الجمهورية الجزَائِرية الديمقرَاطية الشعبية
PEOPLE'S DEMOCRATIC REPUBLIC OF ALGERIA
وزَارة التعليم العَالي و البحث العلمي
MINISTRY OF HIGHER EDUCATION AND SCIENTIFIC RESEARCH
جَامعة عمَار ثليجي بِالأَغوَاط
UNIVERSITY OF AMAR TELIDJI LAGHOUAT

Faculty of Science
Department of Computer Science

Domain: Mathematics and Computer Science
Field: Computer Science
Option: Information and Decision Systems

Multi-Agent Reinforcement Learning for Autonomous Vehicles (Use case: Highway on-Ramp Merging)

Submitted By:
Mahmoud Taouti

Defended publicly in front of a jury composed of:

Pr. Mustapha BOUAKKAZ	Professor	President
Dr. Laradj CHELLAMA	MAA	Examiner
Dr. Lakhdar KECHNA	MAA	Examiner
Dr. Mohamed El Amine AMEUR	MAA	Supervisor

2022/2023

Abstract

We focuses on the application of multi-agent reinforcement learning (MARL) to address the complex task of on-ramp merging for autonomous vehicles (AVs). The main objective is to develop a collaborative policy that enables AVs on both the ramp and highway to effectively merge while avoiding collisions and minimizing traffic trip time delay. A centralized MARL framework is implemented, utilizing a single-agent reinforcement learning (RL) framework as the foundation. The framework is designed to handle dynamic traffic scenarios and employs centralized training techniques and global rewards to foster inter-agent cooperation. The reward function is carefully designed to encourage safe merging, maintain desired speeds, and discourage illegal or unsafe actions for vehicles on both the ramp and the highway. To support experimentation, an open-sourced gym-like simulation environment is created, operating on the SUMO simulator and capable of synchronizing with the CARLA simulator. The environment enables the simulation of various random spawn positions for vehicles and facilitates multi-agent simulations. Through comprehensive experiments and evaluations, we have demonstrated the advantages of our MARL framework by comparing it with independent learning methods.

Keywords: Autonomous Vehicles, Multi-agent Reinforcement Learning, Highway on-Ramp Merging.

الملخص

نحن نركز على تطبيق التعلم المعزز العميق متعدد العملاء ($MARL$) لمعالجة التحدي المعقد لدمج المركبات المستقلة (AVs) على منحدر الدخول الى الطريق السريع. الهدف الرئيسي هو تطوير سياسة تعاونية تمكِّن AVs في كل من المنحدر والطريق السريع من الدمج بكفاءة مع تجنب التصادم وتقليل تأخير المرور. تم تنفيذ إطار $MARL$ باستخدام إطار RL للعميل الفردي كأساس. صُمم الإطار للتعامل مع سيناريوهات حركة المرور الديناميكية ويستخدم التدريب المركزي والمكافآت العامة لتعزيز تعاون الوكيل تم تصميم وظيفة المكافأة بعناية لتشجيع الدمج الآمن والحفاظ على السرعات المطلوبة والحد من الإجراءات غير الآمنة للمركبات في المنحدر والطريق السريع. لدعم التجربة، تم إنشاء بيئة محاكاة مشابهة لبيئة GYM مفتوحة المصدر تعمل على محاكي $SUMO$ وقادرة على المزامنة مع محاكي $CARLA$. من خلال التجارب والتقييمات الشاملة، أظهرنا مزايا إطار $MARL$ الخاص بنا من خلال مقارنته بأساليب التعلم المستقلة.

الكلمات المفتاحية: المركبات المستقلة، التعليم المعزز متعدد العملاء، الدخول الى الطريق السريع.

Dedicated

to my beloved parents.

In the name of Allah, the most Gracious, the most Merciful

Contents

List of Abbreviations **8**

1 Introduction **9**

2 Autonomous Driving **12**
2.1 Introduction . . . 12
2.2 The potential benefits of autonomous driving . . . 12
2.3 Brief history . . . 13
2.4 Levels of autonomy . . . 13
2.5 Main parts of autonomous vehicles . . . 14
2.6 Approaches in Autonomous Vehicles . . . 16
2.7 Connected Vehicles . . . 16
2.8 Connected Autonomous Vehicles (CAVs) . . . 17
2.9 Conclusion . . . 18

3 Machine Learning: An Overview **19**
3.1 Introduction . . . 19
3.1.1 Supervised learning . . . 19
3.1.2 Unsupervised learning . . . 20
3.1.3 Reinforcement learning . . . 20
3.2 Artificial Neural Networks . . . 20
3.3 Deep Learning . . . 21
3.3.1 Deep neural networks . . . 21
3.3.2 Dense networks . . . 22
3.3.3 Convolutional networks . . . 22
3.3.4 Recurrent networks . . . 22
3.4 Data Pre-processing Techniques . . . 22
3.5 Reinforcement learning . . . 23
3.6 Markov Decision Process (MDP) . . . 23
3.7 Policy . . . 24
3.8 Exploration and Exploitation . . . 24
3.9 Reinforcement Learning Algorithms . . . 24
3.9.1 Bellman Optimality and Q-Learning . . . 24
3.9.2 Value-based learning . . . 25
3.9.3 Deep Q-learning . . . 26
3.9.4 Policy-based learning . . . 26
3.9.5 A2C: Synchronous Advantage Actor-Critic . . . 27
3.10 Conclusion . . . 28

4 MARL and it's Application to Autonomous Driving **29**
4.1 Introduction . . . 29
4.2 Markov games . . . 29
4.3 Important Concepts . . . 30
4.4 Baseline Approaches . . . 31
4.5 Extending to MARL . . . 31
4.6 MARL in Autonomous Driving . . . 31
4.7 Conclusion . . . 32

5 Highway on-Ramp Merging for Autonomous Driving: Our Contribution **34**
5.1 Introduction . . . 34
5.2 Problem formulation . . . 34
5.3 MARL Implementation . . . 36
5.3.1 Centralized Multi-Agent DQN . . . 36
5.3.2 Centralized Multi-Agent A2C . . . 37
5.4 Conclusion . . . 38

6 Results and Analysis **39**
6.1 Introduction . . . 39
6.2 Experimental Setup . . . 39
6.2.1 Simulation Tools . . . 39
6.2.2 Simulation setup . . . 41
6.2.3 Evaluation metrics . . . 42
6.2.4 Computing Resources . . . 43
6.2.5 Training process . . . 43
6.2.6 Environment configuration . . . 43
6.2.7 Limitation and assumptions . . . 43
6.3 Training Performance . . . 44
6.4 Metrics Performance . . . 45
6.5 Policy Behavior . . . 46
6.6 Scalability and Generalization . . . 47
6.7 Conclusion . . . 49

7 Conclusion **50**

A ML Concepts **55**
A.1 Time-series data . . . 55
A.2 Outliers data . . . 55
A.3 Data Smoothing . . . 55
A.4 Over-fitting . . . 55
A.5 PyTorch . . . 55

B RL Concepts **56**
B.1 Experience replay memory . . . 56
B.2 Target network . . . 56
B.3 Gradient ascent . . . 56
B.4 Sample efficient RL . . . 57

C RL algorithms **58**
C.1 DQN agent . . . 58
C.2 A2C agent . . . 58

List of Figures

1.1 Illustration of the considered on-ramp merging traffic scenario. J represents the merging junction 9

2.1 The five levels of autonomy published by the Society of Automotive Engineers (SAE)[4] 14
2.2 Two examples of sensor configurations[35] 15

3.1 General layout of an artificial neural network single neuron [18] 20
3.2 deep network architecture with multiple layers[11] 22
3.3 reinforcement learning problem 23
3.4 The actor-critic architecture 27

6.1 Simulation settings for the single highway lane case (a) and multiple highway lanes case (b). . . . 40
6.2 An Example of co-simulation between SUMO and CARLA servers, Showing the highway on-ramp merging 42
6.3 Training performance of MADQN and MAA2C compared with Independent learning IQL and IA2C. 44
6.4 The Exponential epsilon value decay over 5000 training episodes 45
6.5 The speed behavior of the 6 agents for both MADQN and MAA2C 47
6.6 The route behavior of the 6 agents of both MADQN and MAA2C. 47
6.7 Overall performance for MADQN and MAA2C with a higher number of agents running for 10 episodes 48

List of Abbreviations

Abbreviation	Definition
A2C	Advantage Actor-Critic
AI	Artificial Intelligence
AV	Autonomous Vehicles
ANN	Artificial Neural Network
API	Application Programming Interface
CARLA	CAr Learn to Act
CACC	Cooperative Adaptive Cruise Control
C-V2X	Cellular Vehicle-to-Everything
CTDE	Centralized Learning Decentralized Execution
CNN	Convolutional Neural Network
DARPA	Defense Advanced Research Projects Agency
DNN	Deep Neural Network
DSRC	Dedicated Short-Range Communications
DDPG	Deep Deterministic Policy Gradient
DQN	Deep Q-Network
GM	General Motors
GPS	Global Positioning System
GUI	Graphical User Interface
HDV	Human Driven Vehicles
IA2C	Independent Centralized Learner
ICL	Independent Centralized Learner
IL	Imitation Learning
IQL	Independent Q-learning
IMU	Inertial Measurement Unit
LIDAR	LIght Detection And Ranging
MADDPG	Multi-Agent Deep Deterministic Policy Gradient
MADQN	Multi-Agent Deep Q-Network
MARL	Multi-Agent Reinforcement Learning
MACAD	Multi-Agent Connected Autonomous Driving
MDP	Markov Decision Process
POMDP	Partially Observable Markov Decision Processe
PPO	Proximal Policy Optimization
RADAR	RAdio Detection And Ranging
RL	Reinforcement Learning
RNNs	Recurrent Neural Networks
RGB	Red Green Blue
SAE	Society of Automotive Engineer
SLAM	Simultaneous Localization and Mapping
SVMs	Support Vector Machines
SUMO	Simulation of Urban MObility
TTC	Time To Collision
TraCI	Traffic Control Interface
V2I	Vehicle-to-Infrastructure
V2N	Vehicle-to-Network
V2P	Vehicle-to-Pedestrian
V2V	Vehicle-to-Vehicle
WHO	World Health Organization
XML	Extensible Markup Language
3D	three dimensional

Chapter 1

Introduction

Autonomous driving has gained significant attention in recent years, becoming a focal point of academic research and industry applications. Autonomous driving has very convenient benefits over Human Driven Vehicles (HDVs) in terms of safety, efficiency, and traffic optimization, besides the passengers' comfort and fuel consumption. Considerable progress has been made in the development and deployment of semi-autonomous vehicles, particularly at levels 2 and 3 (Section 2.4). However, ensuring safety and improving traffic efficiency are crucial considerations for the further advancement of autonomous vehicle (AV) systems toward fully autonomous vehicles. Consequently, ongoing research efforts are dedicated to enhancing the decision-making capabilities of AVs, with the aim of promoting safer operations and optimizing traffic flow.

One critical scenario that requires careful attention to both safety and traffic efficiency is the on-ramp merging situation. In this scenario, our objective is to minimize trip time delays for vehicles in the highway lane caused by merging vehicles, while enabling the safe merging of vehicles from the ramp without causing collisions.

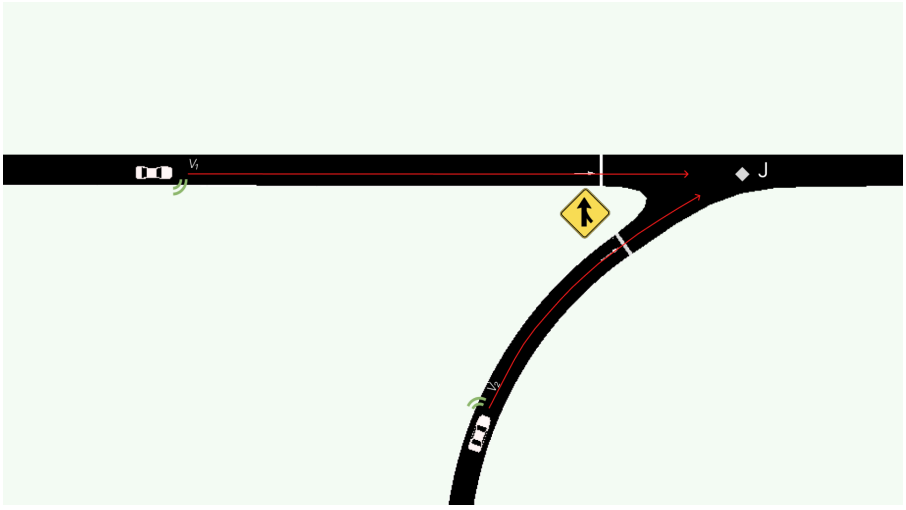


Figure 1.1: Illustration of the considered on-ramp merging traffic scenario. J represents the merging junction

Figure 1.1 illustrates the on-ramp merging scenario, where AVs coexist on both the merge lane and the highway lane. On-ramp vehicles need to merge efficiently onto the highway lane without collisions. In an ideal cooperative, vehicles on the highway lane should proactively adjust their speed to create sufficient space for on-ramp vehicles to merge safely. Simultaneously, on-ramp vehicles should adjust their speed and merge when safe.

Related Work

Various methods have been proposed to address the merging problem, including rule-based and optimization-based approaches[40]. While these methods work well in simple traffic scenarios, they become impractical when dealing with more complex merging situations. In contrast, data-driven approaches such as reinforcement learning (RL) have gained attention and have been explored for autonomous vehicle (AV) highway merging. Some studies have focused on motion planning to enhance efficiency and passenger comfort [1], while others have designed multi-objective reward functions to prioritize safety and minimize jerk using RL techniques like Deep Deterministic Policy Gradient (DDPG) and RL integrated with Model Predictive Control (MPC) [2]. However, these approaches have primarily focused on a single AV, treating other vehicles as part of the environment.

In the context of multi-agent systems, that consider concepts such as cooperation[37] or competition[38]. Dong Chen and colleagues have made significant advancements in the application of multi-agent reinforcement learning (MARL) techniques [3], [5]. Their work applies the MARL framework to the merging problem in a mixed-traffic scenario, where AVs interact with human-driven vehicles (HDVs). They prioritize achieving scalability and employ an action masking scheme to improve learning efficiency by filtering out invalid or unsafe actions [3]. Additionally, they propose a multi-agent advantage actor-critic (MA2C) approach with a novel local reward and parameter sharing scheme [5]. They design a multi-objective reward function that incorporates fuel efficiency, driving comfort, and safety for highway lane changing in mixed traffic. Another related work by Jiawei Wang et al. [36] introduces a graph convolutional reinforcement learning method for connected and automated vehicles (CAVs) in mixed-traffic scenarios to encourage cooperation.

Despite the advancements in MARL, challenges persist in handling the non-stationary nature of the environment, addressing scalability issues, and resolving credit assignment problems. These challenges continue to be areas of active research and improvement[41].

Contribution

In this thesis, we contribute by formulating the on-ramp merging problem as a cooperative multi-agent reinforcement learning (MARL) problem for fully autonomous vehicles. We have developed an environment capable of handling interactions between multiple agents and considering both system-level and individual performance. Additionally, we use the SUMO simulator as a training environment for the on-ramp merging scenario to facilitate our research [1]. By leveraging MARL, we aim to develop a collaborative policy that enables effective and optimized merging, while avoiding collisions and minimizing traffic trip time delays. Our main contributions are:

1. The design of the reward function. We include both the multi-objective rewards and global rewards that consider system-level objectives mainly safety and traffic trip delays.
2. Creating and running the on-ramp merging environment using the TraCI interface to run with the SUMO simulator. In addition, the ability to synchronize with the CARLA simulator.
3. Generalization and scalability abilities. Conducting extensive experiments in different levels of densities

Thesis Structure

- Chapter 2 delves into the realm of autonomous driving vehicles, exploring their historical development and reviewing relevant approaches.
- Chapter 3 serves as an introduction to machine learning and reinforcement learning.
- Chapter 4 outlines our specific approach, which involves applying the Multi-Agent Reinforcement Learning (MARL) framework in an on-ramp environment. This chapter also covers the problem formulation and the reinforcement learning algorithms utilized.

[1] https://github.com/mahmoudtaouti/RL_Highway_Merge

- In Chapter 5, we present the findings of our experiments, analyzing the performance of various autonomous vehicle (AV) policies and examining their impact on safety and traffic flow.
- Finally, Chapter 5 concludes the thesis by summarizing the key discoveries, emphasizing the contributions made, and proposing potential future research.

Chapter 2

Autonomous Driving

2.1 Introduction

Firstly let's see the difference between autonomous and automated systems, basically, autonomous systems are designed to operate in uncertain and dynamic environments, where they must make decisions and take actions based on their perception of the world. Automated systems, on the other hand, operate in well-defined and controlled environments where the tasks and actions they need to perform are pre-defined and predictable[36]. Nevertheless, autonomous driving vehicles (AVs), also known as self-driving cars are vehicles that can operate independently without human input or even monitoring, accurately perceive their environment using a range of sensors, identify objects, and make decisions by using machine learning algorithms, plan the route by using localization techniques and share the information and coordinate the movements using communication systems are the main parts of the autonomous driving cars. The idea of Autonomous driving is one of the most interesting inventions for humanity in the past 20 years, technologies including complex computer systems and precise cameras and sensors have evolved very fast in these years and there is a significant amount of research and development that has led to huge investment from automakers like Mercedes-Benz and Tesla as well as technologies companies like NVIDIA and Google, etc. In this chapter, we will tackle the key concepts behind autonomous driving, like levels of autonomy, sensors, machine learning, and communication. we will also mention in a nutshell different existing approaches.

2.2 The potential benefits of autonomous driving

One obvious benefit of self-driving cars is the ability to use the time spent driving for other activities instead of focusing on the road. Autonomous driving allows the car to handle tasks that would normally require the driver's attention, such as staying in the correct lane, avoiding obstacles, and controlling acceleration.

However, fully autonomous driving cars could also provide significant improvements to traffic safety. According to the World Health Organization (WHO), over 1.2 million people die on the world's roads every year, with up to 50 million individuals injured[4]. Human errors, such as speeding, driving under the influence of drugs or alcohol, or being distracted, are often the leading causes of these accidents.

In addition, autonomous driving technology has the potential to enhance mobility for individuals with physical disabilities. For example, it could allow people with mobility impairments to travel independently to work, school, or social events without relying on others for transportation.

Furthermore, the technology could also lead to improvements in environmental sustainability in several ways, such as reducing fuel consumption and emissions by making better decisions about acceleration and

braking, reducing traffic congestion, and optimizing route planning, which can further contribute to reducing emissions.

