistently. All scenarios are set in the `Town01` map: two on straight roads (one on a bridge and the other in a residential area), and the other two involve navigating left and right turns.
Each agent will be tested over 40 episodes, cycling through the scenarios sequentially without immediate repetition of the same scenario. Due to the deterministic nature of the test setup, the order of scenarios will remain consistent for all agents, ensuring a fair and standardized evaluation.

3.1.4 Ego Vehicle Configuration

The vehicle used for the training and testing will be a Tesla Model 3, as shown in the Figure 3.2. Its sensors' configuration is as follows:

- RGB Camera:
  - Image Size: 640 x 360
  - Field of View: 90 degrees
  - Sensor Tick: 0.0 seconds
  - Location: (x: 0.8, y: 0.0, z: 1.7)
- GNSS:
  - Sensor Tick: 0.0 seconds
  - Location: (x: 0.0, y: 0.0, z: 0.0)
- Collision Sensor:
  - Location: (x: 0.0, y: 0.0, z: 0.0)
- Lane Invasion Sensor:
  - Location: (x: 0.0, y: 0.0, z: 0.0)

Figure 3.2: Ego vehicle.

This suite of sensors ensures that the ego vehicle has the perception capabilities necessary to navigate, detect collisions, and maintain lane discipline effectively.

3.1.5 Agent's Hyperparameters

In RL fine-tuning the model's hyperparameters is essential, since RL algorithms are very hyperparameter sensible. This subsection details the hyperparameters used for both RL algorithms[1]. It is necessary to note that different algorithms have different hyperparameters, and the hyperparameters that maximize one algorithm will not do it to the other. In order to maximize each algorithm to its best, a hyperparameter optimization framework called Optuna[2].

3.1.5.1 DQN

The DQN algorithm hyperparameters can be found in the Table 3.1.

Hyperparameter	**Value**
Learning Rate	1×10^{-4}
Buffer Size	10000
Batch Size	32
Gamma	0.99
Tau	0.005
Initial Epsilon	0.9
Final Epsilon	0.01

Table 3.1: DQN Agent Hyperparameters

3.1.5.2 PPO

The PPO algorithm hyperparameters can be found in the Table 3.2.

Hyperparameter	**Value**
Learning Rate	1×10^{-4}
Number of Steps	1024
Batch Size	64
Gamma	0.999
GAE Lambda	0.98
Entropy Coefficient	0.01

Table 3.2: PPO Agent Hyperparameters

3.1.6 Chronogram

The chronogram present in the Figure 3.3 outlines the timeline for the master's thesis. This ensures that all tasks are completed within the designated periods.

[1]The RL cheatsheet explains most of the hyperparameters meaning. It is present on
[2]https://optuna.org/

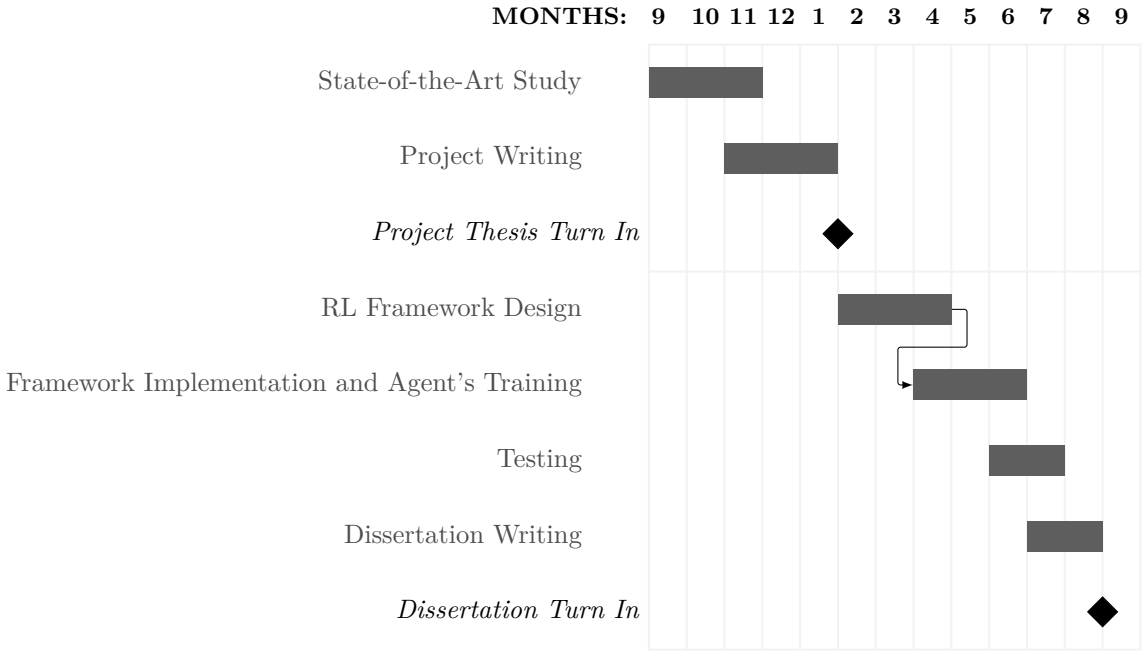


Figure 3.3: Gantt chart showing the rough planning for the master's thesis development.

3.1.7 SCRUM

The project was divided into seven sprints using the SCRUM method. This agile approach allowed for iterative development and continuous feedback, ensuring that each phase of the project was thoroughly planned and executed. By breaking the project into manageable sprints, I was able to maintain focus, adapt to changes, and ensure steady progress toward the completion of the thesis. The sprints can be observed in the diagram present in the Figure 3.4.

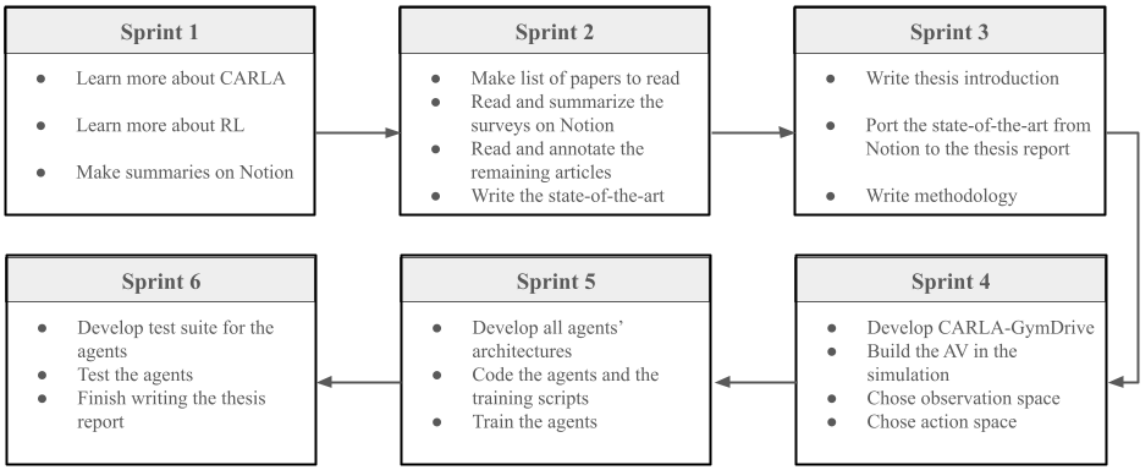


Figure 3.4: SCRUM sprints diagram.

3.1.8 Hardware and Software Configuration

To ensure efficient training and testing of the agents, I utilized a robust hardware and software setup. The Table 3.3 outlines the specifications of the hardware and operating system used throughout this project.

Component	**Specification**
CPU	Intel Core i7-10700
GPU	NVIDIA GeForce RTX 3060 12GB VRAM
RAM	64GB
Storage	1TB NVMe SSD
Operating System	Pop! OS 22.04 LTS
Python Version	3.8
Deep Learning Framework	PyTorch 2.2.2
Simulation Environment	CARLA 0.9.15

Table 3.3: Hardware and Software Configuration for Training and Testing.

3.2 Framework Development

3.2.1 Introduction

The tool developed in this project, named **CARLA-GymDrive** is a powerful framework designed to facilitate RL experiments in AD using the CARLA simulator. By providing a Gymnasium-like environment, it offers an intuitive and efficient platform for training driving agents using RL techniques.

This tool is hosted on GitHub[3] and at the time of the writing, has already garnered thirteen stars and three forks, highlighting its potential and significance in the field of AD research.

3.2.2 Architecture

3.2.2.1 System Architecture

The CARLA-GymDrive framework comprises three primary components: the RL agent, the CARLA-GymDrive environment (client side), and the CARLA simulator (server side). The RL agent sends its action to the CARLA-GymDrive environment, which in turn controls the simulator; this control is encompasses simulation control (i.e., map, weather, traffic, and pedestrian control) and ego vehicle control. The simulator provides all the information relating to the simulation back to the CARLA-GymDrive environment, which then computes the state and reward signals to be sent back to the RL agent. This setup facilitates a smooth interaction loop for training and testing AD policies, as illustrated in Figure 3.5.

[3]https://github.com/angelomorgado/CARLA-GymDrive

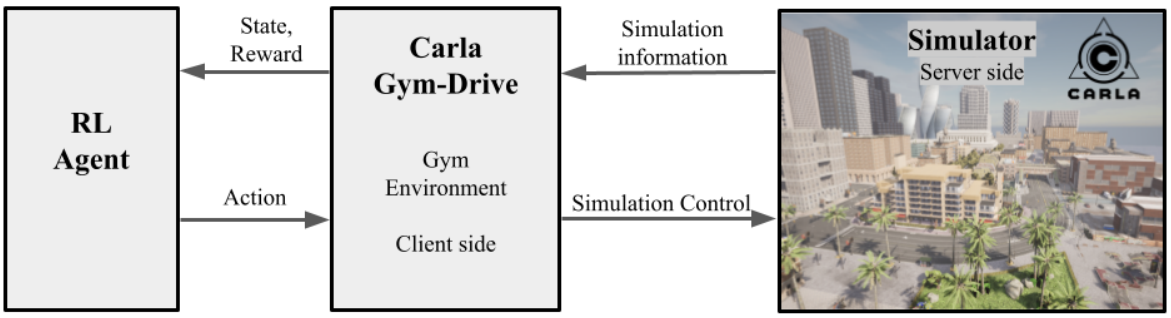


Figure 3.5: CARLA-GymDrive architecture.

