AGH University of Krakow

FIELD OF SCIENCE: ENGINEERING AND TECHNOLOGY

SCIENTIFIC DISCIPLINE: AUTOMATION, ELECTRONIC, ELECTRICAL ENGINEERING AND SPACE TECHNOLOGIES

DOCTORAL THESIS

Applications of reinforcement learning methodologies to autonomous driving

Author: Mateusz Orłowski

Supervisor: dr hab. inż. Paweł Skruch, prof. AGH
dr inż. Krzysztof Kogut

Completed in: Faculty of Electrical Engineering, Automatics, Computer Science and Biomedical Engineering

Kraków, 2023

AKADEMIA GÓRNICZO-HUTNICZA IM. STANISŁAWA STASZICA W KRAKOWIE

DZIEDZINA: NAUK INŻYNIERYJNO-TECHNICZNNYCH

DYSCYPLINA: AUTOMATYKA, ELEKTRONIKA, ELEKTROTECHNIKA I TECHNOLOGIE KOSMICZNE

ROZPRAWA DOKTORSKA

Aplikacja metod uczenia przez wzmacnianie do zagadnienia jazdy automatycznej

Autor: Mateusz Orłowski

Promotor rozprawy: dr hab. inż. Paweł Skruch, prof. AGH
Promotor pomocniczy: dr inż. Krzysztof Kogut

Praca wykonana: Wydział Elektrotechniki, Automatyki, Informatyki i Inżynierii Biomedycznej

Kraków, 2023

Foremost, I extend my sincere appreciation to my esteemed team colleagues—Tomasz Wrona, Nikodem Pankiewicz, Wojciech Turlej, Paweł Kowalczyk, and Michał Sokół — for their professionalism and the camaraderie that permeated our collaborative efforts in implementing machine learning methodologies within the realms of ADAS and AD functions.

I am deeply grateful to my supervisors, dr. hab. inż. Paweł Skruch and dr. inż. Krzysztof Kogut, whose steadfast support, unyielding trust, and constructive feedback accompanied me throughout the entire journey of this research endeavour.

I would like to as well thank my employer, Aptiv Services Poland, for allowing me to participate in the industrial PhD programmeme and removing any obstacles along the way.

Lastly, I extend my heartfelt thanks to my wife, Wiktoria, whose unwavering faith has been a constant source of strength throughout this journey, and to my entire family and circle of friends. This achievement stands as a testament to the collective encouragement and inspiration provided by those closest to me.

Abstract

The autonomous driving (AD) field is currently one of the most advanced and active frontiers in technology development, which needs to address both perception and control problems. Today, AD cars are required to deal with more and more complex environments and scenarios, which often require a data-driven approach to solve. At the same time, reinforcement learning (RL) is a subfield of artificial intelligence which aims at developing intelligent agents capable of acting in predefined environments. This work summarises the research conducted in using reinforcement learning methodologies to control the motion of an autonomous car. By performing a series of experiments, we were able to test how different control approaches can be used in combination with the RL policy and what kind of road scenarios can be solved with such a methodology.

In the first experiment, we trained the agent to control the behaviour of the simulated car in a highway environment with the use of a high-level control interface, defining the manoeuvre and the velocity set point. Execution of this control has been in charge of deterministic, model-based methods. The agent's goal was to reach the lane-based goal, defined in a predefined distance in the shortest time, while adhering to traffic rules and optimising comfort. We examine how different strategies for executing agent action impact both functional performance and training efficiency. In the second experiment, an RL agent was trained to derive the path of a vehicle aiming to park itself at a predefined spot. With straightforward reward design and problem definition, the agent was able to park in complex parking scenarios, including parallel and perpendicular parking spots. In these experiments, we also tested the use of different neural network architectures and checked their impact on functional and computational performance. In the last series of experiments, we applied RL to a multi-agent coordination problem, where multiple cars need to navigate complex road scenarios, such as bottleneck or cross-road. All of the vehicles in the scene were controlled with the same RL trained policy and was able to derive successful strategies to navigate those challenging scenarios. We were able to show that using the reward-sharing mechanism, in which each agent was rewarded for its individual and group performance, improves the overall performance of the group and speeds up training.

In summary, we were able to demonstrate that reinforcement learning methodology can be successfully applied to the autonomous driving domain, although its application to the production environment requires a careful design of the whole system. However, we are of the opinion that the presented research proves that RL methodologies are applicable to the AD domain, and might be necessary to solve the most challenging road scenarios.

Streszczenie

Pojazdy autonomiczne są obecnie jednym z najbardziej zaawansowanych i aktywnych obszarów rozwoju technologicznego, który musi radzić sobie zarówno z problemami percepcji, jak i sterowania. Zastosowane w nich systemy sterowania muszą radzić sobie z coraz bardziej złożonymi scenariuszami drogowymi, które coraz częściej wymagają podejścia wykorzystującego uczenie maszynowe. Jednocześnie uczenie ze wzmocnieniem (z ang. RL) to dziedzina sztucznej inteligencji, która ma na celu tworzenie inteligentnych agentów zdolnych do działania w wcześniej zdefiniowanych środowiskach. Niniejsza praca podsumowuje badania przeprowadzone w zakresie wykorzystania metodologii uczenia ze wzmocnieniem do sterowania ruchem autonomicznego samochodu. Przeprowadzając serię eksperymentów, byliśmy w stanie sprawdzić, jak różne podejścia do sterowania mogą być wykorzystane w połączeniu z polityką opartą o uczenie przez wzmacnianie oraz jakie rodzaje scenariuszy drogowych można rozwiązać za pomocą takiej metodyki.

W pierwszym eksperymencie przeprowadzono uczenie agenta do kontrolowania zachowania symulowanego samochodu na autostradzie za pomocą wysokopoziomowego interfejsu sterowania, definiującego manewr i prędkość zadaną. Realizacja tej akcji spoczywała na algorytmach deterministycznych. Celem agenta było osiągnięcie docelowego pasa na zadanej odegłości w jak najkrótszym czasie, przy jednoczesnym przestrzeganiu przepisów ruchu i optymalizacji komfortu jazdy. Zaprezentowano, w jaki sposób różne strategie wykonania działań agenta wpływają zarówno na funkcjonalność, jak i efektywność treningu. W eksperymencie drugim agent został przeszkolony do określenia trasy pojazdu, mając na celu samodzielne zaparkowanie we wcześniej zdefiniowanym miejscu parkingowym. Przy użyciu naturalnej definicji nagrody opartej o osiągnięcie celu, agent był w stanie zaparkować w określonej pozycji w skomplikowanych scenariuszach parkowania, w tym miejscach parkingowych równoległych i prostopadłych. Dokonano również ewaluacji użycia różnych architektur sieci neuronowych i sprawdzenia ich wpływu na funkcjonalność i wydajność obliczeniową. W ostatniej serii eksperymentów uczenie przez wzmacnianie zastosowano do problemu koordynacji wielu agentów, gdzie kilka pojazdów musiało nawigować w skomplikowanych scenariuszach drogowych, takich jak zwężenia czy skrzyżowania. Wszystkie pojazdy uczestniczące w scenariuszu były sterowane tą samą polityką użytą w procesie uczenia i były w stanie opracować skuteczne strategie sterowania w wymagających scenariuszach. Byliśmy w stanie wykazać, że korzystanie z mechanizmu współdzielenia nagrody, w którym każdy agent był nagradzany za swoje indywidualne jak i grupowe osiągnięcia, poprawia ogólną skuteczność oraz przyspiesza sam process uczenia.