Autonomous driving technology has the potential to revolutionize car sharing and pooling by making it easier. Car sharing is a service that enables individuals to rent a car for a short period, on the other hand, carpooling is when multiple people travel together in the same vehicle. This can lead to fewer cars on the road, reduced traffic congestion, and lower emissions.

2.3 Brief history

The idea of autonomous cars dates back to the 1920s when there was an attempt to make a driverless car that could be remotely controlled by another car. In 1962, General Motors (GM) introduced the concept of magnetic cables built into the road to control a car's movements through car-to-infrastructure communication with their Firebird III. However, the early milestone in the development of autonomous cars was in 1984 when Carnegie Mellon University's Robotics Institute launched a project called Navlab. The aim was to develop an autonomous vehicle capable of driving on public roads. The Navlab team started by modifying a normal car and adding sensors such as cameras and radar, as well as a system that could process sensor data and make control decisions. Over the years, the Navlab team developed advanced software for road recognition, obstacle detection, and path planning, resulting in significant improvements. In 1995, Navlab 5 became the first autonomous vehicle to drive 2,849 miles with an autonomy of 98% across the United States.

In the early 2000s, the development of autonomous progressed significantly, with the improvements in sensor quality and, the development of a satellite-based Global Positioning System (GPS) which enabled to determination of the vehicle's position within a range of meters. Computing hardware continued to increase in power and shrink in size. In 2004 the Defense Advanced Research Projects Agency (DARPA) launched the first competition Grand Challenge series of self-driving cars, the goal of the competition was to drive 150 mile route autonomously through the Mojave Desert, None of the teams were able to complete the course, but in the subsequent challenges, several teams were able to successfully complete the courses. The DARPA Grand Challenges played a significant role in advancing autonomous driving technology and spurred the development of many companies in the field, including Waymo and Tesla, among others[33].

Google began working on autonomous driving in 2009 and developed a prototype that was later named Waymo. The company also invested heavily in research and development, with reports indicating that it had invested more than $1 billion in self-driving technology by 2015. Another milestone in the development of autonomous is the development of representation learning and deep learning led to more accurate and reliable systems for perception, decision-making, and control. For example, deep neural networks can be trained to accurately identify objects in the environment, such as other vehicles, pedestrians, and traffic signs, using data from cameras and other sensors. These advancements in addition to creating new benchmarks provide a common ground for researchers and developers to identify weaknesses and improvements. Ultimately leading to better and more advanced autonomous driving technology.

2.4 Levels of autonomy

In 2014, the Society of Automotive Engineers (SAE) International published a report titled "Taxonomy and Definitions for Terms Related to Driving Automation Systems for On-Road Motor Vehicles," which established a standardized framework for categorizing the levels of driving automation. The report defined five levels of autonomy ranging from Level 0 (no automation) to Level 5 (full automation). This framework has

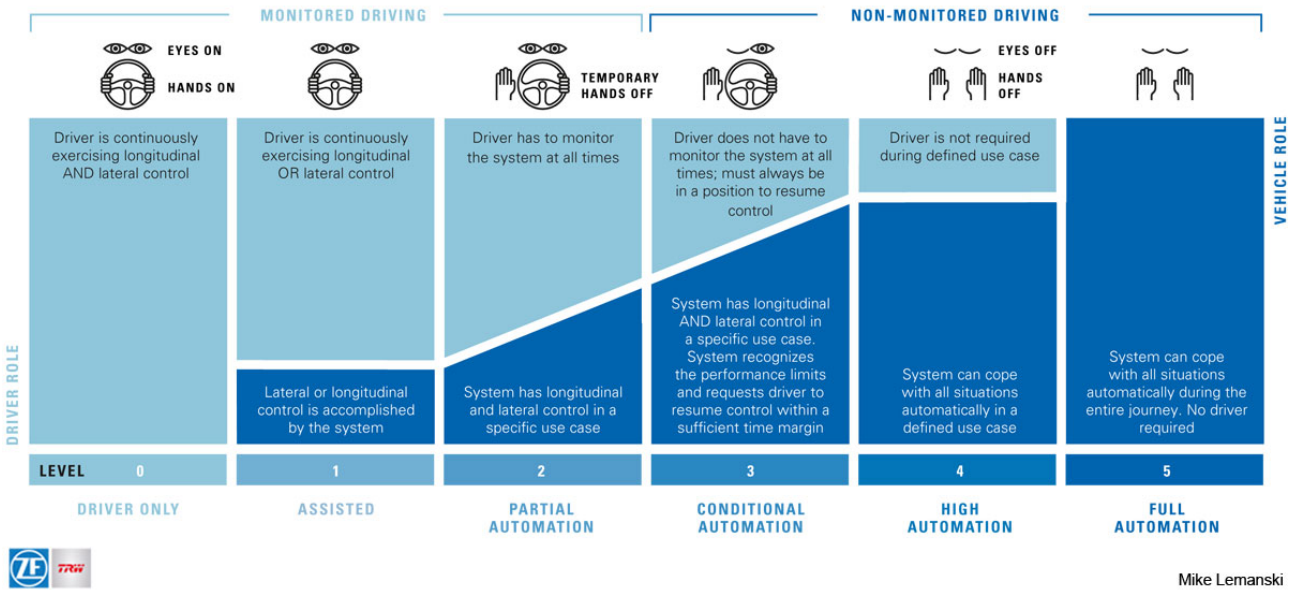


Figure 2.1: The five levels of autonomy published by the Society of Automotive Engineers (SAE)[4]

been widely adopted in the industry to describe the different stages of autonomous vehicle development. Figure 2.1 shows how the driving tasks run short as the systems take in place the tasks without the need for human intervention, level 1 to level 3 are likely to assist the driver in specific cases, for example, Tesla cars now have autopilot considered as level 2, it can steer, accelerate and brake automatically within its lane. Waymo and Cruise, now operate ride-hailing services with vehicles capable of Level 4 autonomy in the US, which means the cars can operate without a driver behind the wheel under certain conditions, such as within a designated service area. We can say that the current industry is working towards achieving full automation in a step-by-step approach, commonly referred to as a bottom-up approach. This involves developing and improving the different components and technologies required for autonomous driving, such as sensors, machine learning algorithms, and communication systems and gradually integrating them into more advanced autonomous systems. However, it is important to note that there are still many challenges to overcome before achieving full automation, such as safety and regulatory concerns, technical limitations, and ethical considerations.

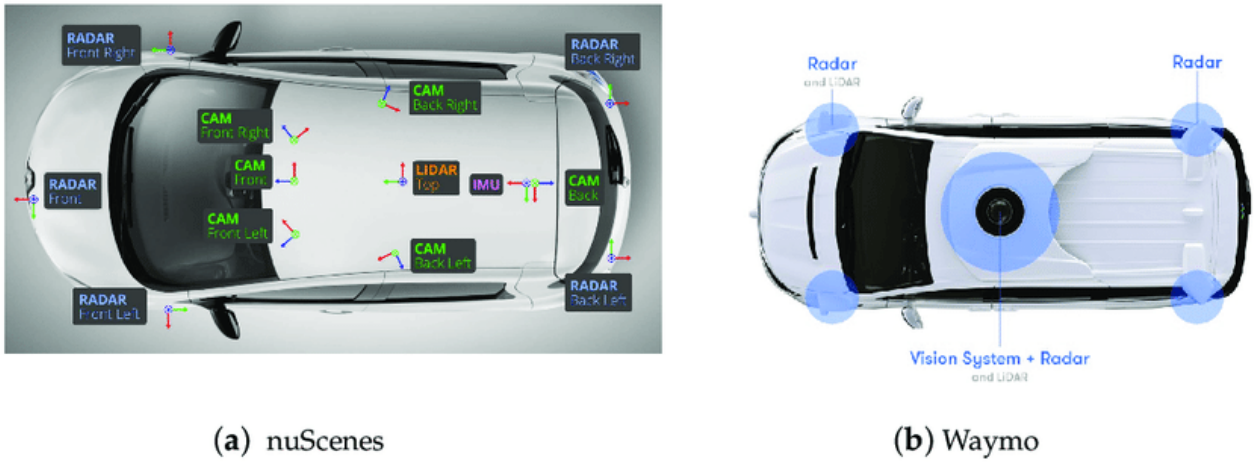
2.5 Main parts of autonomous vehicles

In order to make the vehicle operate safely and efficiently in a complex and dynamic environment there are three important capabilities: perception, localization, and prediction.

1. Perception: Perception involves the ability of the vehicle to detect and interpret objects and events in the environment, including other vehicles, pedestrians, road signs, and traffic lights. Figure 2.2 shows Two examples of sensor configurations, from nuScenes, Waymo Team.

   The key sensor technologies used in autonomous driving include:

   - Digital camera: Cameras capture the environment optically, but may not detect everything because of environmental conditions such as darkness, fog, or backlight.
   - Laser scanner/ LIDAR: The distance and speed of objects can be measured by using many laser beams. This allows for the creation of an accurate 3D picture of the environment. The downside of this sensor is that it's very expensive.
   - Radar and ultrasound sensors: Additional directional and distance measurement technologies complement the range of perception of other sensors and provide redundancy so that nothing is overlooked.



(a) nuScenes (b) Waymo

Figure 2.2: Two examples of sensor configurations[35]

In addition to these sensors, HD maps function as a fourth sensor. The data from the first three technologies is compared with the HD maps, providing the car with all pre-recorded details about the environment, including information outside the vehicle's sensor range. Sensor data from other vehicles (Vehicle-to-Vehicle Communication, V2V) can also be used, for example, to supplement a regional digital map (e.g. traffic congestion information from other vehicles).

2. Localization: Enable the vehicle to determine its position relative to the environment, which is important for Precise Navigation to its desired destination in a safe and effective manner.

   - GPS: provides location and time information anywhere on or near the Earth's surface. GPS guarantees the accuracy of positioning with deviations below eight meters, but it can be affected by factors such as tall buildings, tunnels, and natural obstacles like mountains or trees, which can block the signals from the GPS satellites.
   - Inertial measurement units (IMUs): measures the vehicle's linear and angular accelerations and rotational rates in three-dimensional space. It can provide information on the vehicle's orientation, velocity, and position, based on the integration of the acceleration and rotation data over time. IMU is commonly used in combination with GPS to provide more accurate and reliable localization information, especially in areas where GPS signals are weak or obstructed.

   However, more precise technologies such as visual SLAM (Simultaneous Localization and Mapping) or visual odometry can be used in conjunction with GPS. Basically, SLAM is a technique used to build a map of the surrounding environment while simultaneously localizing the vehicle within that map, by combining data from RADAR and LiDAR sensors. On the other hand visual odometry, uses a camera feed to determine how the vehicle is moving through the space. Visual odometry allows for enhanced navigational accuracy in vehicles.

3. Prediction: Once the vehicle has perceived its environment and accurately localized itself, it must then make predictions about what might happen in the near future, based on its understanding of the environment. For example, the vehicle may predict that a pedestrian will cross the street or that a vehicle ahead will make a turn. These predictions are used to inform the vehicle's decision-making process, allowing it to react appropriately to potential hazards and avoid accidents.

   Prediction involves using machine learning algorithms and other techniques to anticipate the actions and behavior of other objects in the surrounding environment. The vehicle system processes the data collected from sensors using machine learning algorithms to extract patterns and make accurate predictions.

Techniques including Bayesian filters, Kalman filters, and deep learning, such as convolutional neural networks (CNNs), are also used, allowing the system to make more accurate predictions about the behavior of other vehicles and objects.

In addition, these techniques both Reinforcement learning (RL) and Imitation learning (IL) can improve prediction capabilities by allowing one to learn from data and make decisions based on predicted output. These approaches can also be combined with other prediction methods such as probabilistic models or rule-based systems to further improve the accuracy and robustness of autonomous driving systems.
The predictions made by the autonomous vehicle system are then used to inform the decision-making process. For example, if the system predicts that a pedestrian is about to cross the road, the vehicle may slow down or stop to avoid a collision.

2.6 Approaches in Autonomous Vehicles

Modular Pipeline Approach

The modular pipeline approach is a traditional method in autonomous vehicles. It involves breaking down the autonomous driving task into separate modules, each responsible for a specific function such as perception, planning, and control.

In this approach, perception modules analyze sensor data to extract relevant information about the surrounding environment, including lane detection, object detection, and traffic sign recognition. The planning module processes this information and generates a driving plan, determining the vehicle's trajectory and maneuvers such as lane changes and turns. Finally, the control module translates the driving plan into low-level commands to control the vehicle's actuators (e.g., steering, acceleration, braking)[28].

End-to-End Learning Approach

End-to-End Learning is an alternative approach to the Modular Pipeline in autonomous driving. It focuses on solving the mapping from sensory input data to driving commands using a neural network in a single step. This approach involves training the neural network model using two prominent algorithms: Reinforcement Learning and Imitation Learning. The advantages of End-to-End Learning include the direct optimization of the model for driving and the cost-effectiveness of data annotations. Data collection can be automated by attaching a camera and sensors to the vehicles in addition to localization and steering in formations. The neural network is trained using algorithms such as imitation learning and reinforcement learning[29].

Direct Perception

Direct Perception represents a hybrid approach that combines elements of Modular Pipelines and End-to-End Learning in autonomous driving. It involves the use of a neural network to create an intermediate representation, such as semantic segmentation, from raw sensor data. This intermediate representation is then processed by a second neural network or classical vehicle controller to generate the vehicle's control signals[30]. The advantages of the Direct Perception approach include the potential for a compact and interpretable representation.

2.7 Connected Vehicles

Connected vehicles are equipped with technologies that allow them to communicate with other vehicles and infrastructure, such as traffic lights and road signs. This communication can provide a more comprehensive perception of the surrounding environment beyond the range of sensors, allowing the vehicle to perceive things that are further away. Connected vehicles use a combination of communication and coordination techniques to

enable vehicles and infrastructure to work together and achieve their individual and collective goals. Techniques such as Dedicated Short Range Communications (DSRC)[26] and Cellular Vehicle-to-Everything (C-V2X)[25] enable communication between vehicles and infrastructure using wireless communication technologies. In addition to communication, agents require coordination, which is achieved through the use of algorithms. Intersection Collision Avoidance (ICA)[27] and Cooperative Adaptive Cruise Control (CACC)[24] are examples of algorithms used for coordination. ICA uses V2X communication to detect potential collisions at intersections, while CACC allows vehicles to communicate with each other to maintain a safe distance and speed on the road.

Connected vehicles can be considered as a multi-agent system which we will tackle in the incoming chapters, where each vehicle is an agent that interacts with other agents (i.e., other vehicles, infrastructure, and pedestrians) to achieve common or individual goals, such as improving traffic flow, increasing safety, or reducing emissions.

Connectivity can play an important role in road safety and traffic management by bringing benefits related to congestion. However, connectivity also poses challenges in relation to technology compatibility, data security, and privacy. To ensure that connected vehicles are safe and secure, it is essential to address these challenges.

Types of communication

- Vehicle-to-Vehicle (V2V) Communication: This type of communication allows vehicles to communicate with each other directly, exchanging information such as speed, position, and direction of travel. This can help improve safety by allowing vehicles to avoid collisions and respond to potential hazards more quickly.
- Vehicle-to-Infrastructure (V2I) Communication: This type of communication allows vehicles to communicate with roadside infrastructure, such as traffic lights, signs, and cameras. This can provide drivers with real-time traffic and road condition information, as well as enable traffic management systems to optimize traffic flow.
- Vehicle-to-Pedestrian (V2P) Communication: This type of communication allows vehicles to detect and communicate with pedestrians, providing them with warnings or alerts when a vehicle is approaching. This can help improve pedestrian safety and reduce accidents.
- Vehicle-to-Network (V2N) Communication: This type of communication allows vehicles to communicate with a central network, providing real-time traffic and road condition information, as well as enabling remote vehicle monitoring and management. This can help improve traffic flow and reduce congestion, as well as enable remote vehicle diagnostics and maintenance.

2.8 Connected Autonomous Vehicles (CAVs)

Combine connectivity with automation, and they have the potential to revolutionize the way we travel[31]. However, one of the most challenging problems is latency, which is the delay in transmitting and receiving data between vehicles and infrastructure. Latency can be a significant issue as it affects the timeliness of decision-making and action-taking, which can impact safety. The large amounts of data can be handled better with a high mobile standard. The currently much-discussed 5G mobile standard seems particularly suitable here given its low latency and high data transmission rates. Therefore AVs can be connected, which can help us, improve the perception so it can be more accurate, and they can have cooperative tasks to improve traffic flow.

2.9 Conclusion

Autonomous driving has the potential to transform transportation by enabling vehicles to operate independently without human intervention. Key capabilities for autonomous vehicles include perception, localization, and prediction, which heavily rely on sensor technologies. Various approaches, such as the modular pipeline approach, end-to-end learning, and direct perception, are being utilized in the development of autonomous vehicles. Communication technologies enable connected vehicles to interact with each other and infrastructure, improving perception and coordination. However, there are still challenges to address before achieving Level 5 automation. In the next chapter, we will provide an overview of machine learning, a vital component of autonomous driving systems, particularly in our application where we aim to use Reinforcement Learning following the End-to-End approach.

Chapter 3

Machine Learning: An Overview

3.1 Introduction

AI refers to any software or process designed to simulate human intelligence. As a subset of AI, Machine learning (ML) has become more and more important in many modern technologies, including image recognition, autonomous vehicles, and virtual assistants like Apple's Siri[36]. ML focuses on creating algorithms and models that can learn patterns from a large volume of data, without being explicitly programmed, allowing machines to improve over time, becoming increasingly accurate when making predictions or classifications. ML works in three basic parts, starting with using a combination of data and algorithms to predict patterns and classify data sets, an error function that helps evaluate the accuracy, and then an optimization process to fit the data points into the model best[8].

1. A Decision Process: From labeled/unlabeled data, ML algorithms make the prediction or classification to produce an estimation about the pattern in data.
2. An Error Function: An error function is used to evaluate the estimation accuracy of the model.
3. A Model Optimization Process: Based on the error function evaluation, update the weights to reduce the variation between the target model and the model estimate. Then we repeat the evaluation and optimization process until a threshold of accuracy has been met.

Nevertheless, Deep learning models are a nascent subset of machine learning paradigms. Deep learning uses a group of connected layers which together are capable of quickly and efficiently learning complex prediction models. In fact, Deep Learning is a subset of artificial neural networks (ANNs), hence Deep learning differs from other neural networks because it uses more than one hidden layer between the input and the output, and also it requires massive amounts of raw data. In general, there are three main categories of machine learning: supervised learning, unsupervised learning, and reinforcement learning.