3.2.3 Features

3.2.3.1 Main Objective

CARLA-GymDrive's objective is simple, give the user total control over the CARLA simulator to train and test RL agents without the need for extensive coding, therefore cutting development time and allowing the user to focus on experimentation.

3.2.3.2 Core Features

CARLA-GymDrive stands out from similar tools for its ease of use. Most things can be easily customized in the code or even in JSON files. Some of its core features include the following:

- **Compatibility with the Gym library**: Environment interaction can be achieved using only gym methods, such as `reset` or `step`. This decreases the amount of code needed to run training/testing episodes. An example of the simulation running for ten episodes can be found in the code snippet in the Figure 3.6.

```
import env.environment
import gymnasium as gym

env = gym.make('carla-rl-gym', time_limit=30, initialize_server=True,
↪  synchronous_mode=True, continuous=False, show_sensor_data=True)

for i in range(10):
    obs, info = env.reset()
    while True:
        action = env.action_space.sample()
        obs, reward, terminated, truncated, info = env.step(action)

        if terminated or truncated:
            print('Episode terminated cleaning environment')
            break
env.close()
```

Figure 3.6: Example of the use of the CARLA-GymDrive framework.

- **Scenarios customization**: Customizing scenarios for the ego vehicle is an extremely crucial aspect in its training/testing. This tool facilitates that customization by allowing

the user to present a JSON file. The creation of a scenario is as easy as shown in the code snippet present in the Figure 3.7. Any type of scenarios can be built using this tool; from a traffic jam in a highway to simply driving in a rural road. the user simply needs to choose a map, and place the vehicle in the desired position. The user can also specify the type of scenario (e.g., highway, junction, stop sign, etc.) if it is needed.

```
{
    "Town10HD-ClearNoon-Junction-3": {
        "map_name": "Town10HD",
        "weather_condition": "Clear Noon",
        "initial_position": {"x": 40.3, "y": 40.3, "z": 0.3},
        "initial_rotation": {"pitch": 0.0, "yaw": 90.0, "roll": 0.0},
        "target_position": {"x": 74.5, "y": 70.0, "z": 0.3},
        "target_gnss": {"lat": -0.000628, "lon": 0.000678, "alt": 0},
        "traffic_density": "High",
        "situation": "Junction"
      },
    ...
}
```

Figure 3.7: Scenario JSON example.

- **Custom vehicular sensors**: The tool offers the ability to change the ego vehicle's sensors and adjust their parameters in a very easy manner, by simply using a JSON file. An example of this is shown in the code contained in the Figure 3.8.

- **RL libraries ready**: The tool has built-in compatibility with gym-compatible RL libraries such as Stable-Baselines3.

3.2.3.3 Additional Features

List and describe additional features that enhance the framework's functionality.

- **Streamlined environmental control**: The user can easily change the observation and action spaces as well as the reward function to better suit their needs.

- **Modular design**: The code is well organized and divided by classes, making it not only easy to understand but also very customizable for any users.

- **Sensor visualization**: The user can choose to observe the ego vehicle's sensors through a PyGame window while the simulation is running.

- **Ego vehicle physics configuration**: Some problems require a more realistic approach to physics in different weather scenarios. The CARLA simulator doesn't change the vehicle's physics automatically to accommodate the new weather. The framework developed in this project offers ego vehicle's customization in different weather scenarios, through a JSON file, where the user can customize the vehicle physics in dry or wet conditions.

3.2.4 Software Engineering

3.2.4.1 Functional Requirements

The functional requirements detail the specific behaviors and functionalities that CARLA-GymDrive must provide to meet user needs.

1. The framework must allow the simulation to be controlled via standard Gym methods.
2. Users must be able to create and modify driving scenarios using JSON files.
3. The framework must support the addition and modification of ego vehicle sensors through JSON configuration.
4. The framework must be compatible with Gym-compatible RL libraries, such as Stable-Baselines3.
5. Users must be able to customize the observation space.
6. Users must be able to customize the action space.
7. Users must be able to define and modify the reward function as per their requirements.
8. The framework must support the visualization of the ego vehicle's sensors through a separate window.
9. The framework must allow the user to adjust the vehicle's physics in different weather scenarios.
10. The framework must give the user a requirements file, so the user can install the required libraries with minimal effort.
11. The framework must be able to support both discrete and continuous actions, so it is compatible with the most amount of algorithms possible.
12. The framework must give the option to customize how long a single episode lasts.
13. The framework must be able to initialize the server automatically if the user wishes, and also use an already existent server process without creating a double.
14. The framework must give the option to run the simulation in synchronous mode and in asynchronous mode.
15. The framework should give the option to randomize the traffic in each episode or follow what's in the scenarios file.
16. The framework must give the option to disable traffic from all episodes.
17. The framework should give the option to randomize the weather in each episode or follow what's in the scenarios file.

18. The user should be able to choose if they see all the messages from the tool (verbose), or rather hide such messages.

19. The framework should give the option to give a seed to the environment, so the episode is deterministic.

20. Users must be able to customize the PyGame window settings, such as image size, grid layout, margin, and border width.

21. Users must be able to configure vehicle and sensor settings, including sensor FPS and their verbosity.

22. Users must be able to adjust simulation settings such as host, port, timeout, rendering quality, if the simulation is ran offscreen, delta seconds, and FPS.

23. Users must be able to configure environment settings such as scenarios file, maximum steps per episode, and waypoint spacing.

3.2.4.2 Non-Functional Requirements

The non-functional requirements define the quality attributes, performance criteria, and constraints that CARLA-GymDrive must meet.

- **Usability**: The framework should be easy to use and configure, allowing users to quickly set up and run simulations.
- **Performance**: The framework should handle simulations efficiently, minimizing latency and resource consumption.
- **Scalability**: The framework should support large-scale simulations and be able to handle increased computational loads.
- **Modularity**: The code should be modular to facilitate easy customization and extension.
- **Maintainability**: The framework should be designed for easy maintenance and updates.
- **Reliability**: The framework should provide consistent and reliable performance across different simulations and configurations.

3.2.4.3 Sequence Diagram

A sequence diagram is a visual representation used in software engineering to depict the interaction between objects in a sequential order to illustrate how processes operate with one another and in what order. The Figure 3.9 represents the sequence diagram for this project's framework. It depicts how the user interacts with the RL agent, and then how the RL agent interacts with the environment. Every single part of this diagram is necessary for the system to work.

3.2.4.4 Code Structure

The structure for the source code of this framework is listed below:

```
CARLA-GymDrive/
├── src/
│   ├── env/
│   │   ├── env_aux
│   │   ├── README.md
│   │   ├── environment.py
│   │   ├── observation_action_space.py
│   │   ├── pre_processing.py
│   │   └── reward.py
│   ├── config/
│   │   ├── README.md
│   │   ├── configuration.py
│   │   ├── default_scenarios.py
│   │   ├── default_sensors.py
│   │   └── default_vehicle_physics.py
│   └── carlacore/
│       ├── README.md
│       ├── display.md
│       ├── keyboard_control.py
│       ├── map_control.py
│       ├── sensors.py
│       ├── server.py
│       ├── traffic_control.py
│       ├── vehicle.py
│       ├── weather_control.py
│       └── world.py
├── LICENSE
├── README.md
├── carla_0.9.15_vistual_env.yml
├── helpful-scripts/ .2 example_sb3_dqn_training.py
├── main.py
└── requirements.py
```

3.2.4.5 Documentation

The CARLA-GymDrive framework is fully documented, with each module explained with maximum detail. The tool's documentation can be found in the `README.md` file of the repository[4].

3.2.4.6 Maintenance and Support

Maintaining such a framework can be a hard task, but achievable with good organization. The following list details the strategy employed for its maintenance:

- **Bug Tracking**: The framework goes through a scrutinous testing regimen in order to find any existing bug. Some bugs were found along the developing pipeline, and were

[4]https://github.com/angelomorgado/CARLA-GymDrive/blob/main/README.md

swiftly dealt with.

- **Updates**: Any updates to the framework will be through Git. Therefore, any person interested on the tool can be constantly up-to-date with it by going to its GitHub repository.

- **Community Support**: GitHub offers an *Issue* page to the repositories where users can alert the developer of any problem they're facing with the framework. They can also do a pull request with the fix if they desire to directly solve the problem themselves. One example of the community support was the issue raised by the user song-hl, who helped me fix the custom extractor for the PPO agent in the stable-baselines3 library.

3.2.5 Use Cases

3.2.5.1 Case Study 1

One potential use case for the CARLA-GymDrive framework is in the development and testing of autonomous delivery RL vehicles. In this scenario, a delivery company seeks to deploy AVs to navigate urban environments, deliver packages to specified locations, and return to a central hub. Using CARLA-GymDrive, researchers can simulate various delivery routes, traffic conditions, and environmental factors to train and evaluate the performance of their RL-based navigation algorithms. The framework allows for the integration of realistic sensory inputs, enabling the delivery vehicle to perceive and respond to its surroundings effectively. By experimenting with different policy architectures and reward functions within the CARLA simulator, developers can fine-tune the vehicle's behavior to ensure timely deliveries while minimizing collisions and adhering to traffic regulations. This use case highlights the framework's capability to facilitate comprehensive testing and refinement of AD systems in a controlled, virtual environment. Although RL is not necessarily needed to train a working delivery agent, the lack of datasets with this kind of specific data is absent, therefore making RL a very attractive choice, since building a dataset for these kinds of problems is very costly.

3.2.5.2 Case Study 2

A group of investigators aims to test the efficiency of different sets an RL agent's sensors under various weather conditions. Using the CARLA-GymDrive framework, they can simulate an array of environmental scenarios including clear skies, heavy rain, fog, and night-time driving. The framework allows the researchers to configure the ego vehicle with diverse sensor setups, such as RGB cameras and Lidar sensors, to evaluate their performance in detecting and responding to different driving situations. This enables them to identify the optimal sensor configurations for safe and efficient AD, therefore maximizing the RL agent's performance.