Podsumowując, przeprowadzone badania wskazują, że metodologia uczenia ze wzmocnieniem może być skutecznie zastosowana w dziedzinie jazdy autonomicznej, jednakże jej zastosowanie w środowisku produkcyjnym wymaga starannego zaprojektowania całego systemu. Jesteśmy jednak zdania, że przedstaw-

ione badania dowodzą, że metody uczenia przez wzmacnianie mogą zostać zastosowane w dziedzinie jazdy autonomicznej i potencjalnie mogą być konieczne do rozwiązania najbardziej wymagających scenariuszy drogowych.

Contents

List of Figures

List of Abbreviations

A3C Asynchronous Advantage Actor Critic 16, 17

ACC Adaptive Cruise Control 31, 58, 93

AD Autonomous Driving 1, 2, 18, 21, 25, 36, 46, 95

ADAS Advanced Driver-Assistance System 1, 21, 25, 95

AEB Autonomous Emergency Braking 28, 29

AGC Auto Gain Control 24

AI Artificial Intelligence 1

CC Cruise Control 58

CMOS Complementary Metal Oxide Semiconductor 24

DDPG Deep Deterministic Policy Gradient 16, 17

DNN Deep Neural Network 14

DP Dynamic Programing 12–14

DQN Deep Q-Learning 37, 91, 113

DRL Deep Reinforcement Learning 14

EKF Extended Kalman Filter 28

FC Fully-Connected 56, 78, 101

FMCW Frequency Modulated Continuous Wave 23

FSM Finite State Machine 36, 46, 49, 58, 59

GNSS Global Navigation Satellite Systems 25

IMU Inertial Measurement Unit 25

KF Kalman Filter 27, 28

LIDAR Light Detection and Ranging 22, 30, 87

LLF Low-Level Fusion 25

LQR Linear-Quadratic Regulator 31

LRR Long Range Radar 23

LSTM Long Short-Term Memory 56

MARL Multi-Agent Reinforcement Learning 96

MC Monte Carlo 13, 14

MDP Markov Decision Process 9, 11, 12, 14

MEMS Micro-Electromechanical System 22

ML Machine Learning 1, 2

MLP Multilayer Perceptron 78

MoD Mobility-on-Demand 1, 22

MPC Model-Predictive Control 30, 31

NN Neural Network 64, 78, 84, 86

PID Proportional Integral Derivative 31

POMDP Partial Observable Markov Decision Process 35, 68

PPO Proximal Policy Optimisation 16, 18, 55, 56, 68, 81, 89, 95, 101

ReLU Rectified Linear Unit 56, 78

RL Reinforcement Learning 1, 2, 5, 6, 46, 49, 50, 52, 58, 67

RSS Responsibility-Sensitive Safety 37, 50, 52, 54

SAC Soft Actor-Critic 17, 37

SAE Society of Automotive Engineers 19, 20

SGD Stochastic Gradient Decent 14

SLAM Simultaneous Localisation and Mapping 27, 28

SRR Short Range Radar 23

TD Temporal Difference 14

TRPO Trust Region Policy Optimisation 16

Summary of Notation

Capital letters are used for random variables, where the lower letters are used for concrete values of random variables and for scalar functions.

General

$\doteq$	equality relationship that is true by definition
$\mathbb{E}[X]$	expectation of a random variable X
$\mathbb{R}$	set of real numbers
$f : \mathcal{X} \to \mathcal{Y}$	function f from elements of a set $\mathcal{X}$ to elements of set $\mathcal{Y}$
$\text{argmax}_a f(a)$	a value of a at which $f(a)$ takes its maximal value
$\in$	is an element of; e.g. $s \in \mathcal{S}$
$\leftarrow$	assignment
α, β	step-size parameter; learning rate
γ	discount-rate parameter
ϵ	probability of taking a random action in an ϵ-greedy policy
λ	

Reinforcement learning

t	discrete time step
T	final time step of an episode
s, s'	states
a	an action
r	a reward
$\mathcal{S}$	set of all states
$\mathcal{A}$	set of all actions

$\mathcal{R}$	set of all rewards
A_t	action at time t
S_t	state at time t
R_t	reward at time t
G_t	return following time t
$p(s', r\|s, a)$	probability of transition to state s' with reward r from state s by taking action a
$p(s'\|s, a)$	probability of transition to state s' from state s by taking action a
$r(s, a)$	expected immediate reward from state s taking action a
$r(s, a, s')$	expected immediate reward on transitioning from state s to s' by taking action a
π	policy (decision-making rule)
$\pi(a\|s)$	probability of action a in state s under stochastic policy π
π_θ	policy in form of neural network parameterized by values θ
$\hat{R}_t$	estimates of rewards-to-go
$v_\pi(s)$	value of state s under policy π; expected return
$v_*(s)$	value of state s under optimal policy
$q_\pi(s, a)$	value of taking action a in state s under policy π
$q_*(s, a)$	value of taking action a in state s under optimal policy
$q_*(a)$	true value of taking action a - specific for multi-armed bandits problem
$\hat{Q}_t(a)$	estimate at time t of $q_*(a)$ - specific for multi-armed bandits problems
$\hat{V}_t(s)$	estimate at time t of $v_*(s)$
$\hat{Q}, \hat{Q}_t$	array of estimates of q_π or $q_*(s)$
$\hat{V}, \hat{V}_t$	array of estimates of v_π or $v_*(s)$
$\hat{A}$	array of estimates advantage estimation
$N_t(s)$	number of times state s has been visited prior to time t
l	loss function, objective
θ, θ_k	parameters of policy defined as neural network (in k-th iteration)
ϕ, ϕ_k	parameters of value function estimation defined as neural network (in k-th iteration)

D	set of experiences collected in environment
$\|D\|$	size of a experience set

Motion Models and Kinematics

x, y	position of a object / points in X and Y axis respectively
v	longitudinal velocity of an object / host along the car axis
v_x, v_y	velocity of an object in coordinate system aligned with host vehicle in X and Y axis respectively
a	acceleration along the object / host axis
r	turn radius
d	distance
ψ	orientation of object / goal; yaw
δ	front wheels angle in vehicle coordinate system
ω	yaw rate; $\omega = \dot{\psi}$
s	curve length
L	wheelbase - distance between two axis
t	discrete time step
Δt	time update; time to be simulated

Chapter 1

Introduction

1.1 Motivation

The Autonomous Driving (AD) for years has been a dream of both the engineering world and the general public. The concept of driverless cars was one of the hallmarks of a distant, undefined future. Nowadays, automated cars are no longer a futuristic vision from science fiction movies, but aim to become a generally accessible way of transportation. The Advanced Driver-Assistance System (ADAS) are standard equipment for most car brands, while some technology companies provide more advanced transportation services through their Mobility-on-Demand (MoD) fleets. As increasingly advanced products and services become more and more accessible and affordable, it is easy to forget about the immense complexity of the technology that stands behind each of such systems.