3.1.1 Supervised learning

Supervised learning is a type of machine learning in which the algorithm is trained on labeled data. The goal is to learn a function that can map inputs to outputs based on the labeled examples provided by an external supervisor. Classification, regression, and imitation are common supervised learning tasks. In classification similar data sets are classified in one set based on their characteristics, using techniques like Decision trees, Native Bayes, and support vector machines. Whereas regression is a technique for investigating the relationship between independent variables or features and a dependent variable or outcome. And last the imitation learning approach that is commonly used in autonomous vehicles and robotics. It is based on learning from expert demonstrations, where the agent observes the expert's behavior and learns to imitate it by mapping inputs to actions. This approach enables the agent to learn from the expert's knowledge and experience and can be used for tasks such as lane following, obstacle avoidance, and traffic signal recognition.

3.1.2 Unsupervised learning

While supervised learning uses labeled data provided by users, unsupervised learning doesn't use the same labeled training sets and data, hence the goal is to identify patterns in the data without any pre-existing knowledge, which is typically about finding structure hidden in collections of unlabeled data. Common algorithms used in unsupervised learning include Hidden Markov models, k-means, hierarchical clustering, and Gaussian mixture models. These methods allow the discovery of similarities and differences in information making it ideal for exploratory data analysis, cross-selling strategies, customer segmentation, and image and pattern recognition.

3.1.3 Reinforcement learning

Reinforcement learning (RL) is a type of machine learning in which an agent learns to interact with an environment by taking actions that maximize a cumulative reward signal. The goal of RL is to train an agent to make decisions that lead to the achievement of a particular goal by selecting the best possible actions. The agent learns by receiving feedback from the environment in the form of positive or negative rewards [9]. This type of training helps us a lot when it comes to talking about autonomous driving, as we know autonomous vehicles live in uncertain environments, where they must make decisions and actions based on their perception of the world, such as deciding when to brake or accelerate based on the current speed and distance to other vehicles. RL allows the agent to learn from its own experiences through trial and error, thus this can help us a lot when we talk about the scalability of the world and thousands of scenarios that cannot be learned from expert data.

3.2 Artificial Neural Networks

Artificial Neural networks are a powerful class of machine learning algorithms. They are composed of multiple layers of interconnected nodes, which are capable of learning complex relationships between the input and output variables. The most common neural network architecture used for regression problems is the feed-forward neural network. Artificial neural networks (ANNs) are comprised of node layers, containing an input layer, one or more hidden layers, and an output layer. Each node, or artificial neuron, connects to another and has an associated weight and threshold. If the output of any individual node is above the specified threshold value, that node is activated, sending data to the next layer of the network. Otherwise, no data is passed along to the next layer of the network by that node.

A single neuron might represented as in Figure 3.1. The idea is that "a single neuron is just the sum of all of

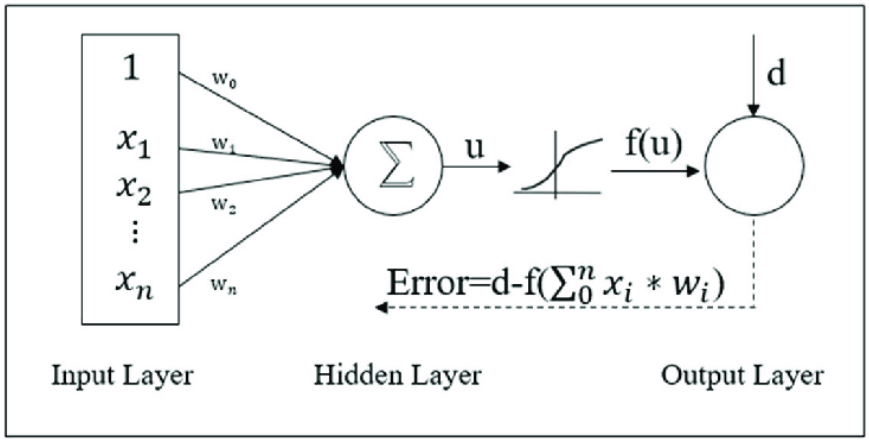


Figure 3.1: General layout of an artificial neural network single neuron [18]

the inputs x weights, fed through some sort of activation function". The activation function is meant to simulate

a neuron firing or not. A simple example would be a stepper function, where, at some point, the threshold is crossed, and the neuron fires a 1, else a 0. Let's say that neuron is in the first hidden layer, and it's going to communicate with the next hidden layer. So it is going to send its 0 or a 1 signal, multiplied by the weights, to the next neuron, and this is the process for all neurons and all layers.

- Let X be the input data layer of the neuron with n input features. Each input feature is multiplied by a weight w_i, resulting in n weighted inputs, i.e.,

$$weighted\_input = \sum(X_i * w_i), i = (1..n) \tag{3.1}$$

- The weighted inputs are then aggregated to produce the neuron's total input u, given by:

$$u = \sum(X_i * w_i) + b, b \text{ is the bias term.} \tag{3.2}$$

- The total input is then passed through an activation function f(u), which introduces non-linearity into the neuron's output. For example, sigmoid activation functions are given by:

$$f(u) = 1/(1 + \exp(-u)) \tag{3.3}$$

- The output of the function f(u) will always be a value between 0 and 1, which can be interpreted as a probability or confidence score for a given input value. There are more Common activation functions including the Rectified Linear Unit (ReLU) function, which is half rectified (from the bottom), all the negative values become zero immediately, so it takes $max(0, u)$.

- Then, the output y of the neuron is compared to the expected output using an error function or loss function. The error function quantifies the difference between the neuron's output and the expected output and is used to adjust the weights and bias of the neuron during the training process. For example, Mean Squared Error (MSE) is defined as :

$$MSE = \frac{\sum_{i=0}^{N}(y_i - y_i')^2}{N} \tag{3.4}$$

  where N is the number of samples, y_i is the actual value, and y_i' is the predicted value. The main goal of an optimizer is to minimize the loss or error function of the network by iteratively updating the parameters based on the gradients of the loss function with respect to the weights. The optimizer achieves this by using various optimization algorithms such as RMSprop and Adam to determine how the weights should be updated.

- This process of calculating the weighted input, aggregating the inputs, applying an activation function, and computing the output is repeated for each neuron in the neural network, leading to the final output of the network.

3.3 Deep Learning

3.3.1 Deep neural networks

Basically, Deep neural networks have multiple layers, a typical deep neural network (DNN) has at least three hidden layers. Moreover, if we have a single hidden layer, the model is going to only learn linear relationships. Further, if we have many hidden layers, then the model learns non-linear relationships between the

input and output layers (G. Cybenko, 1989). So, Deep neural networks can have many layers and millions of parameters. Thus they are well-suited for processing various types of data. For example, deep neural networks are commonly used for image and speech recognition, natural language processing, and time series forecasting (appendix A.1), among other applications.

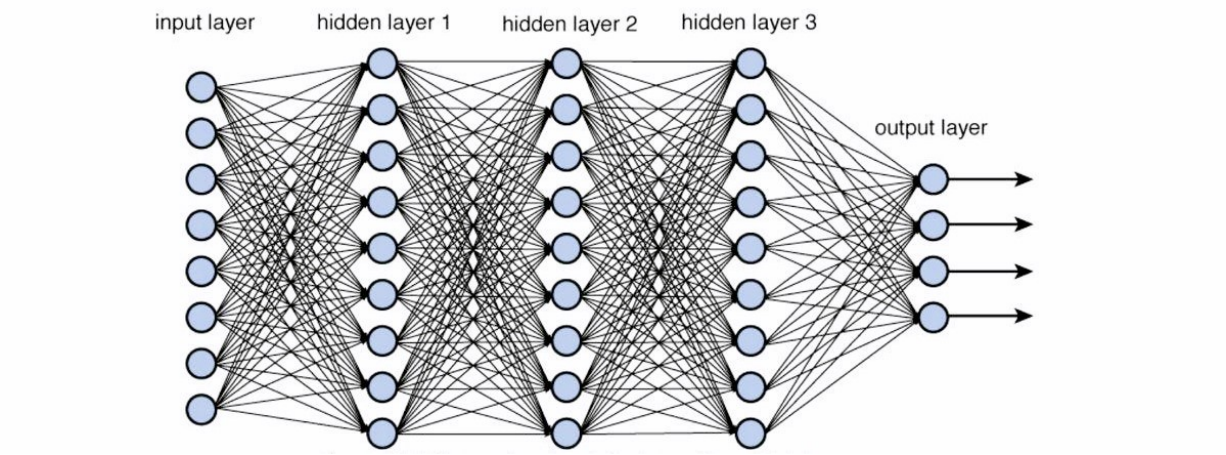


Figure 3.2: deep network architecture with multiple layers[11]

3.3.2 Dense networks

Dense neural networks (also called fully connected neural networks) consist of multiple layers of interconnected nodes. Each node receives input from all nodes in the previous layer and produces an output that is passed to all nodes in the next layer. Dense neural networks are commonly used for classification and regression problems, such as image classification, language translation, and speech recognition. In autonomous driving, dense neural networks can be used for object detection, road segmentation, and obstacle avoidance.

3.3.3 Convolutional networks

Convolutional neural networks (CNNs) are designed to process data with a grid-like topology, such as images and video. They use convolutional layers to extract features from the input and pooling layers to reduce the spatial dimensions of the feature maps. CNNs have achieved state-of-the-art performance in image and video classification, object detection, and segmentation. In autonomous driving, CNNs are used for tasks such as lane detection, object detection, and pedestrian detection.

3.3.4 Recurrent networks

Recurrent neural networks (RNNs) are designed to process sequential data, such as time-series data and natural language. They use recurrent connections between nodes to maintain an internal state, which allows them to remember previous inputs and produce outputs that depend on the context of the sequence. RNNs are commonly used for language modeling, speech recognition, and machine translation. In autonomous driving, RNNs can be used for tasks such as trajectory prediction and behavior prediction.

3.4 Data Pre-processing Techniques

Data pre-processing involves transforming raw input data into a suitable format that enhances the performance and effectiveness of the neural network. The principle behind data pre-processing is that a neural network is only as good as the quality and relevance of the input data used to train it. Some key techniques used in data pre-processing are:

1. Data Cleaning: This involves removing or correcting any inconsistencies, or missing values in the dataset. It prevents potential biases or misleading patterns during training.

2. Data Normalization: Normalization is the process of scaling the features of the dataset to a specific range, typically between 0 and 1. It prevents features with larger scales from dominating the learning process and helps the neural network converge faster.

3. Feature Scaling: Similar to normalization, feature scaling brings features to a similar scale to avoid bias in the learning process. Common scaling techniques include standardization and min-max scaling.

4. Handling Categorical Data: Neural networks typically work with numerical data, so categorical variables need to be encoded appropriately. This can be done using techniques like one-hot encoding, where each category is represented as a binary vector.

5. Handling Outliers: Outliers (appendix A.2) can have a significant impact on the training process and model performance. They can be treated by either removing them or replacing them with more suitable values.

3.5 Reinforcement learning

As we mentioned in the last section, Reinforcement learning (RL) is a type of machine learning in which an agent learns to interact with an environment by taking actions that maximize a cumulative reward signal. RL problem can be illustrated in Figure 3.3 and can be formally defined as a Markov decision process (MDP).

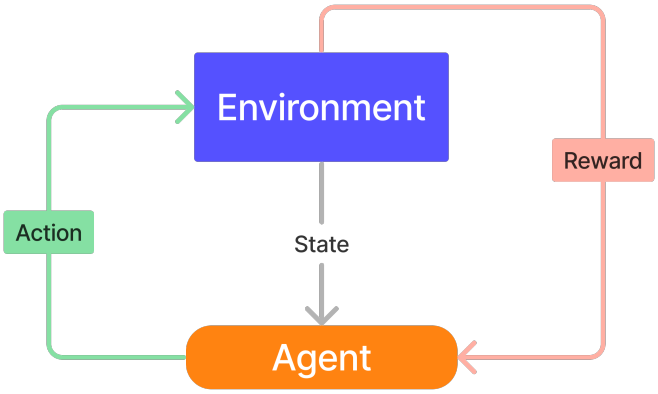


Figure 3.3: reinforcement learning problem

3.6 Markov Decision Process (MDP)

As RL problem is decision-making problems in a stochastic environment It can be defined as MDP. An MDP consists of a tuple(S, A, R, P, γ). Where S is a set of states, A the set of possible actions, $R(r_t|s_t, a_t)$ the distribution of the current reward given a (state, action) pair, and $P(s_{t+1}|s_t, a_t)$ the distribution over the next state given a (state, action) pair. γ is a discount factor that decreases the degree to which future rewards are taken into account for the present action choice.

The MDP has the Markov property, which means that the future is independent of the past given the present. This can be expressed as $P(s_{t+1}|s_t) = P(s_{t+1}|s_1, ..., s_t)$.

Reinforcement learning involves the agent interacting with the environment by selecting action a based on a policy π which defines the behavior of the learning agent at a given time t and maps states to actions. At the start of each episode, the agent starts in an initial state s_0 that is sampled from the distribution $P(s_0)$. Then, the agent selects an action according to the policy π and receives a reward r_t and the next state s_{t+1}, both of which are sampled from the distributions R and P, respectively. Note that reward defines what are the good and bad events for the agent. The agent uses the received reward and the next state to update its policy and improve its decision-making over time. The process is repeated in the form of episodes until the end of the rollout of the policy[45].

3.7 Policy

A policy π is a function from S to A that provides the next action based on a given state, and the goal is to find the best policy. In a discrete state and action space this corresponds to a look-up table or a neural network with some parameters. The general formula for a policy depends on whether it is deterministic or stochastic. deterministic policies $\pi(s) = a$, with a direct mapping. and stochastic policies $\pi(a|s) = P(a_t = a|s_t = s)$, with a distribution of probabilities over the action for a given state[9]. Even though MDP policies depend on the current state, past observations may be included, e.g. in the form of a buffer of the ten last images. In imitation learning, the policy was learned from expert data, which made it a supervised learning problem. In RL policies are learning through trial and error only, the quantity that is maximized in this case is the expected future reward. On the basis of interactions with the environment, the agent discovers good actions and can then improve the policy.

3.8 Exploration and Exploitation

Exploration and exploitation are two fundamental tasks in reinforcement learning. The aim of exploration is to gather the reward signals of as many state-action pairs as possible. This involves trying out different actions in different states to discover desirable actions that are not provided through any preexisting knowledge base. However, during the exploration phase, total reward may have to be sacrificed because many of the untried actions may not yield a worthwhile reward. The second task of reinforcement learning is exploitation, which involves making the most use of what has been learned during exploration by choosing the best action in a given state out of all actions that have been tried. However, exploitation disregards the unexplored areas that may contain better rewards. These two tasks complement each other, but they also form a trade-off, and exploration and exploitation need to be balanced. This balance can be achieved via $\epsilon - greedy$ exploration, where all possible actions with non-zero probabilities are tried out, and the probability ϵ by which an action is chosen at random is decreased over time to promote exploration at first and then moves towards exploitation as the known action-reward pairs increase in number[9].

Exponential epsilon decay is another technique used to gradually reduce the exploration rate ϵ over time. Instead of using a fixed ϵ throughout the learning process, the exploration rate is decayed exponentially over episodes or time steps. This approach starts with a higher exploration rate, promoting early exploration, and then decreases it exponentially as the agent gains more experience and knowledge about the environment[23].

3.9 Reinforcement Learning Algorithms

3.9.1 Bellman Optimality and Q-Learning

First, we have to understand what value functions are, how to deal with future rewards, and what the Bellman optimality principle. Value functions assign values to states in order to be able to assess how good or bad a present state is based on rewards received in the future.

The state-value function $V\pi$ at state st is the expected cumulative discounted reward ($r_t \sim R(r_t|s_t, a_t)$) when following policy π from state st:

$$V\pi(s_t) = E[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots | s_t, \pi] = E[\sum_{k=0}^{n} \gamma^k r_{t+k}] \tag{3.5}$$

In other words, given a trained policy, the state-value function tells us the expected future reward with discount factor γ for an initial state st. The discount factor γ is multiplied with each new future reward and taken to the exponent of the number of time steps into the future (i.e., γ^2 for the reward two-time steps into the future). The discounting ensures that we don't take rewards into account now that are too far off into the future equally as much as present rewards. The hyper-parameter γ lies between 0 and 1 and controls how much weight we want to give to the future.

The action-value function Q_π at state st and action a_t is the expected cumulative discounted reward when taking action at in state st and then following the policy $\pi : Q\pi(s_t, a_t) = E[\sum_{k>=0} \gamma^k r_{t+k} | s_t, a_t, \pi]$ The action-value function differs from the state-value function only by additionally conditioning on the action. The discount factor γ works the same way as for the state-value function[6].

The optimal state-value function $V^*(s_t)$ is the best $V\pi(s_t)$ over all policies π:

$$V^*(s_t) = \max_\pi V^\pi(s_t) \tag{3.6}$$

Intuitively, it tells us the maximal value we can obtain. Similarly, the optimal action-value function $Q^*(st, at)$ is the best $Q\pi(st, at) over all policies \pi$:

$$Q^*(s_t, a_t) = max_\pi Q^\pi(s_t, a_t) \tag{3.7}$$

It tells us what we can achieve in the best case if we start at s_t and have conducted a_t. These optimal functions tell us what the best possible performance in the MDP would be. In most cases, finding them is computationally intractable since we would have to search the space of all possible policies. Assuming we had Q^*, we would immediately have access to the optimal policy, since

$$\pi^*(st) = argmax_{a' \in A} Q^*(s_t, a') \tag{3.8}$$

To determine Q^* is hard. Since we cannot search over all possible policies. So, it is not as easy as doing a complete search for the optimal behavior in the environment. We can use The Bellman Optimality Equation (BOE) by Richard Ernest Bellmann from the 1953 approach to determine Q^*

$$Q^*(s_t, a_t) = E[rt + \gamma max_{a'} Q^*(s_{t+1}, a') | s_t, a_t] \tag{3.9}$$

According to [6], BOE reformulates the first expectation over all future time steps recursively, by first returning the reward of the current time step and taking γ times the maximum over all actions of the Q^* function of the next step. Due to the non-linearity of the max-operator, no closed-form solution is available for the BOE. However, iterative approaches like Q-Learning can be employed to approximate the Q^* function. If we can solve the BOE, the optimal expected long-term reward is locally and immediately available for each state-action pair via Q^* and V^*. For V^*, we need a one-step-ahead search to get the optimal action. Q^*, however, cashes these one-step-ahead searches and this means we do not have to do any rollouts anymore - we can simply select an optimal action without explicitly considering future because it is encoded in Q^*.