3.3 Problem Formulation

An important aspect of RL is problem formulation. This section will define the specific problem addressed in this research and outline the objectives and constraints that guide the

development of the AD agent.

3.3.1 Environment

The agent's environment is the external world in which it must function. In this case, it is the simulated town by the CARLA simulator. As stated, there are three phases to this project; thus, this subsection is going to be divided in three parts. In each one is going to be represented the observation space, action space, and reward function, which are the three crucial aspects of defining an environment.

3.3.1.1 First Phase

This first phase consists in a more simple approach to the learning curve. This observation space can be seen in the equation 3.1, and it is the combination of a frame of the RGB sensor, as well as the ego vehicle's current position and target position (end goal).
Each episode runs for a fixed duration of 55 seconds. Given that the simulation runs at 30 frames per second (FPS), this results in a total of 1,650 frames per episode. However, the reinforcement learning agent interacts with the environment at a lower rate, approximately 14 steps per second. Therefore, the maximum number of decision steps per episode is around 770. This number is important to have in mind since without we lose control of the environment.

$$Obs_{space} = \begin{cases} \text{RGB image} : (360, 640, 3) \\ \text{Current Position} : (3,) \\ \text{Target Position} : (3,) \end{cases} \tag{3.1}$$

The action space is divided into discrete and continuous because of the two different algorithms used. It can be observed in the Table 3.4. In the continuous action space a particularly smart approach was conducted, this being the fusion of the brake and throttle values, which are separate in the CARLA simulator. When the value is higher than zero, the car accelerates, while if it isn't the car brakes.

Continuous Action Space		**Discrete Action Space**	
Actions	**Values**	**Actions**	**Index**
Steering	[-1.0, 1.0]	Accelerate	0
Throttle/Brake	[-1.0, 1.0]	Decelerate	1
		Turn Left	2
		Turn Right	3

Table 3.4: Continuous and Discrete Action Spaces.

The reward function employed in this project is designed to guide the AD agent towards optimal driving behavior by providing feedback based on several factors. The function calculates the reward as a weighted sum of multiple components, each addressing a specific aspect of driving. These components are speed adherence, collision avoidance, and minimizing sudden changes in throttle and steering (referred to as throttle and steering jerk). The weights for these components are defined in `ENV_REWARDS_LAMBDAS`. The function penalizes excessive

throttle and steering jerk to encourage smooth driving. Speed adherence is rewarded up to a limit, beyond which penalties are applied. Collisions and lane invasions result in significant penalties, and a small reward is given for continuous driving without incidents. This comprehensive reward function ensures that the agent learns to drive smoothly, safely, and efficiently. A mathematical representation of this function can be observed in the equation 3.2. This reward function does not take into account more complex scenarios such as curves, light poles, or even stop signs; it is merely focused on the driving aspect.

$$r = r_{\text{orientation}} + r_{\text{steering\_jerk}} + r_{\text{throttle\_brake\_jerk}} + r_{\text{collision}} + r_{\text{time\_driving}} \quad (3.2)$$

where:

$$r_{\text{orientation}} = \cos\left(v_{yaw} - w_{yaw}\right) * \pi/180 \quad (3.3)$$

$$r_{\text{steering\_jerk}} = \begin{cases} -2.0 & \text{if } s_{diff} > 0.2 \\ 0.0 & \text{if } s_{diff} <= 0.2 \end{cases} \quad (3.4)$$

$$r_{\text{throttle\_brake\_jerk}} = \begin{cases} -2.0 & \text{if } t_{diff} > 0.2 \\ 0.0 & \text{if } t_{diff} <= 0.2 \end{cases} \quad (3.5)$$

$$r_{\text{collision}} = \begin{cases} -10.0 & \text{if collision} \\ 0.0 & \text{if not collision} \end{cases} \quad (3.6)$$

$$r_{\text{time\_driving}} = \begin{cases} 5 * 10^{-5} & \text{if } s > 2.0 \\ 0.0 & \text{if } s <= 2.0 \end{cases} \quad (3.7)$$

r is the symbol for reward, which is the score used to evaluate the agent's performance. v_{yaw} is the yaw value of the vehicle's front vector, w_{yaw} is the waypoint's front vector yaw value, s_{diff} is the difference between the last steering value and the current one, while t_{diff} is the same but for the throttle/brake value. s is the speed of the ego vehicle.
In order to properly measure the agent's performance using this reward function, it is essential to determine the minimum and maximum value that the reward function may achieve. The maximum value that this function may get is 770 which happens when the agent is driving successfully without any error. The minimum reward possible needs to be subjective, because assuming the vehicle is driving at 100km/h the reward would be -88550 which would exaggerate the results, making them look much better than they actually are, therefore, the minimum reward possible is considered to be when the vehicle has a collision while having a

lot of jerk, which is -50.

3.3.1.2 Second Phase and Third Phase

The objective of the second and third phase is to address the shortcomings of the first phase. The third phase is similar to the second phase however the problem is simplified and variance reduced, since the project would not be complete in time otherwise. However, the observation space, action space and reward is identical between both phases.
While the initial phase focused solely on driving, these phases enhance the approach by incorporating destination information. This addition helps the agent understand its goals better. Moreover, the provided information is simplified for the agent's processing, consisting of individual values instead of complex arrays, making it easier for the agent to interpret. The RGB data remains unchanged. The updated observation space, which includes these simplified inputs, is represented in the equation 3.8.

$$Obs_{space} = \begin{cases} \text{RGB image} : (360, 640, 3) \\ \text{Distance to the target} : (1,) \\ \text{Distance to the nearest waypoint} : (1,) \\ \text{Ego vehicle's speed} : (1,) \end{cases} \tag{3.8}$$

The action space remains unchanged and can be seen in the Table 3.4.
The reward function aims to create a simpler yet more efficient reward system for the ego vehicle. This streamlined approach facilitates faster learning, requiring fewer episodes to reach optimal performance. The function integrates various factors, including speed, proximity to the target, adherence to waypoints, and collision avoidance. By balancing these elements, the vehicle is rewarded for maintaining appropriate speeds, staying on course, and reaching its target, while being penalized for collisions. This function is mathematically represented in Equation 3.9.
All calculations were performed manually to ensure a well-balanced reward function. The reward values were designed considering an entire episode duration of 30 seconds, which consists of approximately 430 iterations. Since the reward is calculated at each time step, it needs to be normalized by dividing by 430. This ensures that the total accumulated reward over the full episode does not exceed the intended maximum value. By leveraging this normalization factor, the function maintains consistency across different episode lengths and learning conditions.
The study by Pérez et al. [78] served as inspiration for this reward function.

$$r = r_{\text{speed}} + r_{\text{target}} + r_{\text{waypoint}} + r_{\text{collision}} \tag{3.9}$$

, where:

$$r_{\text{speed}} = \begin{cases} -\frac{15}{430} & \text{if } s < 2.0 \\ \frac{15}{430} & \text{if } 2 <= s <= 50.0 \\ -\frac{15}{430} & \text{if } s > 50.0 \end{cases} \quad (3.10)$$

$$r_{\text{target}} = \begin{cases} 100 & \text{if } d_t <= 5.0 \\ \frac{-7*d_t+395}{9*430} & \text{if } 5-0 < d_t <= 50.0 \\ -\frac{100-d_t}{10*430} & \text{if } d_t >= 100.0 \end{cases} \quad (3.11)$$

$$r_{\text{waypoint}} = \begin{cases} 5 & \text{if } d_w < 1.0 \\ 0 & \text{if } d_w >= 1.0 \end{cases} \quad (3.12)$$

$$r_{\text{collision}} = \begin{cases} -10 & \text{if collision} \\ 0 & \text{if not collision} \end{cases} \quad (3.13)$$

r represents the reward value, s is the variable containing the `speed` of the ego vehicle, d_t is the distance between the ego vehicle and the target position, d_w is the distance between the ego vehicle and the next waypoint, and n is the total number of steps per episode.
This phase has a much better formulated reward function, therefore it is much easier to get the minimum and maximum values. The minimum value is attributed to the vehicle being stopped for a long time and then having a collision after moving, therefore it's -25, the maximum value occurs when the vehicle reaches the destination by going through all the checkpoints, which is 120.

3.3.2 Agent

As noted in subsection 2.2.4, a policy serves as the agent's brain, making its structure crucial. To follow this project's methodology, two architectures (modular and end-to-end) were designed for phases 1 and 2, resulting in four structures in total. In phase 3, only the end-to-end architecture, similar to the one used in phase 2, was implemented. This subsection provides a detailed overview of these four structures.
Before being fed to the neural networks, the data is pre-processed, more precisely, the image, where it is resized from `(360, 640, 3)` to `(3, 224, 224)` and its values normalized, so instead of being between 0 and 255 it's between 0 and 1. This improves the learning process. The additional information is not processed, but it is concatenated so it can be fed into a single neural network.
Each phase represents a stage of development in this project. The first phase was a preliminary attempt, serving as a learning tool for RL. The second phase was intended to be the final implementation, but due to time constraints, a simplified version was created instead.

To address these limitations, a third phase was introduced, refining the approach within the available time frame.

3.3.2.1 First Phase

In this first phase's modular structure relies on two modules, the perception module, where an EfficientNet CNN pre-trained with the ImageNet dataset receives the RGB image data, and outputs a vector containing the most important features extracted from the image. The other data, i.e., the ego vehicle's position and the target position, are fed into a simple multilayer perceptron (MLP), which outputs their feature vector. Then both vectors are concatenated and fed into a more robust MLP, called final MLP, which outputs the action A_t in the case of PPO, or the Q-values in DQN. A graphical representation of this structure can be found in the Figure 3.10.
The end-to-end counterpart, instead of processing the image in a separate module, it processes everything in a single cohesive unit. At first glance it might not look significantly different from the modular approach, however, in this method, both the MLPs and the CNN are trained together, possibly resulting in a more efficient decision-making, however, at the cost of longer training time. The diagram for this structure can be observed in the Figure 3.11.