For the initial years of AD development, most of the focus has been on the perception of the outer world, which is the foundation of the whole system. At those times, feature functions have been rather limited in scope and capability, therefore, the methods used to realise those functions were equally straightforward. As perception abilities grew and cars acquired access to more information sources (such as precise maps), prospects for elaborate control functions in more difficult scenarios arose. At that moment, straightforward methods became insufficient as planning the motion became a challenge in itself.

Planning the motion of a car is a multi-agent, closed-loop control problem, without a one-and-only good solution. The resulting trajectory must meet a diverse set of requirements, including those related to safety, traffic rules, comfort, and efficiency in the vastness of road scenarios. The requirements concerning safety and traffic rules suggest the use of handwritten heuristics. The ones connected with comfort and efficiency favour optimisation control methods, where those qualities of behaviour can be directly targeted. Furthermore, the required adaptability to different scenarios and noisy data should be handled well with data-driven approaches.

Artificial Intelligence (AI) and Machine Learning (ML) methods are nowadays the cornerstone of state-of-the-art methods for dealing with perception tasks and have been successfully applied to automotive use case. Most of those methods rely on supervised learning methodology, where the machine learning algorithm uses labelled data sets to classify or predict specific qualities of the data. In parallel, Reinforcement Learning (RL) concerns intelligent decision making by training agents to interact with the environment to maximise

the defined reward signal. With the track record of success in popular games [76, 91, 122], only recently RL methods started to find applications in real world systems [30, 72, 90]. As RL methodology from definition concerns solving the closed-loop decision-making problem, its potential application to autonomous vehicle planning motion is a natural research direction.

Understanding the current state-of-the-art in both autonomous driving and reinforcement learning, a decision has been made to focus research on the applications of RL methodologies to the problem of motion planning of AD cars. At the same time, one of the prior goals of the work was to ensure that the proposed techniques and systems will have industrialisation potential and could be applied to motion planning systems in real cars. Understanding that a purely ML system will probably not address the requirements of a safety-critical system, attention has been paid to hybrid solutions.

1.2 Problem Statement

To properly direct the research and ensure proper planning of the experiments, a general thesis in the form of a theorem has been defined. As the selected research area is relatively new, its shape was general and acquired a form of high-level hypothesis, which will be examined. The thesis will be defined as follows:

Research Hypothesis. *The reinforcement learning methodology is applicable to solve decision-making and trajectory planning problems of autonomous driving vehicles. This statement will be tested and supported by the following claims:*

(i) *Controlling a car with high-level control interface by a reinforcement learning agent is possible.*

(ii) *Introduction of deterministic rules at the time of training improves the training time and the resulting policy.*

(iii) *Controlling the vehicle on a low level with the use of a direct path-planning interface by reinforcement learning agent is possible.*

(iv) *Multi-agent coordination of vehicle scenarios can be solved by reinforcement learning techniques.*

(v) *Making the individual agent's reward dependent on the objectives of other agents improves the overall average performance of all agents.*

To examine the main theorem and its claims, a series of experiments was conducted. Claims (i) and (ii) are examined in Chapter 3, where a behaviour planning agent has been trained. Claim (iii) has been validated in Chapter 4, where the problem of parking a car was considered. Claims (iv) and (v) have been evaluated in Chapter 5, where, using reinforcement learning, the movement of multiple vehicles in urban scenarios was coordinated by reinforcement learning policy.

1.3 Work Structure

The presented work is structured as follows. In Chapter 2 the core information about reinforcement learning methodology and autonomous driving has been introduced. The chapters 3, 4, and 5 are the core of the thesis,

each describing the experiment conducted. Each of those is organised with a problem statement definition, a literature review focused on a given sub-area, and a research description. Chapter 3 deals with the behaviour planning part and controlling the movement of cars with the use of a high-level interface. Parking scenarios, with a low-level path planning interface, have been investigated in Chapter 4. The aspect of multiple agents trying to coordinate their movements in urban scenarios has been investigated in Chapter 5. The conclusions about the research, including summarising contributions, understanding possible alternative research directions from the past, and thoughts about future steps, are placed in Chapter 6.

Chapter 2

Background

2.1 Reinforcement Learning Introduction

It seems that the most basic concept of learning, which humans are familiar with, is the idea of learning from interaction with the environment. The process of interaction is the source of information about cause and effect, strategies to achieve given goals, and the primary tool for gaining knowledge about the world. Even in cases when a given domain seems to be well known, practical interaction and real-world experiments are almost always necessary to fully master it.

In the next sections, a formalised computational approach to learning from interaction with the environment, called reinforcement learning (RL) will be described. Furthermore, the basic nomenclature used in this field will be introduced, and with that basic concepts the reinforcement learning methods will be described.

2.1.1 The Reinforcement Learning Problem

Reinforcement learning problems concern deriving an agent's policy, which tells what to do in a given situation, to maximise a defined reward signal. Almost from the definition, those problems are closed-loop, as the agent's actions impact the environment state and, through that, the next input. Moreover, in contrast to many other popular machine learning applications, the training process is not directly told which actions to take but instead has to discover on its own what actions will result in the highest rewards by trying them out. In the most interesting and challenging cases, actions may impact not only the immediate reward, but also the next situation, and, through that, all subsequent rewards. Those attributes, which are inherently closed loop, lack of direct guidance on what actions to take and where and how the effects of actions play out over extended periods of time, are the three most characteristic features of reinforcement learning problems [129].

One of the challenges of reinforcement learning is the trade-off between exploration and exploitation. To score high rewards, the agent must utilise its knowledge about actions selected in the past that are known to be effective in producing rewards. However, to gain this knowledge, the agent has to try those actions out for the first time. The agent has to exploit his knowledge and at the same time further explore to search for better strategies. Very often, such trade-offs will have to take different forms even within a single episode -

the agent would have to first precisely exploit its knowledge to get to a given place of the environment and from there heavily explore uncharted territory.