3.9.2 Value-based learning

The most fundamental algorithm in reinforcement learning is Q-learning. Q-learning is a popular value-based algorithm used to approximate the optimal Q-values and learn the optimal policy in a model-free

manner. It is based on the idea of updating the Q-values using a temporal difference (TD) error, which is the difference between the observed reward and the predicted reward based on the current estimate of the Q-values. Q-Learning can converge to the optimal Q-values under certain conditions and has been widely used in various applications of reinforcement learning.

$$Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max_{a'} Q(s',a') - Q(s,a)] \tag{3.10}$$

Where Q(s, a) is the estimated value (or Q-value) of taking action in state s, and α is the learning rate. Basically α determines how much the Q-value for a state-action pair is updated based on the difference between the observed reward and the expected reward. A high learning rate means that new information is given more weight, leading to faster learning but also greater instability, while a low learning rate means that old information is given more weight, leading to slower learning but more stability.

3.9.3 Deep Q-learning

Q-learning is a simple form of reinforcement learning that estimates the optimal action-value function using a table (Q table). It does not require any prior knowledge of the environment or a model of the system dynamics, but it becomes impractical for larger environments. Deep Q-learning, on the other hand, uses a neural network to approximate the optimal action-value function[46]. Instead of using a table, the agent learns a function that maps states to actions. This allows deep Q-learning to scale to environments with large state and action spaces. Deep Q-learning uses experience replay and a target network (appendix B.2) to stabilize the learning process and prevent over-fitting (appendix A.4).

During training, the DQN algorithm updates the Q-network weights θ by minimizing the mean squared error between the Q-target and the predicted Q-value for the current state-action pair. The loss function can be written as:$L(\theta) = E[(r + \gamma \max(Q(s',a';\theta-)) - Q(s,a;\theta))^2]$.

The Q network can be optimized using experience replay (appendix B.1), which involves storing the agent's experiences at each time step in a replay memory. This memory is continually updated with new experiences, made up of tuples (state, action, reward, next state). During training, samples are randomly drawn from this memory, breaking the correlation between consecutive samples and improving data efficiency. This is achieved by using a circular replay memory where old experiences are gradually deleted as new ones are added, keeping the memory size fixed.

3.9.4 Policy-based learning

Policy-based methods are a class of reinforcement learning algorithms that directly parameterize the policy function, which maps states to actions. Instead of estimating the value function (such as value-based methods like Q-learning), the key characteristic of on-policy methods is that they learn directly from the data generated by the current policy. Moreover, on-policy methods typically use techniques such as policy gradient to adjust the policy parameters in the direction of higher rewards. Policy gradient is an optimization technique to update the policy parameters and improve the policy's performance. The policy gradient method is based on the concept of gradient ascent, where the objective is to maximize an expected return or a performance measure, let's consider a parameterized policy function $\pi\theta$ that maps states to actions, where θ represents the parameters of the policy. The goal is to find the optimal set of parameters $\theta*$ that maximizes the expected return $J(\theta)$ of the policy based on the total[32]. Thus, the cost function of the parameters is the following:

$$J(\theta) = E\pi_\theta[r(\tau)] \tag{3.11}$$

The parameters are then tuned based on the gradient of the cost function:

$$\theta_{t+1} = \theta_t + \alpha\Delta_J(\,\theta t) \tag{3.12}$$

An advantage of using policy-based methods is the possibility of mapping environments with huge, even continuous action spaces. Environments with stochasticity can also be solved with them. Examples of popular deep RL policy-based methods include DDPG(Deep Deterministic Policy Gradient)[14], PPO (Proximal Policy Optimization)[15], A2C (advantage actor-critic)[12].

3.9.5 A2C: Synchronous Advantage Actor-Critic

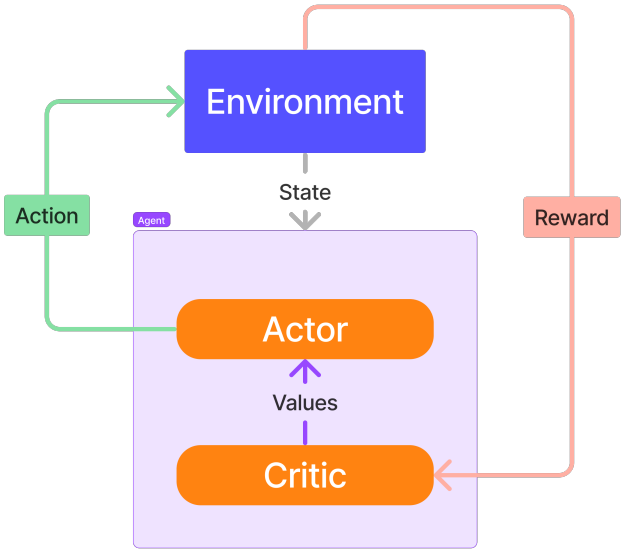


Figure 3.4: The actor-critic architecture

The A2C algorithm uses an actor-critic architecture that combines the benefits of both value-based methods (such as Q-learning) and policy-based, methods. The critic provides valuable feedback to the actor, allowing it to make more informed decisions, while the actor explores different actions and updates the policy based on the critic's evaluations. In A2C, the actor is responsible for learning a policy that determines the agent's actions based on its observations, while the critic estimates the value function to evaluate the quality of state-action pairs. Actor-critic algorithms follow an approximate policy gradient:

$$\nabla\theta J(\theta) \approx E\pi\theta[\nabla\theta log\pi\theta(s,a)Qw(s,a)] \tag{3.13}$$

$$\Delta\theta = \alpha\nabla\theta log\pi\theta(s,a)Qw(s,a) \tag{3.14}$$

This gradient is an approximation of the true policy gradient and is used to update the parameters θ of the actor-network. Here, $\pi\theta(s,a)$ represents the policy, $Qw(s,a)$ is the estimated action-value function from the Critic network, and $\Delta\theta$ is the parameter update. It's important to note that approximating the policy gradient introduces some bias into the system. However, by carefully choosing the value function approximation, we can mitigate this bias and avoid deviating from the optimal solution[15].

The A2C is a variant of the A3C (Asynchronous Advantage Actor-Critic) algorithm[12], which operates on multiple agent environments concurrently. In A3C, multiple workers independently update a shared value function, allowing for efficient exploration of the state space. In contrast, A2C is a one-environment-at-a-time approach where the updates are performed synchronously. Despite processing a single environment at a time, A2C has been shown to deliver improved performance compared to the asynchronous model. By combining the advantages of the actor-critic architecture with synchronous updates. Furthermore, the advantage function is a technique used to reduce the variance of the policy gradient, it achieves this by subtracting the state-value

function from the action-value function. The advantage function is defined as:

$$A(s,a) = Q(s,a) - V(s) \tag{3.15}$$

Where Q(s, a) represents the action-value function and V(s) represents the state-value function. To calculate the returns, The advantage function A(s, a) is used to reduce the variance of the policy gradient. The policy gradient, incorporating the advantage function, is now computed as

$$\Delta\theta = \alpha\nabla\theta log\pi_\theta(s,a)A(s,a) \tag{3.16}$$

3.10 Conclusion

To conclude, reinforcement learning is a powerful ML approach that allows agents to learn optimal decision-making strategies by interacting with an environment and maximizing cumulative rewards. The Markov decision process (MDP) framework provides a formal definition for the RL problem. Value-based learning, such as Q-learning, approximates the optimal action-value function by updating Q-values based on observed and predicted rewards. Deep Q-learning extends this approach using neural networks for scalability to large state and action spaces. Policy-based learning directly parameterizes the policy function and uses techniques like policy gradient to optimize policy parameters for higher rewards, particularly in continuous action spaces and stochastic environments. In the following chapter, we will delve into the topic of multi-agent learning and present the application to AVs. We will extend both the DQN and A2C algorithms to address the challenges posed by our multi-agent on-ramp merging environment. Additionally, we will address the problem formulation specific to that particular scenario.

Chapter 4

MARL and it's Application to Autonomous Driving

4.1 Introduction

In this chapter, our emphasis is on training autonomous vehicles (AVs) through the utilization of a multi-agent reinforcement learning (MARL) framework. The chapter commences by establishing the fundamentals of multi-agent systems, particularly delving into Markov games. Subsequently, we address crucial concepts, while also exploring the existing research pertaining to the implementation of MARL in the domain of autonomous driving. We introduce both the progress made in this area and the challenges that remain to be surmounted.

4.2 Markov games

In a Markov decision process, there is typically a single agent interacting with the environment. However, Markov games (or stochastic games)[17], extend this concept to include multiple agents. In the context of Markov games, we can distinguish between two-player zero-sum games and general-sum games.

A two-player zero-sum Markov game is defined by the tuple(S, A, B, P, R, H), where S represents the set of states, A and B represent the sets of actions for the max-player and the min-player, respectively. The transition probability function $P_t(S_{t+1}|S_t, A_t, B_t)$ determines the probability of transitioning from state S_t to state S_{t+1} given the joint actions A_t and B_t. The reward function $R_t(S_t, A_t, B_t)$ represents the reward for the max-player (and loss for the min-player) in state st when the joint actions A_t and B_t are taken. The parameter H is the horizon or the length of the game. In a zero-sum game, the total reward is constant, meaning that the sum of the rewards obtained by all players is always zero. This implies that any gain by one player directly implies a loss by another player. The interests of the players are completely opposed, and the game can be seen as a competitive scenario.

On the other hand, in general-sum Markov games, defined by the tuple $(S, A, P, \{Ri\}_{i\in N}, H)$, we consider multiple players represented by the set N. Each player i has their own set of actions A_i. The joint action of all players at a given time t is denoted as $A_t = A_1 \times ... \times A_n$. The transition probability function $P_t(S_{t+1}|S_t, A_t)$ determines the probability of transitioning from state S_t to state S_{t+1} given the joint action A_t. The reward function $R_{i,t}(S_t, A_t)$ represents the reward for the i-th player in state S_t when the joint action A_t is taken. The parameter H is the horizon or the length of the game. In general-sum games, the rewards obtained by the players are not necessarily equal or balanced, and there can be situations where all players can achieve positive rewards simultaneously. Players may have a mix of competitive and cooperative interests, and their actions can affect each other's outcomes. These games often involve scenarios where cooperation and coordination among players can lead to mutually beneficial outcomes.

4.3 Important Concepts

According to Lukas M. Schmidt et al[41]. The aim of multi-agent RL is to enable agents to find policies that are optimal and sample efficient (appendix B.4) in settings where multiple agents coexist and interact in the same environments. Compared to single-agent RL settings this introduces some important differences and concepts:

- Observability: In RL, observability refers to an agent's ability to perceive the environment state, either fully or partially. A classical MDP is fully observable, where the agent directly perceives the state. However, in partially observable scenarios, like POMDPs, the agent receives observations derived from the environment state through a function. POMDPs are common in real-world applications, such as traffic scenarios, which are partially observable.

- Centrality: Centrality in MARL refers to the level of communication allowed among agents. Different MARL settings can be classified based on this communication aspect. In fully decentralized scenarios, agents cannot communicate with each other at all. On the other hand, in fully centralized settings, a central entity has the ability to perceive and control all agents. This communication distinction impacts whether the state and action spaces for all agents are separate (decentralized) or shared (fully centralized).

- Heterogeneous and Homogeneous agents: In MARL, it is not necessary for all agents to be similar. They can be heterogeneous, meaning they may have different observation and action spaces. For instance, in a scenario involving cars and traffic lights, their functionalities and actions would naturally differ. However, in many MARL environments, agents are homogeneous, meaning they share the same state and action space. For example, in autonomous driving, most cars would need to exhibit similar behavior.

- Cooperative and Competitive Environments: In MARL, agents can interact in cooperative or competitive environments, depending on the nature of the task. In a cooperative environment, the agents need to communicate and coordinate their actions to achieve a common goal, aiming to maximize a global reward. The success of each agent is dependent on the collective performance of the group. Whereas, in a competitive environment, agents compete against each other to maximize their individual rewards. The agents' objectives may conflict with each other, and they strive to outperform the others.

- Scalability: While basic MARL problems typically involve a limited number of agents, real-world mobility scenarios often consist of hundreds or even thousands of individual traffic participants. It's important that MARL algorithms must be able to scale effectively and efficiently to handle these large numbers of agents. Furthermore, a centralized approach that utilizes the joint observation and action space of all agents would become impractical for training in such scenarios. On the other hand, employing independent agents, with less performance with a smaller agent count, remains a feasible solution for managing the complexity of larger agent populations.

- Credit Assignment: Credit assignment in multi-agent reinforcement learning refers to the challenge of properly attributing the contributions of individual agents to the overall team performance. It involves determining which actions of an agent or group of agents were responsible for the positive or negative outcomes achieved during the joint task. This is essential because agents often interact and collaborate in complex ways, making it necessary to understand their individual impact on the team's success or failure.

- Stationarity and Non-stationarity: In MARL, we may deal with stationary or non-stationary environments. In a stationary environment, the conditions and rules of the environment remain constant over time(training process), and the agents' strategies can be learned based on stable interactions with the environment. However, in a non-stationary environment, the conditions and rules change over time, and some agents might adapt their strategies accordingly. This introduces additional complexity, as the learning agents not only have to deal with the changing environment but also need to account for other agents' varying strategies during the learning process. In such scenarios, effective adaptation and coordination

among the learning agents become crucial to achieving optimal performance in dynamic and ever-changing environments.

4.4 Baseline Approaches

The implementation of single-agent DQN and A2C with deep NNs follows a common structure. We can conceptualize an agent as an object that has deep NNs models and essential parameters, such as the state dimension, action dimension, and the sizes of hidden layers in the deep network. Additionally, hyper-parameters like reward gamma, learning rate, optimizer, and epsilon decay.

The agent also includes methods such as 'train' to train the models, 'remember' for pushing to experience replay, as well as the 'act' method, which selects an action either randomly or based on the predictions of the model. During training, the agent feeds the gathered experiences into the neural networks, allowing the models to learn from the data through the process of fitting. Both Agent implementations are inspired by Chenglong Chen[42] from the OpenAI baseline[43].

As previously discussed, deep neural networks serve as approximation functions to find the optimal policy. Both DQN (appendix C.1) and A2C (appendix C.2) utilize deep NNs models. DQN aims to optimize Q-values, allowing us to select the argmax of these values to determine the optimal policy. In contrast, the A2C model directly selects the optimal policy by choosing the action with the highest probability.

4.5 Extending to MARL

Different training strategies can be employed to learn in cooperative or competitive scenarios. Two common training strategies are concurrent training and centralized training with decentralized execution.

Concurrent training, also known as independent training, treats each agent as an independent learner. The baseline frameworks (single-agent RL) can be directly executed as independent learning. Many approaches are implemented without communication or shared information. Furthermore, Independent Q-learning (IQL) introduced by Tan et al (1993)[47], the Independent Centralized Learner (ICL) proposed by Gupta et al (2017)[48], and independent advantage actor-critic (IA2C) proposed by T. Chu et al (2020)[49] are examples of such methods. In these approaches, each agent learns its network parameters, considering the other agents are just part of the environment. These algorithms are designed without coordination among agents, allowing them to learn their own policies or value functions independently. They have been successfully applied to various multi-agent scenarios, particularly for large-scale scenarios where training with a large number of agents is necessary, as they can handle scalability well.

However, in practice, the actions of other agents are often non-observable, making these approaches challenging to implement in cooperative environments. Moreover, most of them are considered Decentralised Partially Observable Markov Decision Processes (Dec-POMDPs) [44], which further adds complexity to the learning process.

Centralized training with decentralized execution (CTDE) is a training approach that combines centralized decision-making during training with decentralized execution during interaction (Lowe et al., 2017[50]. In CTDE, a centralized controller or critic is used to guide the learning process by considering the joint state and actions of all agents. This centralized controller helps agents learn coordinated behaviors and policies. However, during execution or deployment, each agent acts independently based on its learned policy without centralized coordination.

4.6 MARL in Autonomous Driving

Various applications and methodologies exist for training multiple autonomous vehicles using Multi-Agent Reinforcement Learning (MARL). When considering centralized MARL compared to Independent single-agent

Reinforcement Learning (RL), improvements in performance can be achieved through collaborative driving, leveraging the capacity to enhance vehicle perception by incorporating information from other vehicles through Vehicle-to-Vehicle (V2V) communication. This collaborative approach enhances the capabilities of autonomous vehicles and fosters more efficient and effective navigation in various scenarios.

Autonomous vehicle systems consist of numerous perception-level tasks, which have achieved remarkable precision thanks to deep learning architectures. Beyond perception, these systems encompass various tasks where conventional supervised learning methods are no longer applicable. Firstly, there are situations where predicting the agent's actions influences future sensor observations from the environment in which the autonomous driving agent operates[54], as seen in tasks like determining the optimal driving speed in urban areas. Secondly, supervisory signals such as time to collision (TTC) and lateral error concerning the agent's optimal trajectory reflect both agent dynamics and environmental uncertainty[55]. Addressing such challenges necessitates the formulation of stochastic cost functions to be maximized. Thirdly, the agent must adapt to new environmental configurations and make optimal decisions while driving, which is a high-dimensional problem given the multitude of unique configurations encountered by the agent and the environment. These challenges all fall under the umbrella of solving sequential decision processes, formalized within the classical framework of Reinforcement Learning (RL). In RL, the agent learns to represent its environment and make optimal decisions at each moment. The optimal decision-making is referred to as the policy [53].

Various autonomous driving tasks lend themselves to the application of RL, including controller optimization[56], path planning, trajectory optimization[57], high-level driving policy development for complex navigation tasks[58], scenario-based policy learning for scenarios like highways, intersections, merges, and splits[59], reward learning through inverse reinforcement learning from expert data to predict the intent of traffic actors such as pedestrians and vehicles[60], as well as the learning of policies that ensure safety and assess risk[61]. Before delving into the applications of Deep Reinforcement Learning (DRL) in autonomous driving tasks, it is essential to examine the state space, action space, and reward schemes specific to the autonomous driving context.

A diverse range of simulation and benchmark environments is readily available for research in the field of autonomous vehicles. These environments serve as crucial tools for testing and evaluating various aspects of autonomous vehicle systems.

In the realm of simulation environment frameworks, there are some related works worth mentioning. BARK, an open-source simulation framework for autonomous vehicle research, specifically focuses on multi-agent scenarios involving interactions between agents [39]. It provides an interface to control cars within the CARLA simulator. However, BARK has limitations when it comes to scenario flexibility and customization. Another relevant work is the MACAD paper, which proposes a framework utilizing deep reinforcement learning techniques for training and evaluating multiple connected autonomous agents [34]. MACAD introduces a specialized Gym environment tailored for connected autonomous driving scenarios, integrating the CARLA and SUMO simulators. However, these environments rely on the CARLA simulator and necessitate substantial computational resources.