3.3.2.2 Second Phase

In this second phase, the policy architectures did not undergo massive changes. However, significant adjustments were made to the end-to-end method. The EfficientNet was replaced with a simpler custom CNN, and the additional information MLP was enhanced by adding more layers while reducing the output shape by half. At first glance, this might suggest a potential decrease in performance compared to the first phase. However, larger output vectors can lead to overfitting and make the model harder to train due to its increased size. By simplifying the structure, training becomes more light and manageable, while the custom CNN remains sufficient for the task. Another concern is that if the image feature vector is much larger than the information feature vector, the final MLP might ignore the additional information. Therefore, a more balanced approach between image and information feature vectors results in better overall performance. This structure can be seen in Figure 3.11.
The modular structure for the second phase remains similar to the first phase's counterpart, with the primary change being the enhancement of the first MLP by adding more layers while maintaining its output shape. This adjustment was necessary because the perception unit continued to use the pre-trained EfficientNet, which has an output size of 1024. Altering its output shape would require retraining the network, which was not feasible at the time. To minimize the size discrepancy, the additional information MLP retained its 256 output size. This structure is present in Figure 3.13.

3.3.2.3 Third Phase

The third phase used an end-to-end agent identical to the second phase. It can be observed in the Figure 3.12.

3.3.3 Objective

The agent's primary objective is to reach its destination in the shortest possible time while avoiding collisions and lane evasions. To achieve this, the agent must optimize its actions to maximize the reward function.

3.3.4 Learning Algorithms

The two learning algorithms used in this project are DQN and PPO, which were detailed in Section 2.2. These algorithms utilize different action spaces: DQN operates with a discrete action space, while PPO uses a continuous action space. The environment provided by the framework developed in this project is capable of handling both types of actions.

3.4 Conclusion

This chapter delves into the project's methodology, centered on comprehensive ADS testing within the CARLA simulator. Divided into two primary tasks, it involves implementing and testing various driving frameworks and training an RL-based end-to-end ADS using PyTorch. To address overfitting, the agent undergoes training in diverse CARLA towns and under different conditions. Emphasizing the significance of Software Quality, a meticulously designed test suite evaluates ADS performance across various scenarios, utilizing key metrics.

```
{
    "rgb_camera":{
        "image_size_x": 640,
        "image_size_y": 360,
        "fov": 90,
        "sensor_tick": 0.0,
        "location_x": 0.8,
        "location_y": 0.0,
        "location_z": 1.7
    },
    "lidar":{
        "channels": 32,
        "range": 50.0,
        "points_per_second": 56000,
        "rotation_frequency": 10.0,
        "upper_fov": 10.0,
        "lower_fov": -30.0,
        "sensor_tick": 0.0,
        "location_x": 0.8,
        "location_y": 0.0,
        "location_z": 1.7
    },
    "gnss":{
        "sensor_tick": 0.0,
        "location_x": 0.0,
        "location_y": 0.0,
        "location_z": 0.0
    },
    "collision":{
        "location_x": 0.0,
        "location_y": 0.0,
        "location_z": 0.0
    },
    "lane_invasion":{
        "location_x": 0.0,
        "location_y": 0.0,
        "location_z": 0.0
    }
}
```


Figure 3.8: Ego vehicle's sensors JSON example.

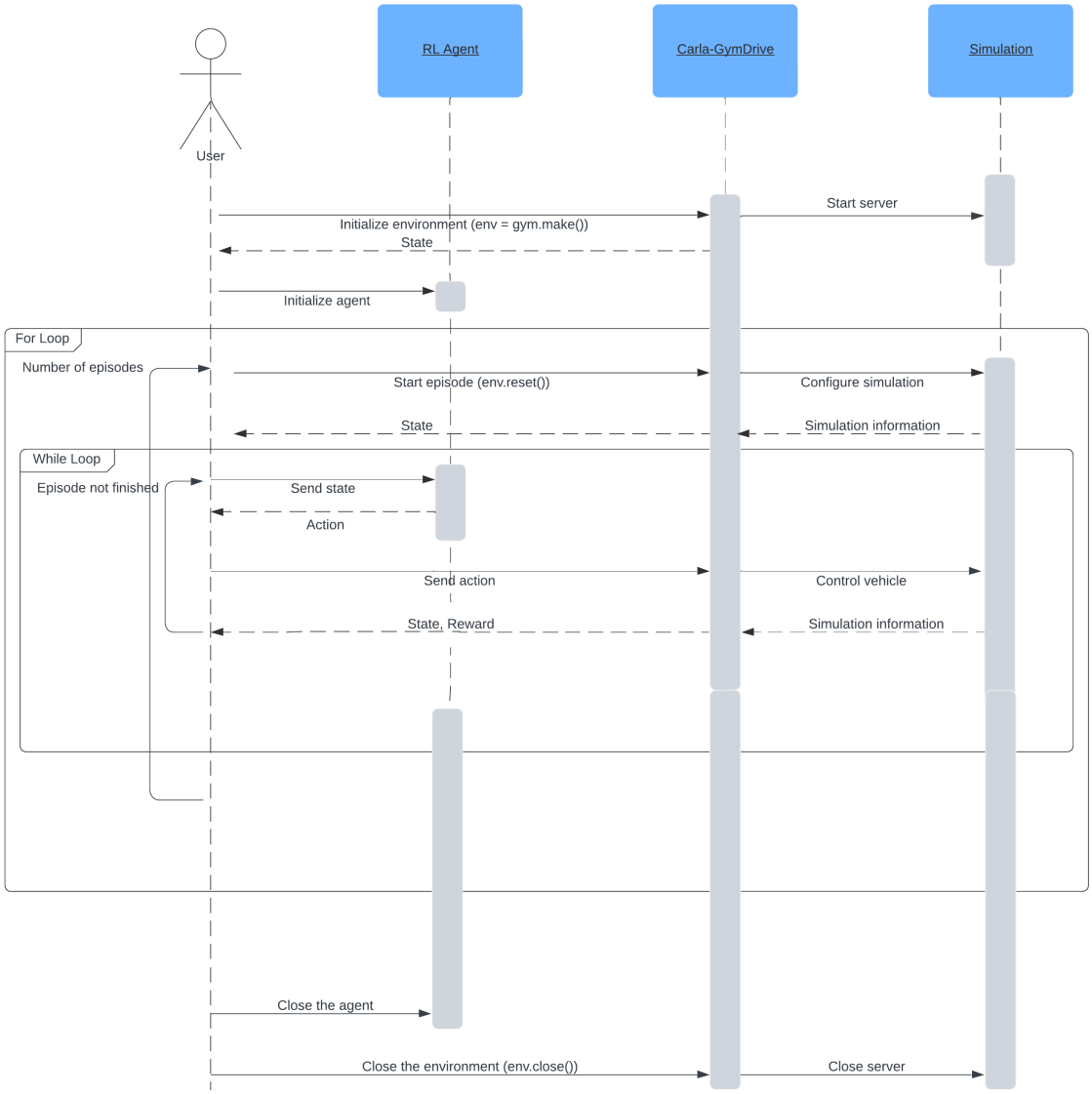


Figure 3.9: Sequence diagram for CARLA-GymDrive.

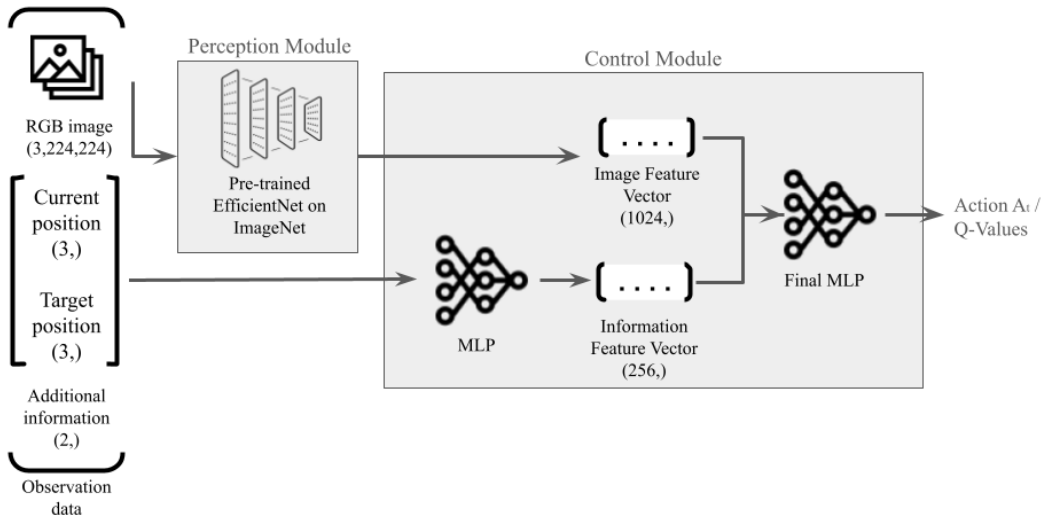


Figure 3.10: Phase one modular agent's structure.

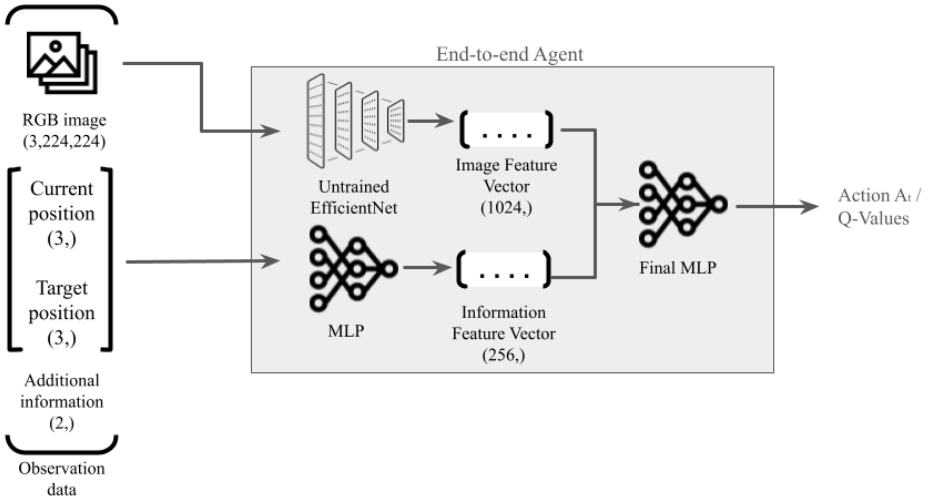


Figure 3.11: Phase one end-to-end agent's structure.