Another feature of reinforcement learning is the lack of the assumption that there is one and only correct action, but rather more and less optimal ones. Here, the reward signal plays the role of an evaluation metric, providing much richer and more natural information for the training process. This stands in contrast with most supervised learning methods, where training requires a definition of the correct answer in a given situation- a label.

Another key characteristic of reinforcement learning is to take an approach that considers the problem of a goal-directed agent interacting with an uncertain environment as a whole. All RL agents have defined goals, can sense characteristics of their environments, and can act to influence them. It is also very often assumed that agents have to operate despite serious uncertainty about the environment. When the perception aspect of reinforcement learning involves supervised learning, it is done for a specific reason directly related to the required capabilities in planning. When dealing with the planning part of reinforcement learning, the agent's performance, as an outcome of exact action realisation in response to the uncertain observation of the environment, is addressed directly. Approaching the problem in an end-to-end manner does not necessarily have to mean being responsible for all aspects of processing in the whole robot or organism. We may easily identify successful examples of the use of RL agent as part of a larger system, being responsible for a specific task, where the rest of the system, together with the external world, are treated as an environment.

Reinforcement learning, next to supervised learning and unsupervised learning, states the third main subfield of machine learning research. It may be as well said that its extension of a supervised learning, for which the reference data are constructed based on reward signal evaluation.

2.1.2 Elements of Reinforcement Learning

In a RL system, one can identify its main subelements, such as an *environment*, along with an optional *environment model*, an *agent* with its *policy* and finally *reward function* with its derivative, a *value function*. These elements can be found in the agent-environment interaction scheme presented in Figure 2.1.

The environment encapsulates the external world with which agents will interact. It is often represented as the dynamics behind moving from one state to the other, taking into consideration the action selected by the agent. The environment as well defines how its internal state is represented to an agent as an observation and how the received actions are encoded and realised. In most applications, the agent does not have direct knowledge of the environment, except for the experiences resulting from interaction with it. The environment may also be considered as a definition of the problem to be solved by an agent. In applications of reinforcement learning to real-world problems, the environment should not be treated as something given but as an element to design as well.

In some reinforcement learning systems, the environment model is introduced. Its job is to mimic the behaviour of the real environment and made judgments about how the environment will behave or respond to the agent's actions. One of the ways in which this modelling may be done is to predict the next state and the next reward, given the current state and action. The primary use of this modelling is *planning* task, which is to derive any kind of strategy on action selection taking into account possible future states and outcomes

before they occur in reality. The class of reinforcement learning methods that use the environment model are called *model-based* ones, which puts them in contrast to *model-free* methods, which are trained by trial and error [129].

The entity that interacts with the environment is called the agent. The way in which agents behave at a given moment is defined by its policy, which may be defined as the mapping between a given environment state and the action that should be taken in that state. Learning the policy is a main task of reinforcement learning, and having it is sufficient to define the behaviour. Policies may be defined as deterministic or stochastic, where the probability of each action is defined.

The metric defining how well the agent is doing is defined as a reward signal. In every round of the agent's interaction, the environment sends to the agent a scalar value called a reward. The agent's sole objective is to maximise the total reward that it received in the long run [129]. The reward defines what are the good and bad events for the agent. Based on that, an agent should modify its policy in such a way that the resulting actions will lead to a high reward, which is associated with positive events or desired behaviour.

As the reward signal says what is good at present, the measure of goodness in the long run is defined as a value function. The value is the cumulative reward that the agent, according to its policy, should expect to get in the future starting from the current state. The primary motivation for defining this quantity is encapsulating what strategy would be good in the long run, not necessarily in a short time horizon. For example, a given state may result in low immediate reward, but it may lead to a series of high-reward states in the future. By this, we may say that as rewards define how good it is to be in a given state in isolation, the value function corresponds to the farsighted judgement of how good agents are doing by being in a given state and acting in a manner specified by the policy. The value function should be treated as a derivative of the rewards experienced with the current policy. The value function is also subject to parameterisation, called discounting, which regulates the trade-off between immediate and future rewards. Still, the primary objective of the agent is the accumulation of high rewards, but a good estimate of the value function should indicate how the agent should act to receive high rewards consistently for long periods or how to get to a high-payoff region of states. The efficient methods of estimating the value function arguably play the most important role in almost all reinforcement learning algorithms [129].

2.2 Reinforcement Learning Methods

In this part, an overview of the methods used to define and solve reinforcement learning problems will be presented. It is important to note that there is no accurate and all-encompassing taxonomy of reinforcement learning methods, as modern algorithms tend to be very modular and often benefit from many different theories. Below, basic solution classes are presented with examples of algorithms that may be associated with those.

2.2.1 Multi-Armed Bandits

One of the distinctive features of reinforcement learning from other types of learning is the process of evaluating actions taken instead of giving instructions about the correct ones. This feature creates the need for

active exploration in the environment to gather knowledge about specific actions evaluations. The necessity for gathering such knowledge leads to an exploration-exploitation trade-off, where agents have to balance between exploitation of their current knowledge and search for even better actions, which may be not optimal in the short term, but will pay off in the long term.

One toy reinforcement learning example is the multi-armed bandit problem. In this case, the agent is repeatedly faced with the choice of a given number of actions. After selecting one, a numerical reward is provided, and the agent's objective is to maximise the total score in a given time horizon. This formulation is analogous to a slot machine (a "one-arm bandit"), which, instead of one lever, has multiple one. Each of the actions, corresponding to each lever, has an associated expected or mean reward, assuming that this action will be selected. This number is often referred to as the value of that action. By denoting the selected action at time t as a_t and receiving the corresponding reward signal as R_t, the value of any arbitrary action, $q_*(a)$, is the expected reward assuming that a is selected:

$$q_*(a) \doteq \mathbb{E}[R_t|A_t = a] \tag{2.1}$$

Having access to the true value of each action $q_*(a)$, an optimal policy could be simply defined as selecting the action with the highest value. This assumption, however, is not useful in practise, as access to the true value function is not granted in most cases. Due to that, the goal is to estimate the value function, which would be as close as possible to $q_*(a)$, and to establish the policy on that estimation. The selection of the action with the highest value is referred to as the *greedy policy* while action in this manner is the *exploitation* of current knowledge. The process of selecting any other action is known as *exploration*, as it enables us to improve our estimates of the corresponding value function. As exploitation is the right strategy in the short term, in the long term, exploration may provide new knowledge, which could result in greater total reward. The appropriate exploitation-exploration balance depends in a complex manner on the accuracy of the current estimates, the uncertainty, and the remaining time until the end of the interaction to collect the reward.