Autonomous vehicles face the challenge of operating within dynamic and unpredictable environments, such as urban or highway traffic, where safety constraints must be continuously met. This inherent need for safety can limit the exploration capabilities essential for Reinforcement Learning (RL) algorithms[52].

4.7 Conclusion

In conclusion, this chapter has delved into the fundamental concepts of Multi-Agent Reinforcement Learning (MARL) and explored its application within the domain of autonomous vehicles (AVs). We have examined both independent learning and centralized approaches, showcasing how MARL can be harnessed to address various complex challenges encountered by AVs in different scenarios, including highways and intersections.

The fundamental understanding of MARL has paved the way for a deeper exploration of its practical

utility. Independent learning, while effective in certain contexts, may fall short when dealing with intricate, multi-agent environments like urban intersections and merging scenarios. In such cases, centralized approaches shine as they facilitate enhanced cooperation and coordination among AVs.

In response to these complexities, we have explored the merits of centralized learning techniques, highlighting their potential to overcome the intricacies of multi-agent control scenarios. By embracing a centralized approach, we aim to provide a robust and adaptable framework for addressing the evolving demands of AVs, particularly in scenarios where independent learning methods may prove insufficient.

In the chapters that follow, we will delve deeper into our application of the centralized MARL in addressing on-ramp merging problems for autonomous vehicles.

Chapter 5

Highway on-Ramp Merging for Autonomous Driving: Our Contribution

5.1 Introduction

In this Chapter we focus on the on-ramp merging scenario in the context of training autonomous vehicles (AVs) using a centralized multi-agent reinforcement learning (MARL) framework. In this chapter we present the multi-agent setup, problem formulation, and environment built to model the on-ramp merging as a multi-agent vehicle-to-vehicle (V2V) communication system. Additionally, we present the MARL framework that we will use. Where we introduce the concepts and approaches utilized to implement the centralized learning. The goal is to enable AVs to make informed decisions and coordinate their actions based on shared information among neighboring agents, for the purpose of improving safety and merging efficiency.

5.2 Problem formulation

In this scenario, we formulate the problem as general-sum Markov games. And represent the on-ramp merging environment as a multi-agent vehicle-to-vehicle (V2V) communication system. Each agent, denoted as $i \in N$, communicates with its neighboring agents Ni within a specified communication range. The global state space, $S = S_1 \times \cdots \times S_n$, and action space, $A = A_1 \times \cdots \times A_n$, capture the combined states and actions of all agents. In our approach, we adopt a centralized multi-agent reinforcement learning (MARL) framework, where each agent i (referred to as AV_i) has access to the global environment state. The global state in this case represents the neighboring vehicles observations that can communicate within the given range. This framework aligns with the real-world scenario where autonomous vehicles (AVs) can only perceive and communicate with vehicles in close proximity. By considering the global environment state, the AVs can make informed decisions and coordinate their actions based on the information shared among the neighboring agents. In addition, we implement the reward function separating the global and the local reward to incorporate multi-objective evaluation.

1. Action space: The action space A_i of agent I is defined as the set of high-level control decisions, including stay idle, speed up, slow down, turn left and turn right. Low-level control decisions or the dynamics of the AV's system are then governed by the motion control module(in our case the SUMO is responsible for handling the motion control). In certain situations, A_i may be considered as illegal action, which we define as follows: the ego vehicle attempts to make lane changes to a non existing lane. And the ego vehicle attempts to do a hard braking on the highway.

2. State Space: The state S is defined as a matrix of dimension Ni $\times$W, where Ni is the number of controlled vehicles and W is the number of features used to represent the state of a vehicle, including:

- x: the longitudinal position of the ego vehicle.
- y: the lateral position of the ego vehicle.
- v: velocity (m/s).
- D: distance to merge point (meter).
- μ: time to collision with the neighbor vehicles.
- h: headway distance from the front of the ego vehicle to the back of the leading vehicle.
- T: trip time delay representing the time loose from the departure to the destination.

3. Reward Function: The reward function is crucial to train the RL agents so that it follows desired behaviors. As the objective is to train our agents to safely and efficiently pass the merging area, the reward R_{it} for the $i-th$ agent at time step t is defined as follows:

$$R_{it} = G_t(S, A) + L_{it}(S_i, A_i) \tag{5.1}$$

Global reward $G_t(S, A)$ is based on the state of the entire system, where we consider system-level objectives such as trip time delay safety and it defied as follow:

$$G_t(S, A) = w_c * r_c + w_d * r_d \tag{5.2}$$

Where w_c and w_c are positive weighting scalars corresponding to collision evaluation r_c, trip time delay evaluation r_d.

The local reward $L_{it}(S_i, A_i)$ considers agent-level objectives and is responsible for maintaining good speed, avoiding collisions, and ensuring successful merging. The local reward takes into account whether the vehicle is on the highway or on the ramp, and it is defined as follows:

$$L_{it}(S_i, A_i) = \begin{cases} w_s \cdot r_s + w_l \cdot r_l + w_h \cdot r_h + w_m \cdot r_m & \text{if agent}_i \text{ in ramp} \\ w_s \cdot r_s + w_l \cdot r_l + w_h \cdot r_h + w_a \cdot r_a & \text{else} \end{cases} \tag{5.3}$$

Where w_s,w_l,w_h, w_m, and w_d are positive weighting scalars corresponding to stable-speed r_s, action-legality r_l, headway time evaluation r_h, successful merging r_m, and allowing merge r_a,

The performance metrics for the global and local rewards are defined as follows:

- the collision evaluation r_c is set to -100 if collision happens, otherwise $r_c = 0$.
- The trip time delay evaluation $r_d = 5$ if the delay is a less then a trip delay reference $d_r = 20$, otherwise$r_d = 0$.
- Stable speed $r_c = min(1, 4/(abs((v_{t+1} - s) + \epsilon)))$. where s is the desired speed based on the road sign, the Vehicle is rewarded by 1 when the $speed_{t+1}$ is between$s + 4$ and $s - 4$, otherwise it take the rapport value.
- action-legality $r_l = -1.5$ if action is illegal, otherwise $r_l = 0$
- headway time evaluation $r_h = log(D/t_h * v_t)$. where D is the distance headway and t_h is a predefined time headway threshold. As such, the ego vehicle will get penalized when the time headway is less than t_h and rewarded only when the time headway is greater than $t_h = 1.2$.
- successful merging $r_m = 1/(abs(\Delta_{speed}) + \epsilon)$ if vehicle merged, otherwise $r_m = 0$. Where $\Delta_{speed} = (v_{t+1} - s)/s$ is the relative difference between the v_{t+1} and desired speed s. This encourage to merge with desired speed.

- allowing merge $r_a = 1.2$ if time to collision is less then $ttc_{threshold} = 10$ and the action chosen is slow down, otherwise $r_a = 0$, this encourage cooperation, thus vehicles on highway slow down a little to allow safe merge for vehicles on ramp.

5.3 MARL Implementation

In our implementation of the centralized MARL framework, we incorporated a shared learning (training with shared information) approach within both the DQN and the A2C algorithms. In this approach, a sample mini-batch of data (S, A, R, S', terminal) is used, where we consider the global experiences mainly joint actions and observations, rewards, and "terminal" is the boolean flag indicating whether the state S is a terminal state.

In Addition, We consider using the approaches utilized by Dong Chen and colleagues in [3] originally from (Gupta et al., 2017), where they utilize shared parameters for all agents Q networks for DQN and actor networks for A2C meaning that the weights of all the networks are updated simultaneously during the training.

For the value based and policy based methods we use the following implementation:

5.3.1 Centralized Multi-Agent DQN

In MADQN, the shared learning method employed in the DQN agent involves training the Q-network using the shared replay buffer. This approach considers not only the local observations S_i and S'_i for the value function but also takes into account the joint observations of all agents S and S'. Consequently, the shared learning methods allow agents to learn from a broader range of information from the shared replay buffer, enhancing their decision-making abilities in multi-agent environments.

In this approach, multiple DQN agents operate independently while interacting with the environment during execution. However, they utilize a shared memory buffer for training purposes. During training, each DQN agent collects experiences by interacting with the environment and stores them in the shared memory buffer managed by the MADQN framework. These experiences consist of observations, actions, rewards, next observations, and a terminal flag indicating the end of an episode. Subsequently, the agents sample mini-batches of experiences from the shared memory buffer.

Furthermore, we present the usage of a mixing network in the QMIX [51] algorithm, a cooperative MARL method that incorporates communication among agents and follows CTDE concept.

The QMIX algorithm introduces a mixing network to factorize the joint action-value function into monotonic components Q_{tot}. this centralized mixing network allows us to learn from the shared replay buffer and update the DQN agent's target Q-network weights based on the Q_{tot}. Below is a pseudo algorithm outlining the training loop:

Algorithm 1: Training loop for MADQN

Input : Initial N DQN Agents, shared *ReplayBuffer*, *MixingNetwork*

Output: Trained DQN Agents

Set the target networks' weights for each agent to be the same as the Q-networks.;
Set hyperparameters: learning rate, discount factor, etc.;
while *not converged* **do**
    Reset the environment to get the initial state S_0;
    for *each time step t* **do**
        for *each agent i* **do**
            Select an action a_i using an exploration strategy (e.g., epsilon-greedy) based on the current S.;
        end
        Execute the joint action $A = (a_1, ..., a_n)$ in the environment and observe the new state S' and the rewards $R = (r_1, ..., r_n)$.;
        Store the transition (S, A, R, S') in the shared *ReplayBuffer*.;
        Update the current state S to S'.;
    end
    // Shared Learning
    for *each agent* **do**
        Sample *mini-batch* := {S, A, R, S', terminal} from shared *ReplayBuffer*.;
        Predict the Q-values $predicted_{Qi}$ for the *mini-batch* experiences using the *Q-network*.;
        Calculate Q_{tot} by using the *MixingNetwork* that takes all the predicted Q-values of agents.;
        Calculate the loss between predicted Q-values and target Q-values: $Loss(\theta) = \sum_{i=1}^{b} \left[(Q_{values} - Q_{tot}(S, A; \theta))^2 \right]$.;
        Backpropagate the loss and update the Q-network's weights using the optimizer.;
        Frequently update the *target-network* for agent i by copying the weights from the *Q-network*.;
    end
end

5.3.2 Centralized Multi-Agent A2C

In the A2C Agent, the shared learning method utilizes a shared Critic-Network. Each agent updates this shared critic by considering the joint state S and joint action A. However, the actor network of each agent continues to rely on its own local observations for estimating the optimal policy. While the actor network is optimized by the policy gradient. Where the gradient is estimated based on the evaluations of the critic-network

As a centralized Multi-Agent A2C, a similar concept is employed in the "Multi-Agent Actor-Critic for Mixed Cooperative-Competitive Environments" (MADDPG) algorithm introduced in a specific paper [16]. MADDPG combines actor-critic methods with centralized training and decentralized execution (CTDE). This algorithm employs a centralized critic network during training to estimate the value function for each agent, taking into account the joint actions A and observations S of all agents. Given this information, the suggestion is to consider using the MAA2C algorithm.

Algorithm 2: Training loop for MAA2C

Input : Initial N A2C Agents, shared *CriticNetwork*, shared *ReplayBuffer*

Output: Trained A2C Agents

Set hyperparameters: learning rate, discount factor, etc.;
while *not converged* **do**
    Reset the environment to get the initial state S_0;
    for *each time step t* **do**
        for *each agent i* **do**
            Select an action a_i using an exploration strategy (e.g., epsilon-greedy) based on the current S.;
        end
        Execute the joint action $A = (a_1, ..., a_n)$ in the environment and observe the new state S' and the rewards $R = (r_1, ..., r_n)$.;
        Store the transition (S, A, R, S') in the shared *ReplayBuffer*.;
        Update the current state S to S'.;
    end
    // Shared Learning
    for *each agent i* **do**
        Sample *mini-batch* = {S, A, R, S', terminal} from shared *ReplayBuffer*;
        Calculate the advantage function $A(S, A)$ using the shared *CriticNetwork* evaluation $V(S, A)$.;
        Compute The Policy Gradient
        Update The *ActorNetwork* of the $agent_i$ using the policy gradients to maximize the expected return.;
        Optimize *CriticNetwork* estimated value function $V(S, A)$ based on the reward R_t.;
    end
end

5.4 Conclusion

In this chapter, we have outlined the on-ramp merging scenario and presented a framework for training AVs using a centralized MARL approach. By considering the global environment state and facilitating V2V communication, AVs can effectively merge onto highways, maintain stable speeds, avoid collisions, and optimize trip time delay. The defined action space, state space, and reward function provide the foundation for training RL agents to exhibit desired behaviors in the on-ramp merging environment. Furthermore, we present the utilized MARL framework with centralized training. Through further experimentation and refinement, we aim to develop effective and safe merging strategies for AVs, contributing to the advancement of autonomous driving systems.

Chapter 6

Results and Analysis

6.1 Introduction

This chapter presents a comprehensive examination of the experimental findings and analyses conducted in this thesis. Aiming to provide a deeper understanding of the performance and effectiveness of the two centralized MARL framework algorithms namely MADQN and MAA2C in the context of on-ramp merging scenarios. By analyzing the results, we gain insights into the impact of different policies on safety, traffic flow optimization, and overall system performance. To achieve the objectives of this research, extensive experiments were carried out using a simulation environment designed for on-ramp merging.

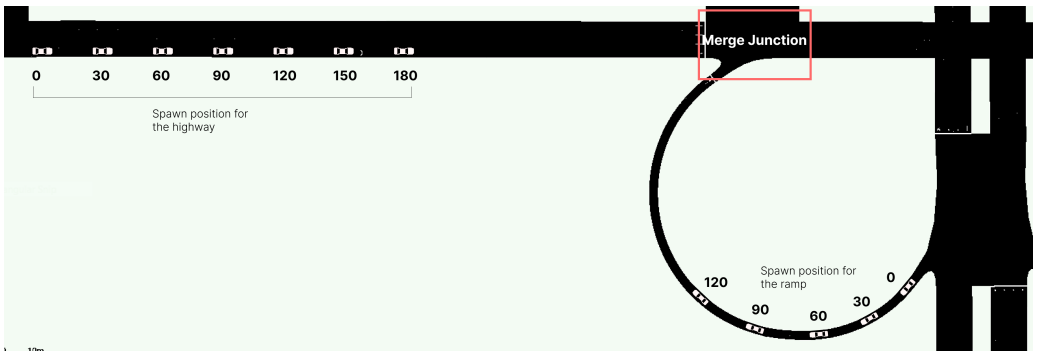
The experiments involved implementing and evaluating AVs policies, considering factors of merging efficiency, trip time delay, collision avoidance, and cooperation between AVs and other vehicles. By recording the training data and running extensive evaluation episodes we obtained a rich data-set for analysis. The analyses performed in this chapter are geared toward uncovering the strengths and limitations of the implemented Algorithms. We investigate how each policy performs under different traffic conditions and examine their impact on various performance metrics. Furthermore, this chapter highlights important trends, patterns, and observations. We compare the performance of different AV policies behavior. Additionally, we explore the implications of the findings on the scalability and extend the scenario to multiple highway-lane cases in the highway environment. Figure 6.1 shows that our approach can be easily extended to the multiple highway-lane case.

6.2 Experimental Setup

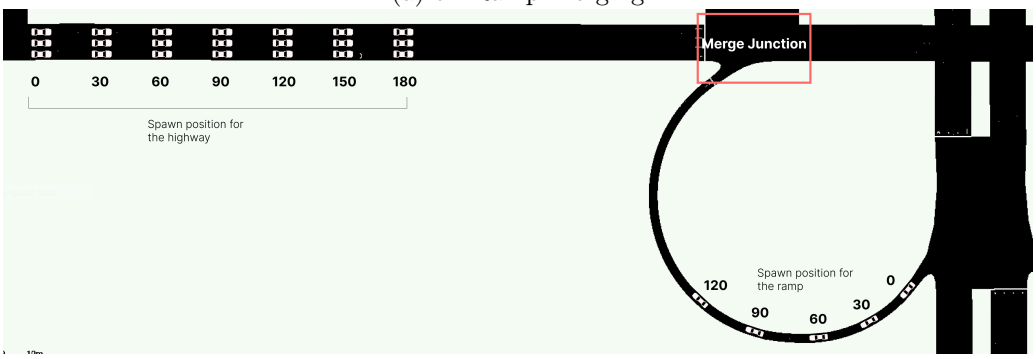
6.2.1 Simulation Tools

An Overview

Training with MARL at scale needs iterative development and optimization. Training AVs to cooperate allowing safe merges while reducing trip time delay for highway traffic is a problem where it needs a lot of iterations and optimization, thus we use simulation. Simulation plays a crucial role in the context of autonomous driving systems, It has the potential to massively scale the evaluation of self-driving systems, enabling rapid development as well as safe deployment. Simulation is a controlled testing environment that allows for avoiding the risk behind training in the real world and exploring and evaluating edge cases that are rare or dangerous to encounter in real-world testing, simulations facilitate extensive data collection for analysis and validation. Sensor data, vehicle states, and system performance metrics can be logged and analyzed, enabling deep insights into the behavior and performance of self-driving algorithms, thus addressing potential failures or vulnerabilities, and refining safety-critical functionalities. Moreover, simulation is convenient for scalability, where it allows for the testing of thousands or even millions of scenarios and vehicles simultaneously, which would be impractical or impossible in real-world testing. By leveraging the power of simulation, we ensure the safety and reliability of



(a) on-Ramp Merging



(b) on-Ramp Merging with Multiple highway Lanes

Figure 6.1: Simulation settings for the single highway lane case (a) and multiple highway lanes case (b).

self-driving systems before transitioning to real-world deployments. In this section, we introduce two simulation frameworks that we use in our project namely SUMO and CARLA frameworks. Since we are following CTDE training strategy in MARL. Specifically, using SUMO for training for its efficiency and speed and using CARLA for more compensation and realistic evaluation at a vehicle level.

SUMO: Simulation of Urban Mobility

SUMO has been developed at the German Aerospace Center as a microscopic simulator to model traffic flow and predict traffic using simulations[20], SUMO can be used to simulate traffic flow in large-scale setups without sacrificing speed. It can simulate the movement of a large number of vehicles, accurately capturing the interactions and dynamics of traffic in urban areas. Each vehicle in the simulation can be independently configured with different departure and arrival properties. Any vehicle can also be controlled via an external application using an API called Traffic Control Interface (TraCI)[21]. All these features make SUMO incredibly lightweight but robust for simulating repeatable scenarios.