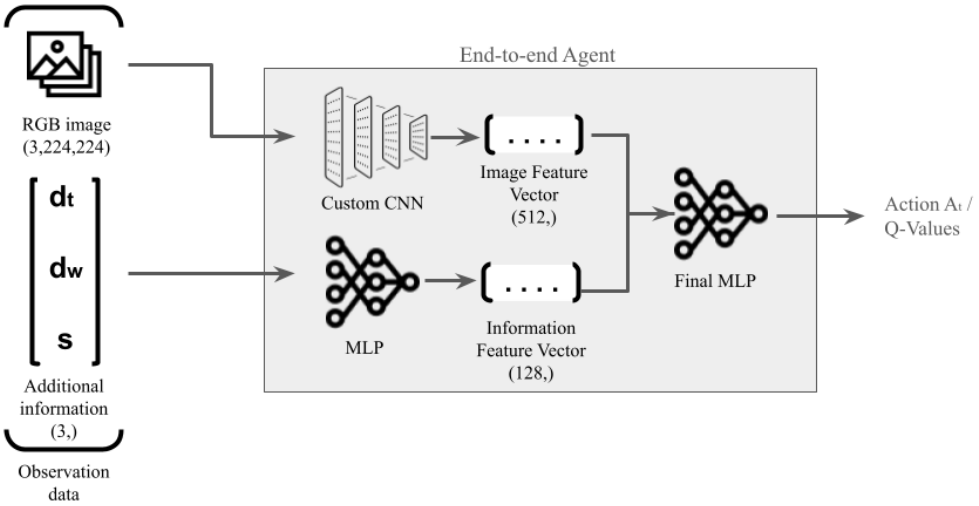


Figure 3.12: Phase two end-to-end agent's architecture.

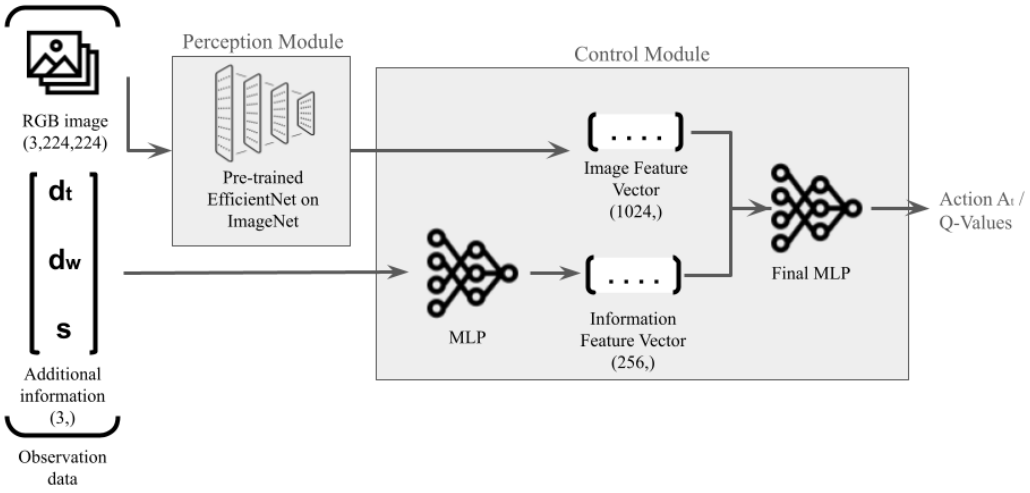


Figure 3.13: Phase two modular agent's structure.

Chapter 4

Results

This chapter presents the outcomes of the experiments conducted during this investigation. The results are divided into two initial phases, and later a third phase, which was introduced later as a means to achieve better training results by training one agent for an extended period of time. The first two phases are thoroughly analyzed both objectively and subjectively to provide insights into the effectiveness of the methods used, as well as the algorithms employed, and the overall impact on the agents' driving capabilities. The last phase is analyzed in a more standard RL evaluation, where the agent is trained and tested occasionally during training to get the results, thus this last phase's results being structured differently. Additionally, this chapter also performs a SWOT analysis of the CARLA-GymDrive framework, in order to easily show the tool's capabilities and drawbacks.

4.1 Investigation Results

In order to better compare the values and easily understand how good they are, they were normalized to be between zero and one using the function present in the Figure 4.1.

```
def normalize_reward(reward, min_reward, max_reward):
    # Normalize the reward to the range [0, 1]
    normalized_reward = (reward - min_reward) / (max_reward - min_reward)
    return normalized_reward
```

Figure 4.1: Normalization function used for the rewards.

4.1.1 Phase One

4.1.1.1 Objective Evaluation

The phase one objective evaluation results can be observed in the Table 4.2, and, for easier comparison, can also be observed in the Figure 4.2. The Table 4.1 shows how long the agents in this phase took to train 5000 episodes.

Agent	Time
End-to-end DQN	15h 58m
Modular DQN	15h 05m
End-to-end PPO	12h 45m
Modular PPO	11h 53m

Table 4.1: Training times for the phase one agents.

	DQN	PPO
End-to-end	0.403	0.069
Modular	0.401	0.073

Table 4.2: Objective Evaluations for the first phase.

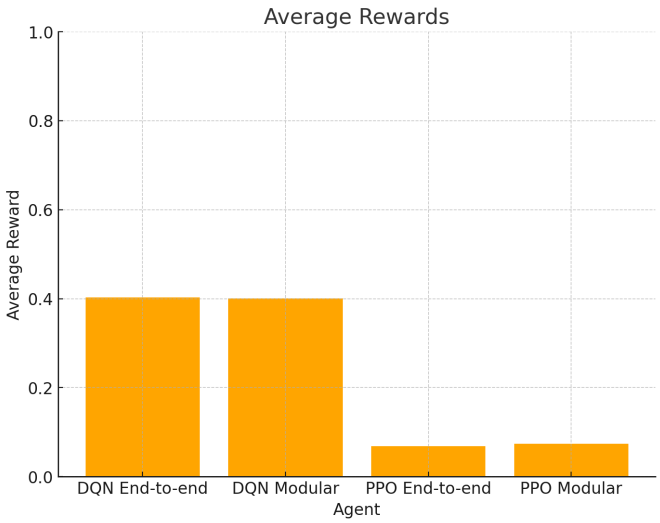


Figure 4.2: Phase one results gathered in a histogram.

4.1.1.2 Subjective Evaluation

This first phase highlights the flaws in the initial reward function design and how they negatively impacted the agent's behavior. Since reinforcement learning (RL) relies on the reward function to shape the agent's decisions, an improperly designed function can lead to unintended behaviors. In this case, the shortcomings of the reward function caused the agent either to remain stationary or to behave erratically.
Key observations from the testing phase include:

- There was no significant difference in performance between the end-to-end and modular agents. This likely occurred because the training duration was insufficient for these distinctions to emerge—typically, the modular approach should learn faster, while the end-to-end approach tends to achieve better long-term performance.

- The vehicle using the DQN algorithm did not move for the entire duration of the episode. This suggests that the reward function inadvertently encouraged the agent to remain still rather than incentivizing movement.

- Unlike DQN, the PPO agent immediately turned off the road, ending the episode prematurely. This behavior indicates that the agent failed to interpret the reward function effectively and was not actually optimizing it in a meaningful way. A key challenge in RL is that agents rely on experience to determine optimal actions. If an agent never explores certain actions—such as stopping—it may never learn their value. In the case of PPO, the continuous action space made it unlikely for the agent to randomly select exactly 0.0 (full stop), meaning it never learned that stopping might be a viable or even

optimal choice in some situations.

This phase underscores a fundamental principle of RL: for an agent to exhibit the desired behavior, the reward function must be carefully refined. Poorly designed rewards can lead to unintended strategies, making it crucial to iteratively adjust and fine-tune the function to align with the intended learning objectives.

4.1.2 Phase Two

4.1.2.1 Objective Evaluation

The second phase objective evaluation results are present in the Table 4.4, and, for easier comparison, can also be analyzed in the Figure 4.3. The Table 4.3 shows how long the agents in this phase took to train 15000 episodes.

Agent	Time
End-to-end DQN	37h 45m
Modular DQN	37h 37m
End-to-end PPO	72h 52m
Modular PPO	59h 24m

Table 4.3: Training times for the phase two agents.

	DQN	PPO
End-to-end	0.138	0.139
Modular	0.135	0.104

Table 4.4: Objective Evaluations for the second phase.

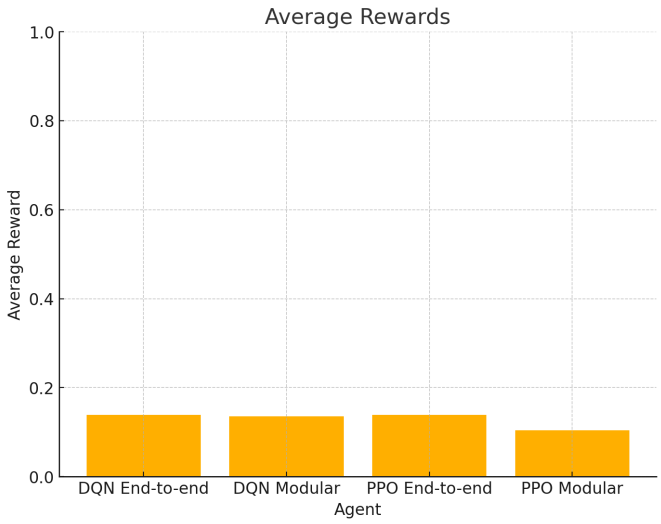


Figure 4.3: Phase two results gathered in a histogram.