Methods concerning the estimation of the values of actions and using that estimation for action selection are called action value methods. One of the most straightforward ways to do so is to average already received rewards, which is called the *sample-average* method. To collect such data, the policy is defined according to which actions will be selected. As mentioned earlier, greedy policy will exploit our current knowledge, however, it will not provide any new information. A simple extension of that is called the *epsilon-greedy policy*, where the agent behaves greedily most of the time but with given probability ϵ, it randomly selects an action from the action space. This provides asymptotic guarantees of convergence to the optimal value $q_*(a)$. To account for computational efficiency, estimates could be acquired by incremental implementation. The update rule is defined as

$$\hat{Q}_{n+1} \doteq \hat{Q}_n + \frac{1}{n}[R_n - \hat{Q}_n] \tag{2.2}$$

is a common form encountered in reinforcement learning research. The difference term $R_n - \hat{Q}_n$ expresses the *estimation error*, where R_n is the *target* and $\hat{Q}_n$ current estimate.

In the case of a non-stationary problem, in which the probability of rewards and their values itself for specific actions may change over time, it makes more sense to weigh rewards acquired recently more than the older ones. One of the most common ways to account for this is to substitute the weighting parameter $\frac{1}{n}$ in the equation 2.2 with a constant parameter $\alpha \in [0, +\infty)$. This results in $\hat{Q}_{n+1}$ being a weighted average of all past rewards and an initial estimate $\hat{Q}_1$:

$$\begin{aligned}\hat{Q}_{n+1} &\doteq \hat{Q}_n + \alpha[R_n - \hat{Q}_n] \\ &= \alpha R_n + (1-\alpha)\hat{Q}_n \\ &= \alpha R_n + (1-\alpha)[\alpha R_{n-1} + (1-\alpha)\hat{Q}_{n-1}] \\ &= ... \\ &= (1-\alpha)^n \hat{Q}_1 + \sum_{i=1} n\alpha(1-\alpha)^{n-i} R_i.\end{aligned} \tag{2.3}$$

There are numerous possible extensions of basic action-value function methods. Optimistic Initial Value [128] initialises the estimates of values to high values, which would not even be encountered. This methodology improves exploration in the initial stages of learning, since actions that have not been tested yet have greater value than those already sampled. The Upper-Confidence-Bound [6] action selection mechanism, along with building up an action value estimate $\hat{Q}$ measures both its uncertainty and its variance. Based on that actions with lower certainty with respect to their value are promoted to be selected more often.

2.2.2 Markov Decision Process and its Extensions

Reinforcement learning problems are most often formally modelled as Markov Decision Process (MDP), or its derivatives. As in multi-armed bandits, this formulation also involves evaluative feedback, but in addition provides the context in which actions should be selected (different actions would be suitable in a different state). This makes MDPs a formalisation of sequential decision-making, where current actions have an impact on immediate rewards and following states and rewards. Because of that, MDPs introduce the concept of delayed reward, which means that the feedback on our choice of action may not be immediate and there is a need for a trade-off between immediate and delayed rewards.

Compliant with the elements introduced in Section 2.1.2, MDP is framing the problem of learning from interaction. The agent (with its policy) is a decision maker and is being trained along the process. The entity with which it interacts is the environment, which encapsulates everything outside the agent. As the MDP formulation is flexible and abstract, the boundary between the agent and the environment does not have to represent the physical boundary between the robot or animal and the environment. In practise, this boundary is often designed within the actor itself. For example, the actuators of a given robot are assumed to be part of the environment, while the agent actions represent some high-level control of those.

The definition of what we expect from the agent should be communicated to the system in the form of a reward function. As an example, the straightforward way to design a reward for a chess game is to assign a +1 reward for winning, -1 for losing, and 0 for a draw. When designing the reward signal, it is crucial to define precisely what the agent should do. The reward signal should take into account any indication of how

the agent should achieve its goals. If it would impart the notion of subgoals (like gaining space in chess), it is quite common that during training agents will "game" the system and find a way to achieve those subgoals at the cost of failing to get to the final goal.

The agent's goal is to maximise the cumulative rewards it receives, which is called *expected return*. In the simplest case, this can be defined as the sum of rewards received after selected timestamp t, as

$$G_t \doteq R_{t+1} + R_{t+2} + R_{t+3} + ... + R_T, \tag{2.4}$$

where T is the final timestamp. This formulation is useful in cases where there is a natural notion of the end of an episode, where the interaction should stop. The end of the interaction occurs in the so-called *terminal states*, which can be associated with both positive outcomes, such as winning the game or arriving at the destination, and negative ones, such as losing the game. After the terminal state, the environment is expected to be reset to a given initial, starting position, which does not depend on the previous terminal state. Settings that can be naturally broken down into subsequences, called *episodes*, and the setup referred to as episodic tasks.

In many cases, however, the interaction of agent and environment cannot be naturally broken into episodes and is better characterised as a continual process without a specific end. Examples of such cases could be ongoing control processes or cases in which an agent has a long life span. These settings are termed *continuation tasks*. In such cases, the definition of reward provided in Equation (2.4) is no longer useful, as it may result in infinite reward due to an unbounded time horizon. An additional element required in such a setting is called the *discount*. The discounted return at each time t can be defined as

$$G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ... = \sum_{k=t}^{\infty} \gamma^k R_{t+k+1}, \tag{2.5}$$

with *discount rate* parameter, $\gamma \in [0, 1]$.

With such a formulation, the present value of future rewards is assessed using a discount rate. This parameter is used to stabilise the learning and indicate how much an agent should care about the future, assuming some uncertainty of it.

From a formal perspective, the Markov Decision Process is defined by a tuple $(\mathcal{S}, \mathcal{A}, p, r, \gamma)$. $\mathcal{S}$ and $\mathcal{A}$ represent state and action spaces, respectively, followed by $p : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$, which denotes the transition probability from any states $s \in \mathcal{S}$ and given action $a \in \mathcal{A}$ to any other state $s' \in \mathcal{S}$. In case of the Finite Markov Decision Process, the set of states, actions, and rewards has a finite number of elements. The $r : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathcal{R} \subset \mathbb{R}$ represents the reward function that determines the immediate reward received by the agent while performing the transition described above. The gamma parameter $\gamma \in [0; 1]$ is the discount factor responsible for modelling the trade-off between current and future rewards. To comply with the Markov property, the future state (s_{t+1}) is conditionally independent of the past ($s_{0:t-1}$, $a_{0:t-1}$) given the present (s_t, a_t), or equivalently, the present state s_t and the action a_t contain all the information that affects the dynamics of environments p and the reward signal r while transiting to s_{t+1}.