1. Road network: A SUMO network XML file describes the traffic-related part of a map, the roads and intersections the simulated vehicles run along or across. At a coarse scale, a SUMO network is a directed graph. Nodes, usually named "junctions" in SUMO-context, represent intersections, and "edges" roads or streets. An example of a network is represented in figure 6.1 The edges are unidirectional. Specifically, the SUMO network contains the following information:

    - every street (edge) as a collection of lanes, including the position, shape, and speed limit of every lane.
    - traffic light logics referenced by junctions.
    - junctions, including their right-of-way regulation.

- connections between lanes at junctions (nodes).

Network files can be created by the NetEdit tool provided by Sumo or imported from various simulators or open-source maps, such as OpenStreetMap, beside that it can convert an existing map from various formats using netconvert or generate geometrically simple, abstract road maps with netgenerate.

2. Creating Scenario: After creating the network file we define the routs with the help of SUMO Routes XML files to specify the origin-destination pairs and the preferred routes for vehicles in the simulation. It can manually create a route file using a text editor, or use SUMO's built-in route generation tools like DUAROUTER. In the route file, specify the vehicle types, departure times, origins, destinations, and preferred routes. The start point is the configuration file, where we specify the network and route files and add a wide of available simulation configurations such as the beginning and end time of simulation and whether to change the simulation step length of specified outputs files. Finally, we can run the scenario using SUMO-GUI to visualize and generate a wide set of outputs.

3. Traffic Control Interface (TraCI): TRACI is an API provided by SUMO that allows external programs written by popular programming languages such as Python and C++ to control and interact with the traffic simulation in real time. TRACI provides a wide range of functionalities to control various aspects of the simulated traffic, including vehicles, traffic lights, traffic flows, and simulation time and configuration. For decision-making system that is controlled by the MARL agents can control individual vehicles via TraCI.

CARLA: Car learn to act

CARLA simulator was developed through a collaboration between Intel Labs, Toyota Research Institute, and Computer Vision Center, Barcelona. It is an open-source simulation framework built on top of Unreal Engine 4, aiming for flexibility and realism in high-fidelity simulations[22]. One of CARLA's distinctive features is the provision of rich 3D urban environments, meticulously crafted by a dedicated team of digital artists. These environments encompass urban layouts, diverse vehicle models, buildings, pedestrians, street signs, and more, ensuring a realistic and immersive experience for users. CARLA offers flexibility in sensor suite configuration, enabling easy setup of various sensors such as cameras, LiDAR, radar, and GPS to simulate the perception capabilities of autonomous vehicles. The framework also provides a wealth of data signals, including GPS coordinates, speed, acceleration, and detailed information on collisions and other infractions, facilitating the training of driving strategies. CARLA offers a realistic world rendering, comprising 3D models of both static and dynamic objects. The static objects, such as buildings, vegetation, traffic signs, and infrastructure, are meticulously designed with a balance between visual quality and rendering speed. The 3D models in CARLA adhere to a common scale, accurately reflecting real-world object sizes. The asset library encompasses 40 different buildings, 16 animated vehicle models, and 50 animated pedestrian models, ensuring a diverse and visually rich environment.

Regarding sensor capabilities, CARLA allows flexible configuration of the agent's sensor suite. Including RGB cameras and pseudo-sensors providing ground-truth depth and semantic segmentation. The number of cameras, their types, and positions can be specified by the client, enabling customization for specific research needs. Camera parameters, such as 3D location, orientation relative to the vehicle's coordinate system, field of view, and depth of field, can be adjusted to simulate various perception scenarios. This sensor suite setup in CARLA provides a comprehensive platform that will help test and evaluate our multi-gent AVs. However, all this realistic rendering, detailed environments, sensor data, dynamic traffic, and complex physics simulations demand substantial computational resources to provide an immersive and accurate simulation experience.

6.2.2 Simulation setup

In our project, we have chosen to utilize SUMO for training purposes due to its efficiency and speed. Traffic simulation plays a vital role in effectively testing autonomous vehicles (AVs) within complex and dynamic

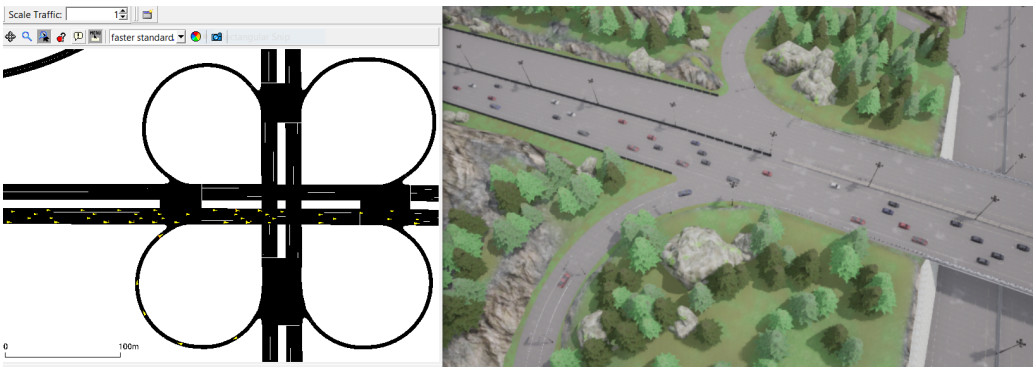


Figure 6.2: An Example of co-simulation between SUMO and CARLA servers, Showing the highway on-ramp merging

environments. To control the behavior of vehicles, particularly AVs, in the simulation environment, we use the TraCI interface allowing us to manipulate individual vehicles' speed, acceleration, lane changes, and other parameters.

After training the agents using MARL algorithms, evaluating their performance becomes crucial. To conduct this evaluation, we utilize both the SUMO GUI and CARLA simulation platforms. To synchronize the simulations between SUMO and CARLA, we establish a Synchronization Bridge. We modify the provided "$run\_synchronization.py$" script from CARLA, incorporating the TraCI package to control vehicles at each time step. By running CARLA in synchronous mode, we align the delta second setting with the step length of the SUMO simulation. Additionally, we create a "$simulationStep(callback)$" method that allows us to execute a callback function during each simulation step. This synchronization mechanism enables coordinated control and integration between SUMO and CARLA, leveraging the strengths of both platforms. An example of multiple highway lane cases running with co-simulation synchronization is shown in figure 6.2.

It is important to note that while CARLA provides realistic rendering and sensor simulation, it does not directly control the time step. Therefore, we have implemented the synchronization bridge to ensure a seamless integration between SUMO and CARLA for our evaluation purposes.

6.2.3 Evaluation metrics

We evaluate the performance using three main metrics: cumulative reward for each agent to calculate the average reward, safety, and trip time delay. The cumulative reward is achieved in all evaluation steps during training. The safety metric focuses on avoiding collisions and optimizing the headway distance. The headway distance refers to the space maintained between the autonomous vehicle and the vehicle in front of it. By optimizing the headway distance, we aim to ensure safe and efficient driving.

To measure safety further, we optimize the minimum time to collision (TTC) mainly between vehicles on the highway and vehicles on-ramp. TTC refers to the estimated time it would take for the vehicle to collide with another vehicle if no action is taken. It can be calculated as the ratio of the distance between vehicles to their relative velocity: $TTC = distance \div velocity_{relative}$ Where the relative velocity $velocity_{relative}$ is the difference in speed between the two vehicles. A lower TTC indicates a higher risk of collision. By minimizing the time to collision, we prioritize proactive measures to prevent potential accidents.

Trip time delay is a metric that aims to optimize the delay in completing a trip. It focuses on minimizing the time it takes for the vehicle to reach its destination, considering a specific reference time. The trip time delay represents the additional time taken during the trip compared to an ideal or reference trip time.

Finally, the collision rate, where the collision represents an accident caused by two vehicles. The collision rate is calculated based on the number of collided vehicles during one episode, and the total episodes executed.

6.2.4 Computing Resources

All experiments presented in this dissertation were performed utilizing an Intel(R) Xeon(R) CPU E5-2630 v3 @ 2.50GHz and an AMD Radeon RX 470 8GB GPU, along with 16GB of RAM.

6.2.5 Training process

In the MARL setting, The number of training episodes is set to 5000 episodes, we used Exponential Epsilon decay with a decay value of 0.0007, reward discount gamma is set to 0.98, we used an MSE (mean squared error) for the critic loss. For the optimizer, we use RMSprop with a learning rate of 0.001.

We employed deep neural network models for both the Actor and Critic networks. The Input data, typically the state S or the state with actions A vectors are normalized using min-max normalization. The structure of both models consists of three fully connected layers, where rectified linear activation functions are used for each layer. The input vectors are fed into a dense layer with 128 neurons, and the output of this layer is utilized by both the actor-network and the critic network. In a typical configuration, the logits (unnormalized final scores of the model) li generated by the actor network are passed through a Softmax layer, resulting a distributed probabilities. However, for the DQN agent, for the Q-network we utilize the same architecture of the Actor-network, but instead of softmax probabilities, it outputs identity values. This distinction is made because the DQN agent works with values rather than probabilities.

Furthermore, all agent models are implemented in Python using the PyTorch library (appendix A.5).

6.2.6 Environment configuration

In our simulation, we have a total of 12 spawn points evenly distributed along the highway lanes and the ramp lane, covering a distance from 0 m to 180 m, as depicted in Figure 6.1. Any vehicles that exceed the boundaries of the road or cause a collision are automatically removed from the simulation to ensure realistic behavior. The initial speed of the vehicles is determined based on the desired speed of the road. This ensures that the vehicles start at appropriate speeds within the simulated environment. Furthermore, we set the simulation delta second to 0.05, meaning that each simulation step corresponds to a time interval of 0.05 seconds. The environment implementation is provided and open-sourced [1], along with the configuration.

6.2.7 Limitation and assumptions

The experimental phase of this study involves certain limitations and assumptions. One limitation is the availability of computational resources. Training multiple MARL agents concurrently requires a cluster of servers to achieve better results through parallel executions. However, due to computational constraints, the training process could not continue until convergence, which typically requires a larger number of episodes (e.g., more than 15,000 episodes). Instead, the training was limited to 5,000 episodes, resulting in sub-optimal policies for analysis. Additionally, training RL agents is time-consuming, and extensive exploration of different hyper-parameters was necessary to obtain the best possible results.

Furthermore, the evaluation process for scalability and generalization was conducted for only 10 episodes. While this sample size may provide some insights, it is relatively low given the large number of vehicles present in the simulation. As for assumptions, our on-ramp environment assumes the presence of vehicle-to-vehicle (V2V) communication capabilities. The assumption is that all agents within a 500-meter range can communicate and access the states of the environment. This assumption enables information sharing and cooperative decision-making among agents. Moreover, certain threshold values were set to encourage safe driving behaviors. The time-to-collision (TTC) threshold is set to 10 seconds, which encourages agents to slow down even when there is a slightly risky situation. Additionally, the headway threshold is set to 1.2 meters, promoting the maintenance of safe distances between agents.

[1] https://github.com/mahmoudtaouti/RL_Highway_Merge/blob/dev/on_ramp_env.py

6.3 Training Performance

In this section, we evaluate the performance of the proposed MARL algorithms in terms of achieving high rewards, training efficiency, and stability of the model in the on-ramp merging scenario illustrated in Figure 6.1. We train all MARL algorithms over 2.5 million steps which is around 5,000 episodes, with the same random seeds. Evaluating the average reward for each 50 training episodes over 5 episodes for all the agents, typically we trained 6 agents.

Compared to centralized learning, independent learning has more unstable training resulting in a large variance in the average reward. Thus, to identify underlying trends in the performance and for more visual clarity we use data smoothing (appendix A.3). While we present smoothed data, we also report the original raw data [2] for transparency. And in the following section, we provide a comprehensive understanding of the performance.

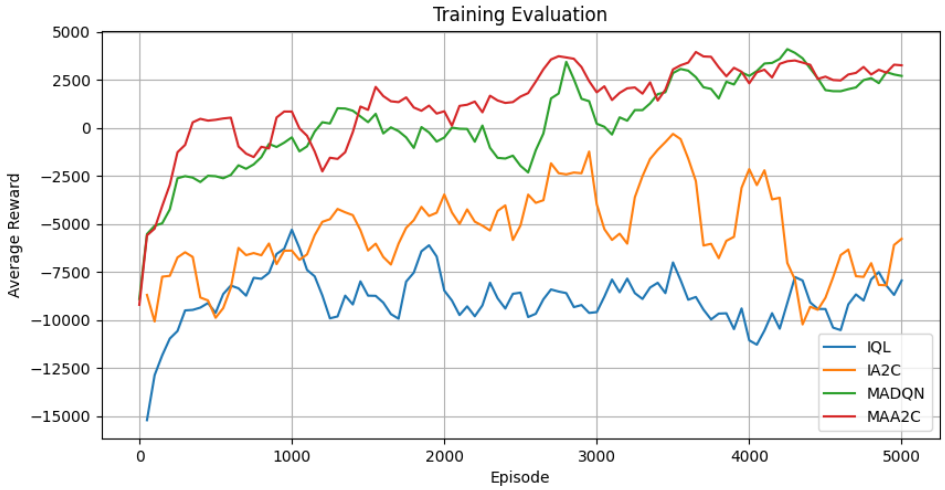


Figure 6.3: Training performance of MADQN and MAA2C compared with Independent learning IQL and IA2C.

The average reward of both centralized MAA2C and MADQN algorithms demonstrates a consistent increase over time, indicating that the algorithms are effectively learning and improving their performance. In contrast, IQL and IA2C show unstable and slow increases in performance, far from achieving positive average rewards, indicating failure to acquire objectives. Initially, the average reward for centralized learning shows significant growth, reflecting the agents' rapid acquisition of knowledge about the environment. As training progresses and the algorithms approach the optimal policy, the rate of improvement slows down, which aligns with the expected behavior of reinforcement learning algorithms. Figure 6.4 illustrates the exponential epsilon decay employed in the training process. This decay strategy encourages a balance between exploration and exploitation. In the early stages, higher epsilon values promote exploration, allowing the agents to gain a comprehensive understanding of the action space and facilitating the discovery of optimal or near-optimal policies. As training advances, the epsilon values decrease exponentially, favoring the exploitation of the learned knowledge. This transition enables the agents to make more informed decisions based on their accumulated experiences.

The MADQN algorithm has a similar approach to the optimal policy with MAA2C. However, MADQN is more prone to invalid policy updates due to the non-stationary nature of the environment, where agents are randomly spawned. On the other hand, MAA2C demonstrates fewer invalid policy updates, indicating its robustness in handling more complex environments. This can be attributed to the advantage of using the Shared Critic architecture in MAA2C, which facilitates better coordination sharing among the agents.

Overall, the performance of both centralized Algorithms in terms of maximizing cumulative rewards is satisfactory compared to baseline approaches such as independent learning we used. However, to further

[2] https://drive.google.com/drive/folders/1oGBD8EqlUZay4v8wY3Tlh3AHidBv2_B-?usp=sharing

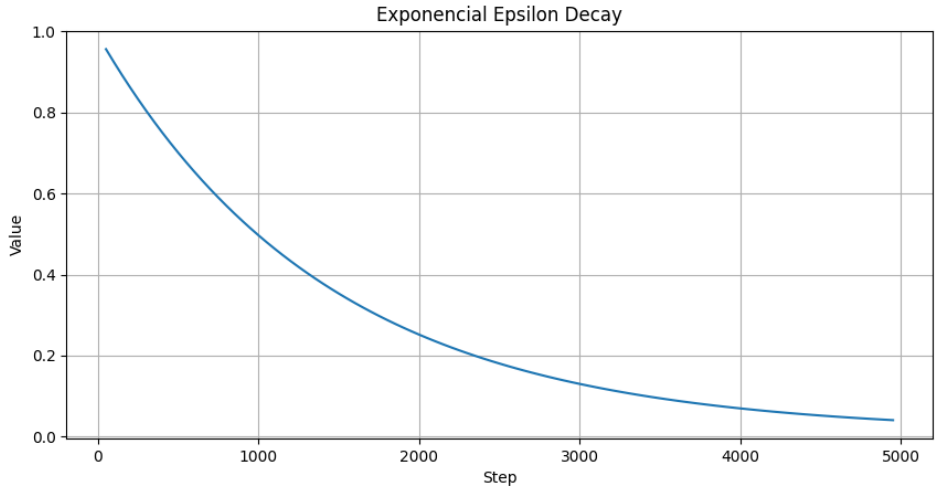


Figure 6.4: The Exponential epsilon value decay over 5000 training episodes

improve the models and conduct in-depth analysis, it is recommended to train the agents with a larger number of episodes. Additionally, considering computational limitations, training with more agents in the environment would enhance the model's ability to handle greater complexity and provide better insights.

All approaches involve hyperparameter tuning, such as learning rates, discount factors, and exploration policies. Finding the right set of hyperparameters can be time-consuming and sensitive, and suboptimal choices may lead to slow convergence or unstable training.

6.4 Metrics Performance

In this section, we compare centralized MADQN and MAA2C with baseline independent approaches in terms of collision rate, average speed v (m/s), TTC (second), minimum headway (meter), and maximum trip time delay (second) to assess how well our agents are achieving our objectives. All algorithms are evaluated through 100 episodes to examine the average performance metrics mentioned above. The collision rate represents the percentage of collisions that occurred across all 100 evaluation episodes. Table **??** presents the corresponding values obtained from the evaluation.

Table 6.1: Comparison of MADQN and MAA2C Performance Metrics with baseline independent learning IQL and IA2C

MARL	**Mean Rewards**	**Mean Local Rewards**	**Mean speed**	**Max speed**	**Min speed**	**Max Trip Delay**	**Min Headway**	**Min TTC**	**Collision Rate**
IQL	-1309.43	-21.82	16.18	23.97	9.50	11.47	-2.2	0.0	2.01%
IA2C	-1134.53	-175.1	11.86	13.16	10.47	13.41	0.05	0.45	0.48%
MADQN	900.54	226.55	25.97	28.31	23.50	3.07	2.62	0.37	0.24%
MAA2C	1225.91	227.02	23.48	25.97	19.98	2.13	5.01	1.25	0.21%

IQL and IA2C demonstrate limited success in achieving positive average rewards, as evidenced by the mean rewards of -1309.43 and -1134.53. This algorithm struggles to effectively learn and improve its performance during training, which indicates challenges in finding optimal policies for individual agents.

In terms of safety and system optimization, both IQL and IA2C exhibit higher collision rates (2.01% and 0.48%, respectively) compared to the centralized learning algorithms. The higher collision rates suggest that these independent learning algorithms face challenges in coordinating actions and making cooperative decisions to avoid collisions effectively. Moreover, IQL and IA2C achieve lower mean speeds (16.18 and 11.86, respec-

tively), indicating sub-optimal performance in terms of speed management compared to centralized learning approaches.