4.1.2.2 Subjective Evaluation

The second phase demonstrated that training time played a critical role in the agent's performance. Unlike the first phase, where the reward function itself was the primary issue, this

phase showed that the agent's behavior was still far from optimal due to insufficient training. While the reward function encouraged progress along the road by rewarding checkpoint completion, the agents did not train long enough to fully learn how to drive properly.
Key observations from this phase:

- The DQN agents exhibited some driving behavior, moving around briefly before veering off the road. This suggests that they had begun to learn basic driving patterns but were still far from successfully completing any scenario. The limited training time prevented them from refining their driving skills further. No significant differences were observed between the end-to-end and modular architectures.
- In contrast to phase 1, where PPO agents moved erratically off the road, they now exhibited an overly cautious approach. Despite the reward function encouraging movement along the road by reaching checkpoints, the agents barely moved, slowly drifting to the side and wasting a large portion of the episode. This behavior can be attributed to the stochastic nature of RL: since the agent never randomly attempted to move straight on the straight roads during training, it failed to discover that doing so would significantly increase its reward. As with DQN, no noticeable differences were observed between the end-to-end and modular methods.

This phase highlights the importance of sufficient training time in reinforcement learning. Even with a well-designed reward function, an agent must explore enough scenarios to learn the best actions. Without adequate training, the agent may settle into suboptimal behaviors, unable to fully utilize the reward signals designed to guide it toward the desired performance.

4.1.3 Phase 3

Phase two showed that it is possible to train an end-to-end model, however it would need to train for a longer time. Therefore, in order to get good results, a third and final phase was conducted in order to achieve it. Using the second phase's configuration and also by reducing any possible variance, which would exponentially increase training times, by training the ego vehicle in only one scenario, without any traffic nor weather variation. This would make the agent overfitted, but it would prove that it is possible to have a functional end-to-end agent, with more training time. An overview of this phase can be found in the table 4.5.

Parameter	**Value**
Architecture	End-to-end
Algorithm	PPO
Episodes	120,000
Training Time	480 hours
Average Result	21.364
Average Result Normalized	0.320
Average Checkpoints	4 out of 5
Scenario Completed	7 times

Table 4.5: Overview of the third phase

The PPO algorithm was chosen for this agent for two key reasons. Firstly, its continuous action space is better suited to the complexities of AD compared to discrete action spaces,

enabling finer control and more realistic decision-making. Secondly, despite its potential, PPO has been underrepresented in the current body of AD research, where DQN predominates. This presented an opportunity to explore PPO's capabilities in this domain and introduce a novel approach.
The agent underwent training for 120,000 episodes over 480 hours, paralleling the training duration used in the study by [78]. The results demonstrated clear evidence of learning: the agent successfully navigated four out of five checkpoints on average and fully completed the scenario on seven occasions. These outcomes confirm that the agent was indeed making progress, with noticeable improvements in performance over time.
However, when compared to the study by Pérez et al., it becomes apparent that the agent's development was less advanced at this stage. While both studies utilized the same number of episodes, Pérez's agent exhibited more robust performance across multiple scenarios. This disparity underscores the inherent challenges of training end-to-end models, which require significantly more time to reach comparable levels of proficiency. The slower learning curve observed here highlights the extensive training demands of end-to-end approaches, reinforcing the conclusion that, despite their promise, such models face substantial obstacles before they can be viably deployed in real-world autonomous driving scenarios.

4.1.4 Final Results Discussion

The final results of this study reveal several critical insights and challenges in developing ADSs using RL.
In the first phase, the exaggerated scores for the DQN algorithm despite being stationary indicate a poorly designed reward function. This issue highlights the inadequacy of the reward mechanism, which failed to penalize inactivity effectively. This phase showed the importance of a well defined reward function and acted as a learning tool for the results obtained in the third phase, where the refined reward function yielded more meaningful agent behavior.
The difficulty in defining a minimum reward value in the first phase underscores the unbalanced and naive nature of the initial reward structure. This imbalance causes prolonged training times and hinders convergence, as the agent might struggle to learn meaningful driving behaviors within an inconsistent reward framework. The lack of a well-defined minimum reward value made it challenging to set clear objectives for the agent, resulting in inefficient training and suboptimal performance.
The prolonged training time of agents in the second phase can be attributed to their initial strategy of remaining stationary to maximize rewards. It took considerable time for the agent to realize that driving, rather than stopping, was necessary to achieve higher rewards. This observation points that, despite the reward function being well-balanced, luck is still an important factor in RL training, and thus the agent needed to find the correct set of actions to maximize reward in an infinite space of possible actions.
In the final phase, the agent made consistent progress, reaching an average of four out of five checkpoints per episode. While this showed that the agent had learned to navigate more effectively, it struggled to consistently complete entire scenarios. Crashes near the finish line often resulted in sharp reductions in its final scores. However, the fact that the agent

improved over time, managed to complete the scenario seven times, and achieved higher scores compared to earlier phases proves that the end-to-end architecture is capable of learning—albeit slowly.

These results underscore a broader issue: the immense training time and computational resources required to make end-to-end architectures viable for complex tasks like autonomous driving. The extensive training durations, as shown in Tables 4.1, 4.3, and 4.5, combined with hardware limitations and time constraints, significantly hindered the ability to achieve better outcomes in this project.

While the project did not result in a highly reliable end-to-end ADS, it succeeded in several key areas:

- Proof of Concept: The development of a functional end-to-end agent that demonstrated clear learning and performance improvements over time.
- Reward Function Insights: A deeper understanding of how critical reward design is to agent behavior and learning efficiency.
- Exploration Challenges: Valuable observations on the role of randomness and exploration in RL training, especially in complex environments.

Ultimately, while the results fell short of creating a robust, deployable ADS, the project provided meaningful contributions to understanding both the potential and the current limitations of end-to-end RL in autonomous driving. It serves as a testament to the architecture's theoretical promise but also as a realistic acknowledgment of the challenges that still need to be overcome.

4.2 SWOT Analysis of the CARLA-GymDrive Framework

Beyond the investigative results, it is essential to emphasize the success of the developed tool. The CARLA-GymDrive framework proved to be highly robust, supporting the training and testing of RL agents for hundreds of hours without system failures. Its reliability and effectiveness have been validated not only through this project but also by external users who have reported positive experiences with the framework. The tool has garnered significant attention within the research community, as evidenced by the growing number of stars and forks in its official repository—an indicator of its increasing relevance and adoption by researchers worldwide.

The tool also managed to implement every single functional requirement explicit in sections 3.2.4.1 and 3.2.4.2. The only bottleneck it has is the simulator's performance; but that shouldn't be a problem in modern computers used in RL experiments.

To further evaluate its potential and limitations, a comprehensive SWOT analysis was conducted, offering a structured assessment of the framework's strengths, weaknesses, opportunities, and threats. This analysis provides valuable insights into its current capabilities and future prospects, positioning CARLA-GymDrive as a meaningful contribution to the field of autonomous driving research.

Strengths

- **User-Friendly Interface:** CARLA-GymDrive provides an intuitive and efficient platform for conducting RL experiments in AD. Its gymnasium-like environment abstracts the complexities of the CARLA simulator, allowing researchers to focus on developing and fine-tuning their RL algorithms without extensive coding.
- **Comprehensive Features:** The framework includes scenario configuration, custom sensor configuration, and compatibility with training libraries like Stable-Baselines3. This versatility ensures that the training and test suites are adequate and can be adjusted intuitively.
- **Enhanced Productivity:** By simplifying the interaction with the simulation environment, CARLA-GymDrive significantly reduces development time and increases researchers' productivity. This allows for more efficient experimentation and quicker iteration cycles.
- **Educational Resource:** The framework serves as an excellent educational tool, helping students and new researchers understand and experiment with AD concepts and RL techniques.

Weaknesses

- **Niche Audience:** While CARLA-GymDrive is highly valuable within the AD research community, its appeal and utility may be limited outside this domain. The framework may not attract significant attention from researchers in other fields outside AD.
- **Dependency on CARLA Simulator:** The framework's reliance on the CARLA simulator means that any limitations or issues within CARLA can impact the performance and usability of CARLA-GymDrive.

Opportunities

- **Growing Interest in AD:** The AD field is experiencing steady growth, with increasing numbers of researchers and developers dedicating efforts to this area. This trend provides an opportunity for CARLA-GymDrive to become a pivotal tool in the field.
- **Community and Collaboration:** As more researchers recognize the benefits of using CARLA-GymDrive, its adoption is likely to increase. This can lead to a larger community of users who contribute to the framework's development and enhancement.
- **Advancements in RL:** Ongoing advancements in RL techniques can be leveraged within CARLA-GymDrive to develop more sophisticated and effective AD agents.

Threats

- **Technological Changes:** Rapid advancements in technology and changes in the AD landscape may require continuous updates and improvements to CARLA-GymDrive to maintain its relevance and effectiveness.

Chapter 5

Conclusion

5.1 Publications

During the course of my investigation I was able to publish a scientific article titled *Enhancing Autonomous Vehicles: An Experimental Analysis of Path Planning and Decision-Making Processes through Simulink-Based Architecture* [80], which objective was to investigate the feasibility of meta-heuristic models in the decision-making module of an ADS. Since a meta-heuristic is a way to find solutions to optimization problems, using it to determine the best path between a set of possible paths is an idea with great potential. These experiments were developed in the initial stages of my thesis, however they aren't a part of this document since I was more interested in researching RL and its possibilities.
Also, a tool paper regarding the CARLA-GymDrive framework was published in the SoftwareX journal under the title *CARLA-GymDrive: Autonomous driving episode generation for the Carla simulator in a gym environment* [81], and it was met with great enthusiasm.

5.2 Limitations

During the time of the development of this project, some limitations were encountered despite the advancements made:

- **Computational Constraints**: The provided computer, although high-performing, was insufficient for training a fully-fledged AD agent. Specifically, it lacked the necessary graphical power to utilize more advanced CNNs such as RESNET-50. This limitation affected the depth and robustness of the models that could be developed and tested.
- **Limited Access to Resources**: The time allocated with the borrowed computer was not enough to train a comprehensive agent. AD agents, particularly those trained using RL, require extensive computational resources and time to achieve optimal performance. The limited access restricted the ability to conduct thorough and extended training sessions, thereby impacting the overall quality and capability of the developed agent.

These limitations made it impossible for the second phase to be a success and are the reason a third phase had to be created.