The agent's interaction with the environment may be described in the following manner. In each time step t, based on the state of the system $s \in S$, the agent takes an action $a \in \mathcal{A}$. Subsequently, the environment transfers to the next state $s' \in \mathcal{S}$, according to the probability of state transition $p(\cdot|s, a)$, which is most often

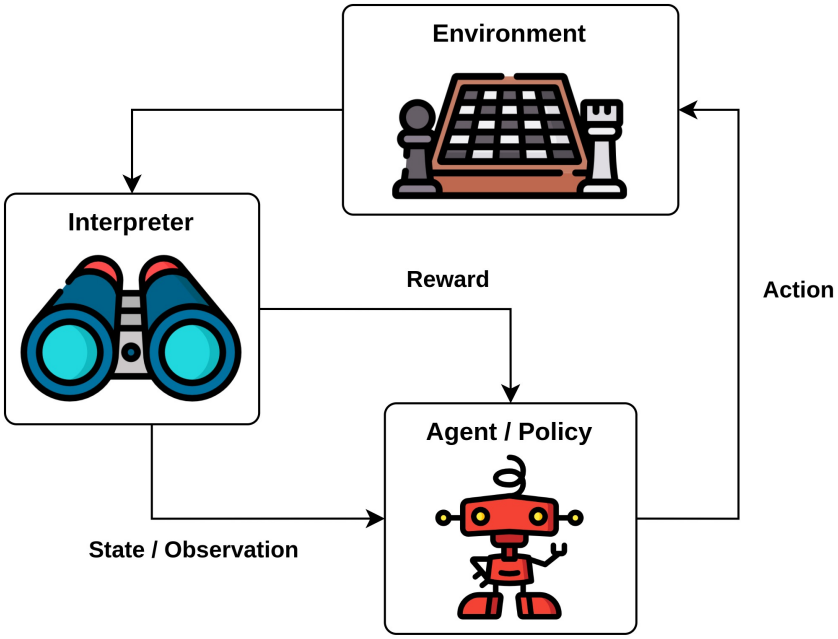


Figure 2.1: Agent interacting with environment scheme.

unknown, while providing the agent with reward $r \in \mathcal{R}$. The goal of an agent is to find a policy $\pi : \mathcal{S} \to \mathcal{A}$, so that the generated action distribution $a \sim \pi(\cdot|s)$ will maximise the expected return

$$G \doteq \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R_t | a_t \sim \pi(\cdot|s_t), s_{t+1} \sim p(\cdot|s_t, a_t), s_0\right]. \tag{2.6}$$

In the following, the action-value function q_π and the value function v_π for a given policy π can be defined. The action value function is an extension of the action value function presented in the multi-armed bandits case (see Section 2.2.1), which instead of estimating how good it is to take a given action ($q : \mathcal{A} \to \mathcal{R}$), has to account also for the context in which this action has been selected ($q : \mathcal{S} \times \mathcal{A} \to \mathcal{R}$). The value function ($v_\pi : \mathcal{S} \to \mathcal{R}$) represents the potential of a given state, without considering the concrete action selected from that state. They are, for any state $s \in \mathcal{S}$ and action $a \in \mathcal{A}$, respectively, defined as

$$q_\pi(s, a) \doteq \mathbb{E}_\pi\left[G_t | S_t = s, A_t = a\right] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | S_t = s, A_t = a)\right], \tag{2.7}$$

$$v_\pi(s) \doteq \mathbb{E}_\pi\left[G_t | S_t = s\right] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | S_t = s)\right]. \tag{2.8}$$

Both of these functions represent the discounted accumulated future reward starting from $S_t, A_t = (s, a)$ and $S_t = s$ respectively for given policy π, however, with such a difference, that for the Q-function the first action may be chosen arbitrarily while in the value function all actions are sampled from current policy π.

The concept of MDP can also be extended to cases in which the agent does not have full access to the environmental state. In such a setting, called the Partially Observable Markov Decision Process (POMDP) [80], the agent has access only to the *observation*, which is the subset of state by which the environment's dynamics and rewards are governed. Although this characteristic of the environment increases the

difficulty of the problem and there are specific methods that try to tackle this problem directly [50, 87, 117], most of the standard methods used in reinforcement learning can be applied to such cases with success.

2.2.3 Dynamic Programming

The Dynamic Programing (DP) methods, firstly introduced in [13], are the algorithms that can be used to learn optimal policies having access to a perfect model of the environment, which is represented in a form of MDP. Because of this requirement and their great computational expense, those methods have limited utility for bigger problems. The basic concepts of them have been, however, used as a base for other methods, which often attempt to achieve the same results as dynamic programming methods, but with fewer computations and with limited knowledge about the environment.

The core idea behind dynamic programming is to solve complex problems by breaking them into sub-problems, solving them, and then combining solutions to those to solve the original task. In the case of the Markov Decision Process, usage of the Bellman equation brings recursive decomposition to the problem, as the value function is capable of storing and reusing knowledge, which allows using DP concepts.

Dynamic programming methods usually assume that the environment is finite MDP. Although concepts of DP can be applied to continuous state and actions spaces, acquiring exact solutions is only possible in special cases. Common practise is to discretise the state and action spaces and apply finite MDP dynamic programming algorithms. DP methods as well assume that they have access to the environment's dynamics function p and the reward function r.

The standard DP algorithm applied to RL problems is composed of a few elements and stages. The first step is to evaluate the policy. At this stage, the evaluation of the given policy π is done by estimating the corresponding state-value function v_π.

At each iteration $k+1$, for each state $s \in \mathcal{S}$, the estimation of the state value function $v_{k+1}(s)$ is updated based on the known dynamics of the environment p and the previous estimate of $v_k(s')$, where s' is a successor state of s. This update is based on the application of the Bellman equation

$$v_{k+1}(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \sum_{s' \in \mathcal{S}, r \in \mathcal{R}} p(s', r|s, a) \left(r + \gamma v_k(s')\right). \tag{2.9}$$

Evaluating policy π gives us a way to guide our policy improvement process. Having current policy π, the step of policy improvement is based on checking whether in a given state s, it is better to select action $a \neq \pi(s)$ and then follow π or follow π all the time. If the former is true, then the improved policy π' would be exactly the same as the old policy π, instead of the action suggested in the state s. The new suggested action, a can be determined by acting greedily with respect to v_π, which is

$$a = \underset{a}{\operatorname{argmax}} \left(\sum_{s' \in \mathcal{S}, r \in \mathcal{R}} p(s'|s, a) \left(r + \gamma v_k(s')\right) \right). \tag{2.10}$$

Having the two elements in place, the *policy iteration* can be defined, which alternately evaluates the policy and improves it. Because finite MDP has a finite number of states, this process converges to an optimal value and a policy function in a finite number of steps.