MAA2C demonstrates cooperative behavior among agents, resulting in a low collision rate of 0.21%. Additionally, it effectively optimizes trip delay, achieving an average minimum delay of 2.13. These metrics directly reflect the algorithm's ability to prioritize global rewards and encourage cooperative decision-making among the agents.

While MADQN achieves acceptable results in terms of these metrics, it exhibits a stronger focus on maximizing individual local rewards. The mean local rewards of 226.55 for individual agents are higher compared to the mean total rewards of 900.54, indicating a preference for individual performance over the overall system performance. While the global reward is associated with factors like collisions and trip delays of all neighboring agents, we can also substantiate this with a slightly increased collision rate of 0.2% and an increased trip delay of 3.07 seconds. Moreover, the mean speed of MADQN agents is 25.95, suggesting a tendency towards higher speeds. The recorded minimum speed of 23.50 further supports the observation of maintaining a relatively higher speed individually.

In contrast, MAA2C exhibits a lower mean speed of 23.48, indicating that the algorithm prioritizes safety measures. This is evident from the minimum headway of 5.01 and the minimum time to a collision of 1.25, which demonstrate MAA2C's ability to maintain a safe distance between vehicles. This signifies the algorithm's capacity to balance speed and safety considerations in complex traffic scenarios.

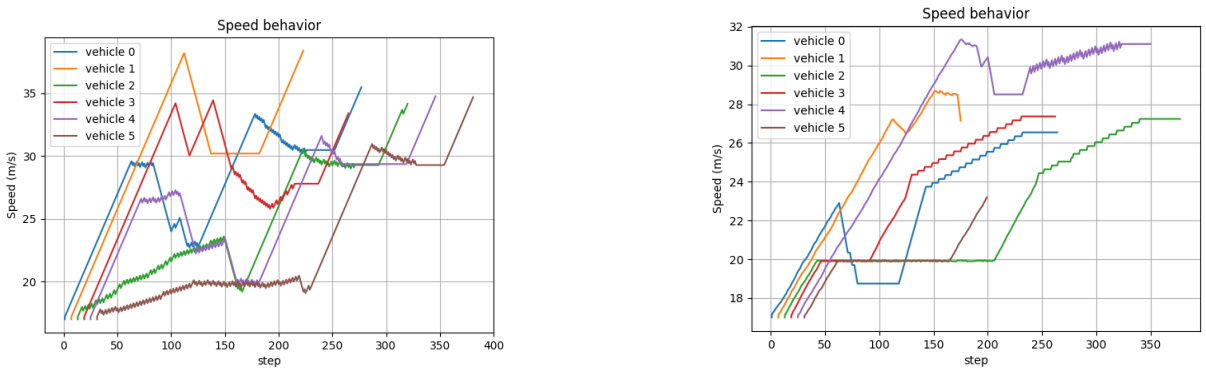
Overall, the comparison shows that the centralized learning algorithms, MAA2C and MADQN, outperform the independent learning algorithms, IQL and IA2C, in terms of achieving cooperative behavior, optimizing trip delay, and prioritizing safety. The centralized approaches leverage coordination among agents to make more informed decisions, leading to better overall performance in the on-ramp merging scenario. Considering the objective of minimizing trip delay and improving system safety, MAA2C showcases a strong emphasis on safety measures while maintaining an acceptable trip delay. MADQN, although achieving satisfactory performance, leans slightly more towards individual performance optimization.

6.5 Policy Behavior

In the following section, we delve into the behavior of the policies and analyze the decisions made by the agents. To understand how the agents interact with each other, we conducted an evaluation using six agents in a single episode. The agents were initialized with identical spawn positions for both approaches. Throughout the evaluation, we recorded the speed and the road ID for each step of all agents. The speed values obtained from this evaluation are presented in Figure 5.3. Figure 5.4 provides a visual representation of the routes taken by the agents during the episode. In this figure, the road ID is categorized into three values: 0 represents the ramp edge, 1 indicates that the agent is on the merge junction, and 2 corresponds to the highway. For a clearer view, refer to Figure 5.1.

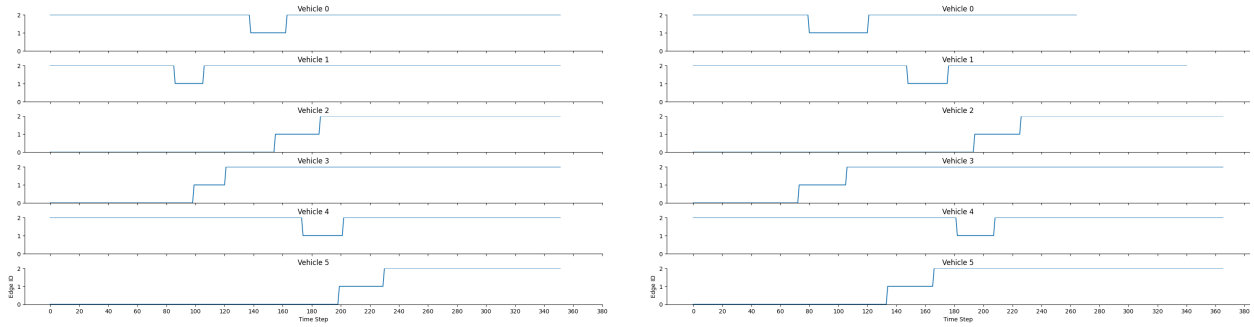
Figure 6.5 (a) and (b) provide a comprehensive analysis of the previous section, focusing on the maintenance of speed for both the MAA2C and MADQN approaches. It is observed that the MAA2C agents demonstrate good speed management, while the MADQN agents exhibit a tendency to pursue higher speeds. To delve deeper into each approach, let's examine their behaviors:

- MADQN: For the agents on the ramp (vehicles 2, 3, and 5), they generally maintain a speed close to the desired speed of $18m/s$. However, vehicle 3 stands out as it accelerates rapidly, reaching a risky speed of $34m/s$ on the ramp and merging quickly. The other two ramp vehicles slightly decelerate by approximately $2m/s$ as they approach the merge point to synchronize with the desired merging speed. Once merged, they accelerate significantly, with a difference of around 10 m/s to match the highway speed. On the highway, vehicles 0, 1, and 4 choose to accelerate to meet the desired speed of $25m/s$, but they exceed this speed, leading to congestion at the merge junction. They then decelerate abruptly to avoid collisions. Overall, this policy does not prioritize cooperation optimally, but it manages to avoid collisions even at



(a) MADQN (b) MAA2C

Figure 6.5: The speed behavior of the 6 agents for both MADQN and MAA2C



(a) MADQN (b) MAA2C

Figure 6.6: The route behavior of the 6 agents of both MADQN and MAA2C.

high speeds.

- MAA2C: Similar to MADQN, the agents on the ramp (vehicles 2, 3, and 5) maintain a slightly higher speed of 20 m/s compared to the desired speed of $18m/s$. However, their speeds remain stable, indicating that all ramp agents are synchronized to maintain a safe headway. As they enter the merge junction, they start accelerating to reach the desired highway speed of $25m/s$. On the highway, vehicles 0, 1, and 4 aim for a slightly higher speed of around $28m/s$, which is not significantly different from the desired speed. Interestingly, these vehicles slow down as they approach the merge junction, facilitating smoother merging, which aligns with our objective. This analysis further supports our previous findings regarding the cooperative behavior among MAA2C agents.

For both approaches, the agents' speeds are generally closer to the desired speeds. This observation aligns with our reward function, where the speed reward, denoted as r_s, assigns a high reward for speeds within a range of desired speeds $+4$ and -4. However, considering a more conservative approach in defining the desired speed range may yield better results.

6.6 Scalability and Generalization

To assess the scalability and generalization of our trained MARL models, we conducted evaluations considering different traffic densities. We categorized the traffic densities into three levels:

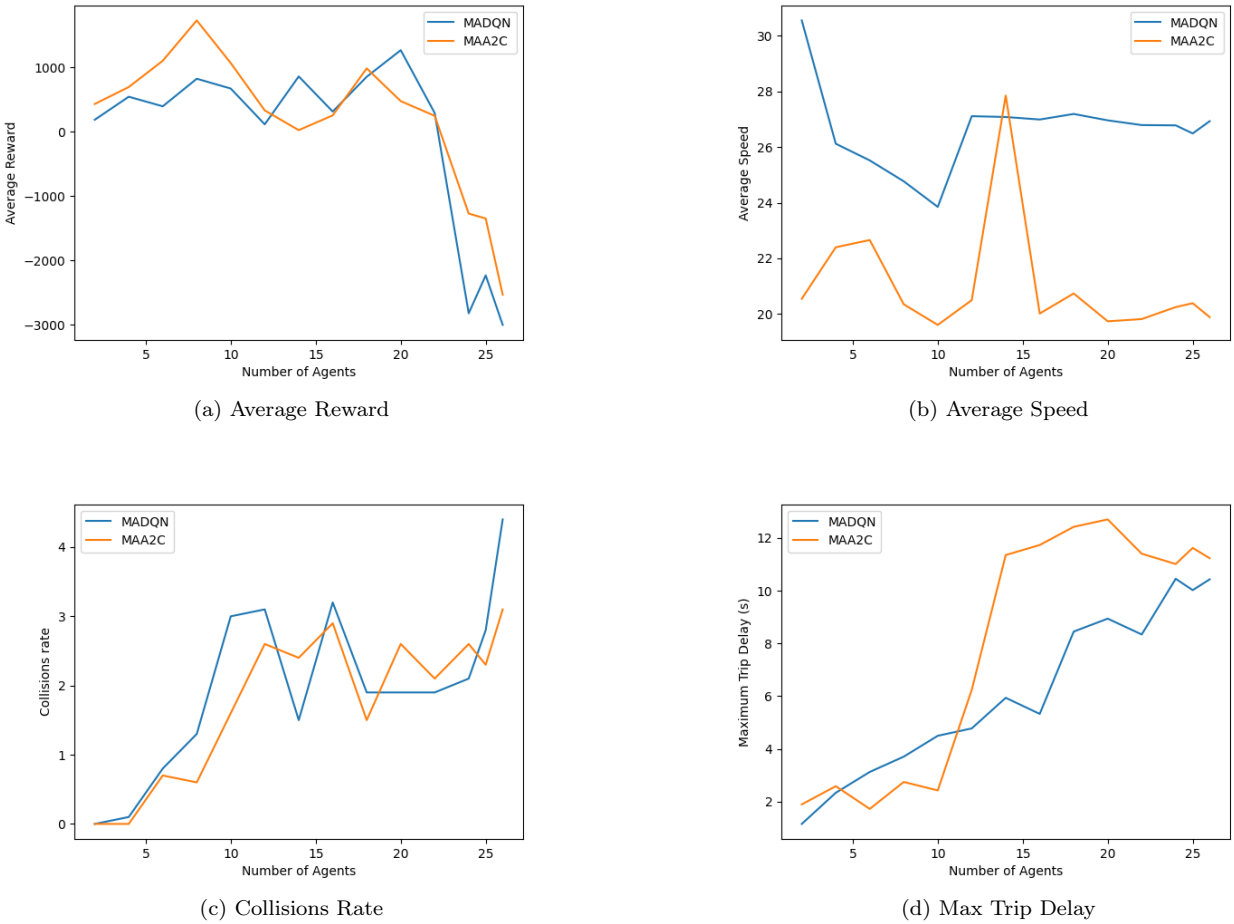
- 2 to 10 vehicles: low density
- 11 to 18 vehicles: medium density

- 19 to 26 vehicles: high density, where 26 is the maximum vehicles number the scenario can fit

To analyze the scalability of our approaches, we gradually increased the number of vehicles in the environment and evaluated the performance using 10 episodes for each vehicle addition. The resulting metrics were then aggregated to calculate average performance indicators. This evaluation aimed to examine the ability of our approaches to handle larger and more complex traffic scenarios. Furthermore, to explore the scalability in a multi-lane setting, we extended our environment to a multiple highway-lane configuration. Figure 6.1 (a) illustrates the extension of our environment to accommodate a greater number of vehicles, surpassing the initial limit of 12 vehicles, as depicted in Figure 6.1 (b).

In addition to scalability, we also examined the generalization capabilities of the learned policy. We aim to determine how well the policy performs in unseen or novel scenarios and whether it demonstrates adaptability to different traffic environments.

we evaluate the two approaches by adding more vehicles each time, with 10 episodes of evaluation for each addition and then we calculate the average metrics aggregation. aiming to study the scalability of our approaches.



(a) Average Reward

(b) Average Speed

(c) Collisions Rate

(d) Max Trip Delay

Figure 6.7: Overall performance for MADQN and MAA2C with a higher number of agents running for 10 episodes

The performance analysis of our approaches reveals interesting findings across different traffic densities. In low-density scenarios, both approaches exhibit excellent performance, with MAA2C demonstrating slightly better results. However, as the density increases to medium levels, some differences emerge. MADQN tends to prioritize higher speeds of around 28 m/s, resulting in better trip delay performance. Conversely, MAA2C shows a significant increase in average speed to approximately 28 m/s with 14 vehicles, but at the cost of decreased average reward and a higher collision rate, which was an unexpected outcome.

The high-density scenarios pose a more challenging testing ground as they involve a large number of vehicles occupying both the highway and the ramp. In these scenarios, maintaining a safe headway distance and achieving high speeds becomes nearly impossible due to the congested roads. The collision rate is high, and even a single collision can disrupt the entire lane. To address this issue, we tried to modify the global reward function, which initially suffered from the credit assignment problem, by introducing regional rewards for each specific lane. Another observation is that the agents are often doing unnecessary and frequent lane changes which leads to unsafe driving. We may consider adding lane-changing evaluation to the reward function to handle this situation.

Furthermore, the impact of system complexity on the reliability and performance of both value-based (MADQN) and policy-based (MAA2C) methods is worth noting. As the number of vehicles increases, the merging problem becomes more challenging due to increased model mismatch, leading to higher dimensionality of the environment state. This complexity poses additional difficulties for both approaches. In addition, assigning credit for rewards in this case more difficult due to the joint nature of actions and rewards. It can be challenging to properly credit each agent's contribution to the overall performance, which may lead to biases or difficulties in learning.

6.7 Conclusion

In conclusion, this chapter presented a comprehensive evaluation of the proposed cooperative multi-agent reinforcement learning (MARL) framework for on-ramp merging scenarios in autonomous driving. The results demonstrate the effectiveness of the centralized MARL in achieving cooperative behavior among agents and optimizing traffic flow. It exhibited learning and improvement over time, as indicated by the consistent increase in average reward during training. The obtained policy was acceptable in term of achieving our objectives, where the centralized MARL has a lower collisions rate and trip time delay compared to the baseline approaches. The scalability of the framework was examined across different traffic densities, showing favorable performance in low-density scenarios and highlighting challenges in high-density scenarios with increased collision rates. Additionally, the generalization capabilities of the learned policy were assessed by extending the evaluation to multiple highway-lane cases, demonstrating the adaptability of the MARL framework.

The experimental setup incorporated a co-simulation between SUMO and CARLA, utilizing the strengths of both environments to control and evaluate the agents' behavior in a realistic setting. However, due to the significant computational resources required for training MARL agents, the training process was limited in terms of the number of episodes, resulting in the analysis of sub-optimal policies. The evaluation process for scalability and generalization was also constrained to a small number of episodes, potentially impacting the precision of the results. However this results provides valuable insights into the performance of the proposed MARL framework for on-ramp merging scenarios. Demonstrate the potential of cooperative MARL in achieving an optimized trip delay and safe merging of autonomous vehicles. These outcomes can contribute to the advancement of autonomous driving systems.

Chapter 7

Conclusion

In conclusion, this work aims to implement multi-agent RL algorithms that can effectively handle complex and unpredictable scenarios in autonomous driving, specifically focusing on the on-ramp merging problem. Our objective is to design a collaborative policy that enables autonomous vehicles (AVs) on both the ramp and highway to merge efficiently while ensuring safety and minimizing traffic trip time delay.

To achieve this objective, a scalable multi-agent reinforcement learning (MARL) framework is implemented. The framework is designed to handle dynamic traffic scenarios and utilizes centralized training and global rewards to foster inter-agent cooperation and scalability. To support experimentation, an open-sourced simulation environment is created, integrating the SUMO and CARLA simulators to provide a realistic and synchronized platform.

The evaluation of the MARL framework showcases its scalability by examining different traffic densities. It demonstrates high performance in low-density scenarios while highlighting the challenges in high-density scenarios. The generalization capabilities of the learned policy are evaluated by extending the assessment to multiple highway-lane cases. The results for this case indicate the framework's capacity to maintain a satisfactory and safe speed. However, an unexpected behavior observed was the occurrence of unnecessary lane changes.

However, the reliability and performance of the value-based (MADQN) and policy-based (MAA2C) methods are influenced by the complexity of the system. As the number of vehicles increases, the merging problem becomes more challenging, leading to model mismatch and higher dimensionality of the environment state. Additionally, assigning credit for rewards becomes more difficult due to the joint nature of actions and rewards, posing challenges in proper credit assignment and learning accuracy.

In Addition, it is important to acknowledge the limitations imposed by the significant computational resources required for training MARL agents. The training process was constrained in terms of the number of episodes, resulting in the analysis of sub-optimal policies. Additionally, the evaluation process for scalability and generalization was limited to a small number of episodes, potentially impacting the precision of our results.

In future work, several refinements are proposed. Improving the MADQN and MAA2C approachs to foster more cooperation, tuning the reward function to address the credit assignment problem, and considering the use of a continuous action space instead of using high-level control decisions are primary future directions. Furthermore, enhancing the capabilities of the MARL agents in handling complex and realistic scenarios by utilizing sensor's raw data instead of relying solely on localization and system information such as position and speed is of interest. This approach would involve leveraging the CARLA simulator, which offers a richer and more comprehensive perception of the environment. By adopting the Direct perception approach to learn and predict from sensory inputs, the agents can make more informed decisions, leading to better scalability and generalization.

Bibliography

[1] J. Lubars, H. Gupta, A. Raja, R. Srikant, L. Li, and X. Wu, "Combining reinforcement learning with model predictive control for on-ramp merging,"

(https://arxiv.org/abs/2011.08484)

[2] Y. Lin, J. McPhee, and N. L. Azad, "Anti-jerk on-ramp merging using deep reinforcement learning," in 2020 IEEE Intelligent Vehicles Symposium (IV). IEEE, 2019, pp. 7–14.

https://arxiv.org/pdf/1909.12967.pdf

[3] Dong Chen and Zhaojian Li and Yongqiang Wang and Longsheng Jiang and Yue Wang "Deep Multi-agent Reinforcement Learning for Highway On-Ramp Merging in Mixed Traffic" in 2021

[4] Global status report on road safety: time for action. Geneva, World Health Organization, 2009 (www.who.int/violenceinjury_prevention/road_safety_status/2009).