5.3 Future Work

Building on the current foundation, some avenues for future work can be pursued to enhance this project:

- **Multi-Agent Training Compatibility**: One significant improvement to the CARLA-GymDrive would be to make the framework compatible with multi-agent training. This would allow for the simulation and study of more complex driving scenarios involving multiple autonomous agents. Currently, this enhancement is constrained by the available computational resources, but future iterations of the project should aim to overcome this limitation.
- **Train a fully-fledged AD agent in different weather conditions**: Currently, adverse weather conditions are one of the main obstacles to AD systems. By leveraging the CARLA-GymDrive framework, it's possible to train and test an agent in various weather presets and also to do more complex actions, such as overtakes, and respond to light poles and stop signs. This project only aimed to create a simple agent that could learn how to drive due to the other main objectives, but future iterations could aim to exclusively create a more complex agent.

5.4 Final Conclusion

This thesis developed a RL training and testing framework for AVs using the CARLA simulator. The framework, CARLA-GymDrive, has gained recognition within the research community. This highlights its practical value. It serves as both a research tool and an educational platform, enabling students and researchers to explore AD concepts and RL techniques without needing to dive too deeply into the code's complexity.
Throughout this project, several critical challenges in the development of end-to-end AD systems were identified. The research highlighted the pivotal role of reward function design in influencing agent behavior and learning efficiency, as well as the inherent complexities associated with training end-to-end ADSs. These findings emphasize the significant computational demands and prolonged training times required to achieve reliable and effective AD solutions. Despite these challenges, the framework demonstrated the potential of end-to-end architectures. The agent's gradual performance improvements throughout the study provide clear evidence that, with sufficient time and resources, meaningful advancements can be achieved. However, it also became evident that end-to-end models currently require substantially more extensive training compared to traditional modular approaches, limiting their immediate practicality for real-world applications.
This research contributes valuable insights to the evolving field of autonomous driving, particularly in the context of RL-based systems. The lessons learned from this work offer a solid foundation for future research aimed at refining and enhancing end-to-end architectures. By addressing the identified limitations, this thesis paves the way for continued advancements in AD technologies, ultimately supporting the development of safer, more efficient autonomous vehicles in the future.

Bibliography

[1] S. Esselborn, "Introduction: Auto-mobilities. automation, safety and responsibility in the history of mobility," *Technikgeschichte*, vol. 87, no. 1, pp. 3–10, 2020. 1

[2] C. Mentiko, "Autonomous vehicle funding stuck in neutral," 2023, published on Crunchbase News. [Online]. Available: https://news.crunchbase.com/transportation/autonomous-vehicle-startups-vc-funding/ 1

[3] S. Singh, "Critical reasons for crashes investigated in the national motor vehicle crash causation survey," *Art and Design Review*, vol. 10, no. 4, 2015, https://www.scirp.org/reference/referencespapers?referenceid=3332499. 1

[4] P. Kyriakopoulos, P. Jaworski, and S. Kanarachos, "Digicav project: Exploring a test-driven approach in the development of connected and autonomous vehicles," in *2019 8th IEEE International Conference on Connected Vehicles and Expo, ICCVE 2019 - Proceedings*, 2019, pp. 1–6. 1

[5] P. Kaur, S. Taghavi, Z. Tian, and W. Shi, "A survey on simulators for testing self-driving cars," 2021. 1

[6] P. S. Chib and P. Singh, "Recent advancements in end-to-end autonomous driving using deep learning: A survey," 2023. 1, 16

[7] M. Towers, J. K. Terry, A. Kwiatkowski, J. U. Balis, G. d. Cola, T. Deleu, M. Goulão, A. Kallinteris, A. KG, M. Krimmel, R. Perez-Vicente, A. Pierré, S. Schulhoff, J. J. Tai, A. T. J. Shen, and O. G. Younis, "Gymnasium," Mar. 2023. [Online]. Available: https://zenodo.org/record/8127025 2, 20

[8] E. Yurtsever, J. Lambert, A. Carballo, and K. Takeda, "A survey of autonomous driving: Common practices and emerging technologies," *IEEE Access*, vol. 8, pp. 58 443–58 469, 2019. 5, 7, 8

[9] T. Guardian, "Cruise recalls all self-driving cars after grisly accident and california ban," *The Guardian*, 2023, https://www.theguardian.com/technology/2023/nov/08/cruise-recall-self-driving-cars-gm. 5

[10] T. T. R. L. I. P. F. Siddiqui and I. Uraizee, "The final 11 seconds of a fatal tesla autopilot crash," *The Washington Post*, 2023, https://www.theguardian.com/technology/2023/nov/08/cruise-recall-self-driving-cars-gm. 5, 6

[11] D. Golson, "We put our blind faith in mercedes-benz's first-of-its-kind autonomous drive pilot feature," *The Verge*, 2023, published on September 27, 2023. 6

[12] Z. Le Hong and N. Zimmerman, "Air quality and greenhouse gas implications of autonomous vehicles in vancouver, canada," *Transportation Research Part D: Transport and Environment*, vol. 90, p. 102676, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1361920920308609 6

[13] U. Nations, *World Population Prospects 2022*. United Nations Department of Economic and Social Affairs, Population Division, 2022. 7

[14] S. Kuutti, S. Fallah, K. Katsaros, M. Dianati, F. Mccullough, and A. Mouzakitis, "A survey of the state-of-the-art localization techniques and their potentials for autonomous vehicle applications," *IEEE Internet of Things Journal*, vol. 5, no. 2, pp. 829–846, 2018. 7

[15] A. Chalvatzaras, I. Pratikakis, and A. Amanatiadis, "A survey on map-based localization techniques for autonomous vehicles," *IEEE transactions on intelligent vehicles*, vol. 8, pp. 1574–1596, 2023. 7

[16] G. Bresson, Z. Alsayed, L. Yu, and S. Glaser, "Simultaneous localization and mapping: A survey of current trends in autonomous driving," *IEEE Transactions on Intelligent Vehicles*, vol. 2, no. 3, pp. 194–220, 2017. 7

[17] R. W. Wolcott and R. M. Eustice, "Visual localization within lidar maps for automated urban driving," in *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2014, pp. 176–183. 7

[18] D. Gruyer, R. Belaroussi, and M. Revilloud, "Accurate lateral positioning from map data and road marking detection," *Expert Systems With Applications*, vol. 43, pp. 1–8, 2016. 7

[19] R. Tao, W. Han, Z. Qiu, C. zhong Xu, and J. Shen, "Weakly supervised monocular 3d object detection using multi-view projection and direction consistency," 2023. 7

[20] Y. Peng, "Deep learning for 3d object detection and tracking in autonomous driving: A brief survey," 2023. 7

[21] P. Shunmuga Perumal, Y. Wang, M. Sujasree, S. Tulshain, S. Bhutani, M. K. Suriyah, and V. U. Kumar Raju, "Lanescannet: A deep-learning approach for simultaneous detection of obstacle-lane states for autonomous driving systems," *Expert Systems with Applications*, vol. 233, p. 120970, 2023. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0957417423014720 7

[22] W. M. D. Chia, S. L. Keoh, C. Goh, and C. Johnson, "Risk assessment methodologies for autonomous driving: A survey," *IEEE Transactions on Intelligent Transportation Systems*, vol. 23, no. 10, pp. 16 923–16 939, 2022. 7

[23] K. A. Madala and M. Solmaz, "Scenario-based risk quantification approach for assuring safety in autonomous vehicles," *SAE technical paper series*, 2023. 7

[24] W. Yu, Y. Qian, J. Xu, H. Sun, and J. Wang, "Driving decisions for autonomous vehicles in intersection environments: Deep reinforcement learning approaches with risk assessment," *World Electric Vehicle Journal*, vol. 14, p. 79, 03 2023. 7

[25] Y. Xiao, B. Li, H. Li, A. Liu, H. Lyu, H. Pan, and D. Li, "Driving style recognition of interacting vehicles in multiple scenarios," in *2022 6th CAA International Conference on Vehicular Control and Intelligence (CVCI)*, 2022, pp. 1–6. 7

[26] S. Malik, M. A. Khan, H. El-Sayed, J. Khan, and O. Ullah, "How do autonomous vehicles decide?" *Sensors*, vol. 23, no. 1, 2023. [Online]. Available: https://www.mdpi.com/1424-8220/23/1/317 7

[27] P. Wang, D. Liu, J. Chen, H. Li, and C.-Y. Chan, "Decision making for autonomous driving via augmented adversarial inverse reinforcement learning," 2021. 7

[28] F. Meng, Z. Wu, and Z. Jia, "Research on the influence of automotive instrumentation hmi design on driving behavior," in *Human-Computer Interaction*, M. Kurosu and A. Hashizume, Eds. Cham: Springer Nature Switzerland, 2023, pp. 468–484. 7

[29] M. Gerla, E.-K. Lee, G. Pau, and U. Lee, "Internet of vehicles: From intelligent grid to autonomous cars and vehicular clouds," in *2014 IEEE World Forum on Internet of Things (WF-IoT)*, 2014, pp. 241–246. 7

[30] S. Kumar, L. Shi, N. Ahmed, S. Gil, D. Katabi, and D. Rus, "Carspeak: A content-centric network for autonomous driving," *SIGCOMM Comput. Commun. Rev.*, vol. 42, no. 4, p. 259–270, aug 2012. [Online]. Available: https://doi.org/10.1145/2377677.2377724 7

[31] M. Gerla, "Vehicular cloud computing," *2012 The 11th Annual Mediterranean Ad Hoc Networking Workshop (Med-Hoc-Net)*, pp. 152–155, 2012. [Online]. Available: https://api.semanticscholar.org/CorpusID:1719123 7

[32] M. Amadeo, C. Campolo, and A. Molinaro, "Information-centric networking for connected vehicles: a survey and future perspectives," *IEEE Communications Magazine*, vol. 54, no. 2, pp. 98–104, 2016. 7

[33] J.-F. Bonnefon, A. Shariff, and I. Rahwan, "The social dilemma of autonomous vehicles," *Science*, vol. 352, no. 6293, p. 1573–1576, Jun. 2016. [Online]. Available: http://dx.doi.org/10.1126/science.aaf2654 8