To address one of the drawbacks of policy iteration, which is the necessity to run a policy evaluation at each iteration, the policy evaluation can be truncated in several ways. A special case is stopping the policy evaluation phase just after a single update of each state, which results in an algorithm called *value iteration*. In that case, a simple update rule can be defined as

$$v_{k+1}(s) \doteq \max_a \sum_{s',r} p(s', r|s, a) \left(r_s^a + \gamma v_k(s')\right) \tag{2.11}$$

Another important advancement to DP methods is its asynchronous version, which instead of running operations over the entire state set, can update them in any arbitrary order. This, as well, brings flexibility to a number of applied updates to specific states with reference to others.

An extensive review of dynamic programming methods can be found in many texts, such as the most recent work by Bertsekas [15, 16].

2.2.4 Monte Carlo Methods

Monte Carlo (MC) methods have been referenced from the 1940s when physicists working on the atom bomb were trying to understand its complex physical phenomena in a form of a game of chance. Coverage of those methods in the problem, as well as in a more general sense, can be found in [54, 106].

MC methods learn directly from episodes of experiences and are model-free, thus they do not need for a dynamics model of the environment. MC methods learn from complete episodes without bootstrapping and use the most straightforward idea of estimating a value as the mean of returns. One caveat of MC methods is that they can only be applied to episodic tasks, thus all episodes must terminate. Unlike the DP methods, they rely on sampling actions rather than evaluating all actions at once. Because of that, it is useful to estimate action values instead of state values. As a single state may occur multiple times during the episode, two distinct strategies have been developed to address that. In first-visit MC, the value of state s is estimated by averaging all returns after the first visit to s, while in every-visit MC the returns are averaged after all visits to s [123].

The Monte Carlo control algorithm follows a generalised policy iteration scheme [129], in which we alternately run an evaluation to estimate the action value function and policy improvement. The evaluation phase uses the empirical mean return, which is the total discounted reward, instead of the expected return, as in DP methods. The update rule in each state s_t and return G_t for value function estimation may be represented as

$$N(s_t) \leftarrow N(s_t) + 1, \tag{2.12}$$

$$\hat{V}(s_t) \leftarrow \hat{V}(s_t) + \frac{1}{N(s_t)} \left[G_t - \hat{V}(s_t)\right]. \tag{2.13}$$

Later, policy improvement relies on building a greedy policy with respect to current estimation of action value

$$\pi = \operatorname*{argmax}_a q(s, a). \tag{2.14}$$

To build a reliable estimate of $q(s, a)$, we have to ensure that we will continue exploration. Here, the division is made into *on-policy* methods, which use a single policy for evaluation and exploration, and *off-policy* methods, where behaviour policy is used to explore and update the target policy, which is supposed to behave optimally [129].

2.2.5 Temporal-Difference Learning

Temporal Difference (TD) learning is the combination of DP and MC ideas. TD methods, same as the MC ones, learn directly from episodes of experiences and do not require knowledge of environment dynamics. In contrast to MC methods and DP methods, however, they learn from incomplete episodes by bootstrapping, which is updating one estimate based on the other. Without the necessity of observing complete episodes, in addition to episodic tasks, TD methods can also be applied to continuing tasks (see Section 2.2.2 for an explanation).

TD methods, such as DP and MDP ones, with minor differences, follow the generalised policy iteration scheme [129]. Most significant differences are present in the execution of policy evaluation, or prediction problems, where the value function v_π is estimated. MDP methods have to wait until the end of the episode, where the return G_t at time t is known, and use that value as a target in value estimation. In contrast, TD methods have to wait only for the next time stamp $t + 1$ and form a target from its current estimate of the value function. The simplest update rule for the TD methods can be represented as

$$\hat{V}(s_t) \leftarrow \hat{V}(s_t) + \alpha \left[R_{t+1} + \gamma \hat{V}(s_{t+1}) - \hat{V}(s_t)) \right], \tag{2.15}$$

where $R_{t+1} + \gamma \hat{V}(s_{t+1})$ is the target, and α parameter is step-size parameter. The algorithm using this kind of update rule is called TD(0) or *one-step* TD, introduced in [131].

TD, DP and MC methods can be combined and blended in multiple ways. An example of the combination of TD and MC ideas is *n-step bootstrapping*, which in short entails spanning the bootstrapping process across multiple timestamps. The approach is seamlessly unified with them is the $TD(\lambda)$ algorithm, which introduced the idea of *eligibility traces* and adds the mechanism for weighting for different n-step horizons. The foundation for these ideas has been introduced [151] while important details regarding practical implementations have been studied in [27, 116, 143].

2.2.6 Deep Reinforcement Learning

Deep Reinforcement Learning (DRL) is one of the most active areas of research in reinforcement learning. DRL methods utilise artificial neural networks (ANN) as a nonlinear function approximation. In most cases, the methods use Deep Neural Network (DNN), which is composed of many hidden layers, to allow automatic feature extraction. Those networks are most often trained with the use of the back-propagation algorithm, which relies on Stochastic Gradient Decent (SGD) or its variations.

Deep neural networks are used to approximate different entities in the reinforcement learning process. The first application of the deep learning model which successfully learns control policies by approximating the action value function based on raw high-dimensional sensory input [76, 77]. The successful use of ANN as a policy approximation, sometimes in parallel with the value function approximation (see Section 2.2.9 for

actor-critic methods), can be found in [44, 75, 114, 115]. In [112, 160], in addition to the approximation of the policy and value, DNNs were also used for the semantic representation of the state and the approximation of the dynamics of the simulation environment of the representation.

2.2.7 Value-Based Methods

Value-based methods focus on finding a good estimate of the state action value function $q_\pi(s, a)$ or the state value function $v_\pi(s)$. Achieving this, the approximate optimal policy can be defined as taking greedy actions according to q-function estimates.

One of the first and popular value-based algorithms introduced was Q-learning [151] designed for finite MDP, where both the state space $\mathcal{S}$ and the action space $\mathcal{A}$ are discrete. The use of the neural network to approximate Q-function unlocked the possibility of operating in continuing state spaces and has resulted in the surpassing of humans in some applications [77]. To ensure exploration, the DQN algorithm uses the epsilon-greed policy, where the epsilon parameter has been decaying during training. As DQN is an off-policy algorithm, it reuses past experiences during training by storing them in *experience-replay buffer*, from which they are later randomly sampled in the form of mini-batches in training.