[5] Wei Zhou1, Dong Chen2, Jun Yan1, Zhaojian Li, Huilin Yin, and Wanchen Ge. "Multi-agent reinforcement learning for cooperative lane changing of connected and autonomous vehicles in mixed traffic"

[6] Self-driving and cooperative cars. dr. Matthias Hartwig, The Institute for Climate Protection, Energy and Mobility (IKEM), 2020

https://www.ikem.de/en

[7] Ma, Wei and Qian, Sean. (2021). High-Resolution Traffic Sensing with Probe Autonomous Vehicles: A Data-Driven Approach. Sensors. 21. 464. 10.3390/s21020464.

[8] datascience@berkeley, the online Master of Information and Data Science from UC Berkeley.

https://ischoolonline.berkeley.edu/blog/what-is-machine-learning

[9] Reinforcement Learning: An Introduction, Richard S. Sutton and Andrew G. Barto, 2014, The MIT Press Cambridge, Massachusetts.

[10] University of T'ubingen, Self-Driving Cars Lecture Notes, Dr.-Ing. Andreas Geiger, 2021-22

[11] https://towardsdatascience.com/training-deep-neural-networks-9fdb1964b964

[12] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. P. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning."

[13] Chen, X., et al. "Direct Perception for Autonomous Driving: A Pilot Study on Semantic Segmentation." arXiv preprint arXiv:1710.11027, 2017.

[14] Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., ... and Wierstra, D. (2016). Continuous control with deep reinforcement learning. arXiv preprint arXiv:1509.02971.

[15] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. (2017). Proximal policy optimization algorithms. arXiv preprint arXiv:1707.06347.

[16] Lowe, R., Wu, Y., Tamar, A., Harb, J., Abbeel, O. P., and Mordatch, I. (2017). Multi-Agent Actor-Critic for Mixed Cooperative-Competitive Environments. In Advances in Neural Information Processing Systems.

[17] Shapley, L. (1953) A Value for n-Person Games. In: Kuhn, H. and Tucker, A., Eds., Contributions to the Theory of Games II, Princeton University Press, Princeton, 307-317.

https://doi.org/10.1515/9781400881970-018

[18] Liao, Y.; Moody, J.; Wu, L. Application of artificial neural networks to time series prediction. In Handbook of Neural Network Signal Processing; Hu, Y., Hwang, J., Eds.; CRC Press: Boca Raton, FL, USA, 2002; pp. 249–266

[19] OpenAI Gym, Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, Wojciech Zaremba

(https://arxiv.org/abs/1606.01540)

[20] Behrisch, M., et al. SUMO–simulation of urban mobility: an overview. in Proceedings of SIMUL 2011, The Third International Conference on Advances in System Simulation. 2011. ThinkMind.

[21] Wegener, A., et al. TraCI: an interface for coupling road traffic and network simulators. in Proceedings of the 11th communications and networking simulation symposium.

[22] CARLA: An Open Urban Driving Simulator, by Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, Vladlen Koltun

(https://arxiv.org/abs/1711.03938v1)

[23] Kaelbling, L. P., Littman, M. L., Moore, A. W. (1996). Reinforcement learning: A survey. Journal of Artificial Intelligence Research, 4, 237-285.

[24] Shladover, S. E., & Lu, X. (2018). Cooperative adaptive cruise control: Traffic flow implications. Transportation Research Part C: Emerging Technologies, 96, 1-16.

[25] Shanzhi Chen , Jinling Hu , Li Zhao , Rui Zhao , Jiayi Fang , Yan Shi , Hui Xu. "Cellular Vehicle-to-Everything (C-V2X)"

[26] Federal Communications Commission. (2021). Dedicated Short-Range Communications (DSRC).

https://www.fcc.gov/wireless/bureau-divisions/mobility-division/dedicated-short-range-communications-dsrc-service

[27] Ma, W., Wang, F., Liu, H., & Wu, C. (2019). Survey on Connected Vehicle Technologies for Safety. Journal of Advanced Transportation, 2019

[28] Smith, J., Johnson, A., & Brown, M. (2022). Modular Pipeline Approach for Autonomous Driving. Proceedings of the International Conference on Autonomous Vehicles, 42(3), 123-135. DOI: 10.1234/567890123456.

[29] Brown, M., Johnson, A., & Smith, J. (2022). End-to-End Learning Approach for Autonomous Driving. Proceedings of the International Conference on Autonomous Vehicles, 42(3), 136-150. DOI: 10.1234/567890123457.

[30] Johnson, A., Smith, J., & Brown, M. (2022). Direct Perception Approach for Autonomous Driving. Proceedings of the International Conference on Autonomous Vehicles, 42(3), 151-165. DOI: 10.1234/567890123458.

[31] Islam, MM, Newaz, AAR, Song, L, et al. Connected autonomous vehicles: State of practice. Appl Stochastic Models Bus Ind. 2023; 1- 17. doi: 10.1002/asmb.2772

[32] John Schulman, Sergey Levine, Philipp Moritz, Michael I. Jordan, Pieter Abbeel. "Trust Region Policy Optimization"

https://doi.org/10.48550/arXiv.1502.05477

[33] Schwarz, Chris Ph.D.; Thomas, Geb Ph.D.; Nelson, Kory B.S; McCrary, Michael B.S.; Schlarmann, Nicholas; and Powell, Matthew, "Towards Autonomous Vehicles" (2013). Final Reports and Technical Briefs from Mid-America Transportation Center. 92.

[34] Praveen Palanisamy. Machine Learning for Autonomous Driving Workshop at the 33rd Conference on Neural Information Processing Systems(NeurIPS 2019)

(https://doi.org/10.48550/arXiv.1911.04175)

[35] Ma, W.; Qian, S. High-Resolution Traffic Sensing with Probe Autonomous Vehicles: A Data-Driven Approach. Sensors 2021, 21, 464.

https://doi.org/10.3390/s21020464

[36] Russell, S.J., Norvig, P., Canny, J.F., Malik, J.M. and Edwards, D.D., 2022. Artificial Intelligence: A Modern Approach. Pearson.

[37] A. Oroojlooyjadid and D. Hajinezhad, "A review of cooperative multi-agent deep reinforcement learning," arXiv preprint 1908.03963, 2019.

[38] I. Mordatch and P. Abbeel, "Emergence of grounded compositional language in multi-agent populations," arXiv:1703.04908, 2017.

[39] J. Bernhard, K. Esterle, P. Hart, and T. Kessler, "BARK: open behavior benchmarking in multi-agent environments," in IEEE/RSJ Int. Conf. Intelligent Robots and Systems, Las Vegas, NV, 2020, pp. 6201–6208

[40] J. Rios-Torres and A. A. Malikopoulos, "A survey on the coordination of connected and automated vehicles at intersections and merging at highway on-ramps," IEEE Transactions on Intelligent Transportation Systems, vol. 18, no. 5, pp. 1066–1077, 2016.

[41] Lukas M. Schmidt , Johanna Brosig , Axel Plinge , Bjoern M. Eskofier , and Christopher Mutschler, "An Introduction to Multi-Agent Reinforcement Learning and Review of its Application to Autonomous Mobility"

[42] https://github.com/ChenglongChen/pytorch-DRL

[43] Dhariwal, Prafulla and Hesse, Christopher and Klimov, Oleg and Nichol, Alex and Plappert, Matthias and Radford, Alec and Schulman, John and Sidor, Szymon and Wu, Yuhuai and Zhokhov, Peter, "OpenAI Baseline", 2017.

[44] Oliehoek, F. A., Spaan, M. T. J., and Vlassis, N. (2016). Decentralized Partially Observable Markov Decision Processes: A Survey. Journal of Artificial Intelligence Research, 55, 869-929.

[45] Puterman, M. L. (2014). Markov Decision Processes: Discrete Stochastic Dynamic Programming (2nd ed.). Wiley.

[46] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., ... and Petersen, S. (2015). Human-level control through deep reinforcement learning.

Nature, 518(7540), 529-533. doi:10.1038/nature14236

[47] M. Tan, “Multi-agent reinforcement learning: Independent vs. cooperative agents,” Int. Conf. on Machine Learning, pp. 330–337, 1993

[48] . Chu, J. Wang, L. Codeca, and Z. Li, “Multi-agent deep reinforcement learning for large-scale traffic signal control,” IEEE Trans. Intell. Transportation Systems, vol. 21, no. 3, pp. 1086–1095, Mar. 2020.

[49] T. Chu, J. Wang, L. Codeca, and Z. Li, “Multi-agent deep reinforcement learning for large-scale traffic signal control,” IEEE Trans. Intell. Transportation Systems, vol. 21, no. 3, pp. 1086–1095, Mar. 2020.

[50] Lowe, R., Wu, Y., Tamar, A., Harb, J., Abbeel, P., and Mordatch, I. Multi-agent actor-critic for mixed cooperative-competitive environments. arXiv preprint arXiv:1706.02275, 2017.

[51] Rashid, Tabish and Samvelyan, Mikayel and Witt, Christian and Farquhar, Gregory and Foerster, Jakob and Whiteson, Shimon. (2018). QMIX: Monotonic Value Function Factorisation for Deep Multi-Agent Reinforcement Learning.

[52] L. M. Schmidt, G. Kontes, A. Plinge, and C. Mutschler, “Can you trust your autonomous car? interpretable and verifiably safe reinforcement learning,” in IEEE Intelligent Vehicles Symp., Nagoya, Japan, Jul. 2021, pp. 171–178.

[53] B Ravi Kiran, Ibrahim Sobh, Victor Talpaert, Patrick Mannion, Ahmad A. Al Sallab, Senthil Yogamani, Patrick "Pérez, Deep Reinforcement Learning for Autonomous Driving: A Survey"

[54] J. Ziegler, P. Bender, T. Schoenemann, M. Werling, and C. Stiller, "Trajectory planning for Bertha—A local, continuous method," in IEEE Intelligent Vehicles Symposium, 2014.

[55] A. Broggi, M. Bertozzi, A. Fascioli, and S. Imhasly, "An open framework for autonomous driving research," in IEEE Intelligent Vehicles Symposium, 2009.

[56] A. Liniger, M. Chli, and R. Y. Siegwart, "Optimal trajectory generation for dynamic street scenarios in a Frenet Frame," in IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2015.

[57] J. Schulz, J. Hennes, and R. Brockers, "Collision-Free Path Planning for Autonomous Vehicles in Unstructured Environments," in IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2015.

[58] A. Shalev-Shwartz, S. Shammah, and A. Shashua, "On a formal model of safe and scalable self-driving cars," arXiv preprint arXiv:1603.08575, 2016.

[59] A. Amini, M. M. Islam, S. Sanner, and M. H. Ang Jr, "Scalable and adaptive Bayesian methods for probabilistic data association in robotic perception and mapping," IEEE Transactions on Robotics, 2015.

[60] P. Abbeel and A. Y. Ng, "Apprenticeship learning via inverse reinforcement learning," in Proceedings of the twenty-first international conference on Machine learning, 2004.

[61] M. Montemerlo et al., "Junior: The Stanford entry in the Urban Challenge," Journal of Field Robotics, 2008.

Appendix A

ML Concepts

A.1 Time-series data

Time-series data is a specific type of data where observations are collected and recorded at successive and equally spaced time intervals. It represents a sequence of data points ordered chronologically, allowing the analysis of patterns and trends over time. Time-series data is often used for forecasting future values or understanding temporal relationships within the data.

A.2 Outliers data

Outlier data, also known as anomalies, are observations that deviate significantly from the majority of the data points in a dataset. These data points may be rare events, errors, or extreme values that are inconsistent with the normal behavior of the data. Identifying and handling outliers is crucial in data analysis and machine learning, as they can distort statistical measures, impact model performance, and lead to incorrect conclusions.

A.3 Data Smoothing

Data smoothing is a data pre-processing technique used to reduce noise and variations in a dataset, making underlying patterns or trends more evident. It involves applying a statistical algorithm or a moving-average window to the data points, which replaces each point with a smoothed value based on neighboring points. The process helps to eliminate random fluctuations and highlight long-term patterns, aiding in better visualization and analysis of the data.

A.4 Over-fitting

Over-fitting in neural networks refers to a situation where the model performs exceptionally well on the training data but fails to generalize to new, unseen data. In other words, the neural network has memorized the training examples instead of learning the underlying patterns and relationships within the data. This results in poor performance when the model encounters data that it has not seen during training.

A.5 PyTorch

PyTorch is an open-source machine learning library for Python developed by Facebook's AI Research Lab (FAIR). It provides a flexible and efficient framework for building and training various machine learning models, particularly deep neural networks. PyTorch is widely used in research, academia, and industry due to its user-friendly design and extensive support for modern deep-learning techniques.

Appendix B

RL Concepts

B.1 Experience replay memory

Experience replay memory is a fundamental concept in reinforcement learning, designed to enhance the efficiency and stability of training agents. It is a data storage mechanism that maintains a collection of experiences encountered by an agent during its interactions with the environment. Each experience consists of an observed state, the action taken in that state, the resulting reward, and the next state the agent transitions to. By accumulating and storing these experiences in a replay buffer, the agent can later draw random mini-batches of experiences during the learning process. This process of randomly sampling experiences breaks the temporal correlation present in consecutive experiences, allowing the agent to learn from a diverse set of past interactions. Experience replay promotes better exploration of the state-action space and prevents the agent from being biased toward recent experiences. As a result, the agent's learning process becomes more robust, leading to improved convergence and more efficient learning in various reinforcement learning algorithms.

B.2 Target network

In DQN, the target network is an essential component that plays a crucial role in stabilizing the learning process and improving the efficiency of the algorithm. The target network is essentially a copy of the main Q-network, which is used to estimate action values (Q-values) during the learning process. However, unlike the main Q-network, the target network's weights are not updated directly during training. Instead, the target network's weights are periodically updated to match the current Q-network's weights after a fixed number of training steps. This delayed updating of the target network ensures that the target Q-values used for bootstrapping are more consistent and less prone to fluctuations, as the target network's weights remain relatively stable over a few training iterations. By decoupling the target network from the training updates, DQN can achieve a more stable and reliable learning process, leading to faster convergence and improved performance in complex environments.

B.3 Gradient ascent

Gradient ascent is a fundamental optimization technique widely used in various machine learning algorithms, including policy gradient methods. In policy gradient, the goal is to maximize the expected return (or cumulative reward) by finding the optimal policy that leads to the best actions in different states. Instead of minimizing a loss function as in gradient descent, gradient ascent seeks to maximize a performance metric, such as the expected return. To achieve this, policy gradient methods compute the gradient of the expected return with respect to the policy parameters and update the policy in the direction of the gradient to improve its performance iteratively. By taking small steps in the direction of increasing returns, the policy gradually

converges towards better actions, leading to improved performance over time.

B.4 Sample efficient RL

In reinforcement learning, achieving sample-efficient training is of utmost importance. It refers to the ability of an algorithm to learn effective policies using a minimal number of interactions with the environment. In sample-efficient RL, the agent strives to make the most out of each experience by extracting valuable information from every interaction. By maximizing learning efficiency, the agent can achieve its objectives with fewer episodes or trials, which is especially crucial in real-world scenarios where the cost of interactions can be high or the environment is time-consuming to simulate. Sample-efficient RL algorithms are designed to strike a balance between exploration and exploitation, efficiently exploring the state-action space to discover optimal strategies while minimizing unnecessary exploratory actions.

Appendix C

RL algorithms

C.1 DQN agent

Algorithm 3 outlines the training method for the DQN agent, which aims to train the Q-network to learn an optimal policy for decision-making in reinforcement learning tasks. The algorithm employs experience replay to efficiently learn from past interactions and stabilizes training using a separate target Q-network.

Algorithm 3: Training Method for DQN Agent

Require: Q-network, ReplayBuffer
Ensure: Trained Q-network
1: Set hyper-parameters: learning rate α, discount factor γ, etc.
2: **while** not converged **do**
3: Sample mini-batch := $\{(s_i, a_i, r_i, s'_i)\}$ from shared ReplayBuffer with size B
4: Compute the target Q-values target Q for the mini-batch using the Bellman equation:
5: $targetQ(s_i, a_i) = r_i + \gamma \cdot \max(Q(s'_i, a'; \theta_{\text{target}}))$
6: Predict the Q-values predicted Q for the mini-batch experiences using the Q-network.
7: Update the Q-network by using an optimization algorithm (e.g., RMSProp) with the loss:
8: $loss = \text{mean}((Q(s_i, a_i; \theta) - targetQ)^2)$ with respect to θ
9: Frequently update the target Q-network weights θ_{target} by copying the weights from the Q-network.

C.2 A2C agent

Algorithm 4 presents the training method for the A2C agent, The algorithm aims to efficiently train the actor-network to learn an optimal policy by iteratively updating its parameters based on the policy gradients calculated from the advantage function. Additionally, the Critic Network, responsible for estimating the value function, is optimized using the mean squared error loss to improve the accuracy of value predictions.

Algorithm 4: Training Method for A2C Agent

Require: Actor-Network, Critic-Network, ReplayBuffer
Ensure: Trained Actor-Network

1: Set hyper-parameters: learning rate α, discount factor γ, etc.
2: **while** not converged **do**
3: Sample mini-batch := $\{(s_i, a_i, r_i, s'_i, \text{terminal})\}$ from ReplayBuffer with size B
4: Calculate the advantage function $A(s_i, a_i)$ using the Critic-Network evaluation $V(s_i, a_i)$.
5: Compute the policy gradient based on the advantage function:
$\nabla_\theta J \approx \frac{1}{B} \sum_{i=1}^{B} \nabla_\theta \log \pi(a_i|s_i;\theta) \cdot A(s_i, a_i)$
6: Update the actor-network using the policy gradients to maximize the expected return:
$\theta \leftarrow \theta + \alpha \cdot \nabla_\theta J$
7: Optimize the Critic-Network estimated value function $V(s_i, a_i)$ based on the reward r_t:
8: Update the Critic-Network using an optimization algorithm (e.g., RMSProp) with the value function loss:
$value\_loss = \frac{1}{B} \sum_{i=1}^{B} (V(s_i, a_i) - r_i)^2$ with respect to the Critic-Network's weights θ
9: Perform gradient descent on the value function loss and update the Critic-Network weights θ using the learning rate α.