[34] "Philosophical and legal approach to moral settings in autonomous vehicles: An evaluation," pp. 95–114, 2023. 8

[35] O. Saoudi, I. Singh, and H. Mahyar, "Autonomous vehicles: Open-source technologies, considerations, and development," *Adv. Artif. Intell. Mach. Learn.*, vol. 3, pp. 669–692, 2022. [Online]. Available: https://api.semanticscholar.org/CorpusID:246634971 8

[36] S. Campbell, N. O' Mahony, L. Krpalkova, D. Riordan, J. Walsh, A. Murphy, and C. Ryan, "Sensor technology in autonomous vehicles : A review," pp. 1–4, 06 2018. 8

[37] J. Z. Varghese and R. G. Boone, "Overview of autonomous vehicle sensors and systems," 2015. [Online]. Available: https://api.semanticscholar.org/CorpusID:2028153 8

[38] Y. Zhang, A. Carballo, H. Yang, and K. Takeda, "Perception and sensing for autonomous vehicles under adverse weather conditions: A survey," *ISPRS Journal of Photogrammetry and Remote Sensing*, vol. 196, pp. 146–177, 2023. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0924271622003367 8

[39] T. Fersch, A. Buhmann, A. Koelpin, and R. Weigel, "The influence of rain on small aperture lidar sensors," in *2016 German Microwave Conference (GeMiC)*, 2016, pp. 84–87. 8

[40] F. Reway, W. Huber, and E. Ribeiro, "Test methodology for vision-based adas algorithms with an automotive camera-in-the-loop," pp. 1–7, 09 2018. 8

[41] S. Zang, M. Ding, D. Smith, P. Tyler, T. Rakotoarivelo, and M. A. Kaafar, "The impact of adverse weather conditions on autonomous vehicles: How rain, snow, fog, and hail affect the performance of a self-driving car," *IEEE Vehicular Technology Magazine*, vol. 14, no. 2, pp. 103–111, 2019. 8

[42] V. Sharma and S. Sergeyev, "Range detection assessment of photonic radar under adverse weather perceptions," *Optics Communications*, vol. 472, p. 125891, 04 2020. 8

[43] R. M. Rasmussen, J. Vivekanandan, J. Cole, B. Myers, and C. Masters, "The estimation of snowfall rate using visibility," *Journal of Applied Meteorology and Climatology*, vol. 38, pp. 1542–1563, 1999. 8

[44] N. Rawashdeh, J. Bos, and N. Abu-Alrub, "Drivable path detection using cnn sensor fusion for autonomous driving in the snow," p. 5, 04 2021. 8

[45] C. Brunner, T. Peynot, T. Vidal-Calleja, and J. Underwood, "Selective combination of visual and thermal imaging for resilient localization in adverse conditions: Day and night, smoke and fire," *Journal of Field Robotics*, vol. 30, no. 4, pp. 641–666, 2013. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/rob.21464 8

[46] K. Qian, S. Zhu, X. Zhang, and L. E. Li, "Robust multimodal vehicle detection in foggy weather using complementary lidar and radar signals," in *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021, pp. 444–453. 8

[47] Y. Hamzeh and S. A. Rawashdeh, "A review of detection and removal of raindrops in automotive vision systems," *Journal of Imaging*, vol. 7, no. 3, 2021. [Online]. Available: https://www.mdpi.com/2313-433X/7/3/52 8

[48] R. Quan, X. Yu, Y. Liang, and Y. Yang, "Removing raindrops and rain streaks in one go," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2021, pp. 9147–9156. 8

[49] W. Wang, X. You, L. Chen, J. Tian, F. Tang, and L. Zhang, "A scalable and accurate de-snowing algorithm for lidar point clouds in winter," *Remote Sensing*, vol. 14, no. 6, 2022. [Online]. Available: https://www.mdpi.com/2072-4292/14/6/1468 8

[50] H.-P. Schoener, "The role of simulation in development and testing of autonomous vehicles," in *Procedings*, 09 2017, doi: 10.1007/978-3-658-21194-3_82. 9

[51] M. C. Figueiredo, R. J. F. Rossetti, R. A. M. Braga, and L. P. Reis, "An approach to simulate autonomous vehicles in urban traffic scenarios," in *2009 12th International IEEE Conference on Intelligent Transportation Systems*, 2009, pp. 1–6. 9

[52] S. Ergün, "A study on multi-agent reinforcement learning for autonomous distribution vehicles," *Iranian Journal of Computer Science*, vol. 6, pp. 297–305, 2023. [Online]. Available: https://doi.org/10.1007/s42044-023-00140-1 10

[53] P. Yadav, A. Mishra, and S. Kim, "A comprehensive survey on multi-agent reinforcement learning for connected and automated vehicles," *Sensors*, vol. 23, no. 10, 2023. [Online]. Available: https://www.mdpi.com/1424-8220/23/10/4710 10

[54] L. Curiel-Ramirez, R. Ramirez-Mendoza, J. Izquierdo-Reyes, R. Bustamante-Bello, and S. Navarro Tuch, "Hardware in the loop framework proposal for a semi-autonomous car architecture in a closed route environment esci-journal," *International Journal for Interactive Design and Manufacturing (IJIDeM)*, 12 2019. 10

[55] A. Tampuu, T. Matiisen, M. Semikin, D. Fishman, and N. Muhammad, "A survey of end-to-end driving: Architectures and training methods," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 33, no. 4, pp. 1364–1384, 2022. 10, 11

[56] B. Argall, S. Chernova, M. Veloso, and B. Browning, "A survey of robot learning from demonstration," *Robotics and Autonomous Systems*, vol. 57, pp. 469–483, 05 2009. 11

[57] A. Vellido, "The importance of interpretability and visualization in machine learning for applications in medicine and health care," *Neural Computing and Applications*, vol. 32, pp. 18 069–18 083, 2020. [Online]. Available: https://doi.org/10.1007/s00521-019-04051-w 11

[58] L. Chi and Y. Mu, "Deep steering: Learning end-to-end driving model from spatial and temporal visual cues," 2017. 11

[59] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, 2nd ed. Cambridge, MA: MIT Press, 2018. 11

[60] T. Simonini and O. Sanseviero, "The hugging face deep reinforcement learning class," https://github.com/huggingface/deep-rl-class, 2023. 11

[61] J. Achiam, "Spinning Up in Deep Reinforcement Learning," 2018. 11

[62] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," 2017. 15

[63] K. Renz, K. Chitta, O.-B. Mercea, A. S. Koepke, Z. Akata, and A. Geiger, "Plant: Explainable planning transformers via object-level representations," 2022. 17

[64] C. Hubschneider, A. Bauer, M. Weber, and J. Zöllner, "Adding navigation to the equation: Turning decisions for end-to-end vehicle control," 10 2017. 17

[65] J. Hossain, "Autonomous driving with deep reinforcement learning in carla simulation," 2023. 17

[66] Y. Wu, S. Liao, X. Liu, Z. Li, and R. Lu, "Deep reinforcement learning on autonomous driving policy with auxiliary critic network," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 34, no. 7, pp. 3680–3690, 2023. 18

[67] T.-Y. Lin, M. Maire, S. Belongie, L. Bourdev, R. Girshick, J. Hays, P. Perona, D. Ramanan, C. L. Zitnick, and P. Dollár, "Microsoft coco: Common objects in context," 2015. 18

[68] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, and B. Schiele, "The cityscapes dataset for semantic urban scene understanding," 2016. 18

[69] P. Wang, X. Huang, X. Cheng, D. Zhou, Q. Geng, and R. Yang, "The apolloscape open dataset for autonomous driving and its application," *IEEE transactions on pattern analysis and machine intelligence*, 2019. 18

[70] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for autonomous driving? the kitti vision benchmark suite," in *2012 IEEE Conference on Computer Vision and Pattern Recognition*, 2012, pp. 3354–3361. 18

[71] H. Schafer, E. Santana, A. Haden, and R. Biasini, "A commute in data: The comma2k19 dataset," 2018. 19

[72] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, 2009, pp. 248–255. 19

[73] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, "Carla: An open urban driving simulator," 2017. 19

[74] P. Cai, Y. Lee, Y. Luo, and D. Hsu, "Summit: A simulator for urban driving in massive mixed traffic," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2020, pp. 4023–4029. 19

[75] G. Rong, B. H. Shin, H. Tabatabaee, Q. Lu, S. Lemke, M. Možeiko, E. Boise, G. Uhm, M. Gerow, S. Mehta *et al.*, "Lgsvl simulator: A high fidelity simulator for autonomous driving," *arXiv preprint arXiv:2005.03778*, 2020. 19

[76] "Deepdrive," https://deepdrive.io/, accessed: 2024-07-04. 20

[77] K. Qing, Y. Hao, Y. Yuan, Y. Ma, and W. Zhang, "Macad-gym: Multi-agent learning environment for autonomous driving research," *arXiv preprint arXiv:1911.04175*, 2019, accessed: 2024-07-04. [Online]. Available: https://arxiv.org/abs/1911.04175 20

[78] . Pérez-Gil, R. Barea, E. López-Guillén, L. Bergasa, C. Gómez-Huélamo, R. Gutierrez, and A. Díaz, "Deep reinforcement learning based control for autonomous vehicles in carla," *Multimedia Tools and Applications*, vol. 81, 01 2022. 23, 37, 49

[79] E. Yurtsever, L. Capito, K. Redmill, and U. Ozgune, "Integrating deep reinforcement learning with model-based path planners for automated driving," in *2020 IEEE Intelligent Vehicles Symposium (IV)*, 2020, pp. 1311–1316. 23

[80] . Morgado, C. Gonçalves, and N. Pombo, "Enhancing autonomous vehicles: An experimental analysis of path planning and decision-making processes through simulink-based architecture," 10 2023, pp. 1–9. 53

[81] Ângelo Miguel Rodrigues Morgado and N. G. C. C. Pombo, "Carla-gymdrive: Autonomous driving episode generation for the carla simulator in a gym environment," *SoftwareX*, vol. 28, p. 101955, 2024. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S235271102400325X 53