Numerous advancements have been made to the baseline DQN algorithm. To reduce the effect of the $\max$ operation during estimation of value for a future state, [142] decoupled the Q-network used for the selection of bootstrapped targets and actions in the evaluation. In [149], tailored neural network architecture was used, where two streams of processing have been merged within the head layer by a special aggregation process, allowing the representation of the state value $\hat{V}$ and the so-called advantages $\hat{A}$ that should be summarised with the estimates of the value of action $\hat{Q}$. As in original DQN experiences that have been sampled uniformly, [111] introduced the concept of Prioritised Experience Replay, which samples more often transitions from which there is more to learn. The priority of sampling given transition is proportional to the last encountered TD-error. The concept of learning reward distributions instead of expected return has been proposed in [12]. To address the limitations of uncoordinated exploration using epsilon-greed policies, [38] proposed the introduction of the linear noisy layer to combined deterministic and noisy streams, allowing for state-conditional exploration. The above advancements are not mutually exclusive and have been successfully combined in [47]. For goal-based environments with sparse reward, an interesting add-on has been presented in [3]. The Hindsight Experience Replay method adds mocked episodes to replay buffer, where states at which agent actually arrived are marked as goal state. This process introduces positive reinforcement, especially beneficial in situations where the probability of getting to the goal by random policy is low.

2.2.8 Policy-Based Methods

In the case of value-based methods, the main attention has been paid to estimation of the value or action-value function, based on which the policy was defined (like ϵ-greedy). On the contrary, policy-based methods aim to directly parameterize the policy π_θ.

The advantages of policy-based methods include better convergence properties and being more effective in high-dimensional and continuous action spaces. Additionally, the policy-based algorithms may learn

stochastic policies which may be beneficial in some environments. On the other hand, those methods tend to stick to local optimum and evaluation of a policy is typically inefficient and associated with a high variance.

The simplest implementation of policy gradient methods is the REINFORCE algorithm, also known as the Monte Carlo Policy Gradient [155]. The algorithm works in a series of gradient ascents based on the estimated return from the sampled experience trajectory. The Vanilla Policy Gradient method, in which basic ideas have been drawn and explained in [113, 130], resembles the REINFORCE algorithm, however, it performs gradient ascents once over multiple episodes by averaging returns, expressed as advantages. The Trust Region Policy Optimisation (TRPO) [114] aims to take the biggest possible step to improve performance while making sure that the new and old policies are not too far away from each other. This distance is expressed in the form of KL-divergence, which may be understood as a distance measure between two probability distributions. The same principle of stepping as far as possible without decreasing performance has been tackled in the Proximal Policy Optimisation (PPO) algorithm [115]. Instead of using the complex second-order method as TRPO, PPO uses the first-order method along with a few tricks to keep the policy close to the previous one. PPO is often implemented in two variants. PPO-Penalty uses KL-Divergence to penalise the objective function instead of making it a hard constraint. PPO-Clip version does not use KL-Divergence but instead uses specialised clipping in the objective function to keep new and old policies close to each other.

2.2.9 Actor-Critic Methods

One of the biggest issues in policy-gradient is the high variability of action probabilities and cumulative reward. This causes high variance (noisy gradients), which may destabilise learning and skew the policy distribution into a non-optimal one. One of the ways to reduce this negative effect is by introducing a baseline, which makes the gradient calculation less prone to variability mentioned above.

Actor-critic methods try to address this issue by approximating a better baseline and are a hybrid of policy-based and value-based methods. In this setup, the *actor* plays the role of policy and maps the current state to action. The *critic* aims at evaluating the agent's actions, in most cases by approximating the value function of the current policy.

Policy-based methods (Section 2.2.8), such as TRPO [114] or PPO [115], can also be extended to approximate the value function and improve the estimation of advantages. In [75], Asynchronous Advantage Actor Critic (A3C) was introduced, which used a mix of n-step returns. The training has been scaled up to multiple workers which gather experience in parallel, and based on them updates the global network in an asynchronous manner. The synchronous version of this algorithm, A2C, works in iterations and updates the policy and value after collecting the experience batch from all actors. An combination of on-policy and off-policy updates has been proposed in ACER [150], utilising the concept of the retrace algorithm [85]. In [66], the Deep Deterministic Policy Gradient (DDPG) method has been introduced, which combined ideas from DQN and the Deterministic Policy Gradient and was successfully applied to continue problems off-policy. The policy gradient is assumed to be equal to the estimate gradient of the value function. To address exploration, additive and correlated noise in the form of the Ornstein-Uhlenbeck process [139] has been added to deterministic actions. The Twin Delayed DDPG [41] improves the baseline DDPG with clipped double Q

learning, less frequent policy updates than the Q function, and smoothing of the target policy. As an extension that aims to improve the efficiency of DDPG, D4PG [9] introduced distributional learning [12], n-step returns as in A3C and multiple parallel actors gathering experience to prioritised experience replay. Soft Actor-Critic (SAC) [44, 45], in contrast to DDPG, employs a stochastic policy, and with the use of entropy regularisation done along value estimation, automated the exploration and made it state dependent.

2.2.10 Multi-Agent Reinforcement Learning

Driving, in most cases, involves dealing with other road users. Aligning our actions with the actions of others is presumably the fundamental problem of controlling a car and one of the main challenges of automated driving.

In Reinforcement Learning, the multi-agent aspect of the environment introduces inherent complexity to the problem of finding an optimal policy for an agent. This is true, especially in settings in which other agents also undergo training processes. This makes the environment (from the perspective of a single agent) nonstationary during the learning phase, which is a straightforward violation of Markov's property. Because of that, the literature introduced the Markov Game (see e.g. [145]), which is the extension of the Markov Decision Process to a multi-agent use case.

Looking at general objectives and models of agents' interaction, environments may be divided into subcategories representing underlying multi-agent problems. In a *competitive* setting, the gain of each agent goes along with the loss of another agent or agents. This formulation goes along with the definition of the zero-sum game, in which experience gain and loss are equivalent. Those settings are easier to solve, as the rules of the game provide a much easier indication of better and worse policies due to the clear definition of winner and loser. Examples of such environments could be predator-pray settings or the game of chess [122].

The *collaborative* model of agents' interactions represents situations in which all agents in the environment act together to accomplish a single, common task and maximise a joint objective. This often calls for strict cooperation between agents, as the task cannot be accomplished by a single agent alone, while performance is always measured from a team's perspective. Examples of such environments could be warehouse environments with robots delivering goods [25] or coordinating a railway traffic infrastructure [79].

Most real-world problems represent the *cooperative* model, which is a middle ground between competitive and collaborative dynamics. In this setting, agents aim to work with others to achieve their own goals, parts of which might also be shared. These involve settings present in all team games, where agents need to cooperate with team members to successfully compete with the enemy team (such as football [60] or the real-time strategy game Starcraft II [108]). Another option, however, valid for driving task, is the mixed setting in which agents need to cooperate and compete with other agents at the same time. In such a situation, the agent must care for his egocentric objectives and goals, but also consider the objectives and goals of other road users. Another example of such an environment could be a Diplomacy game, where agents compete, cooperate, negotiate, and communicate with the aim of owning the majority of the territory on the map [59].

Numerous reinforcement learning methods have been already presented for a broad variety of multi-agent problems. Optimising agent's policies with the use of machine learning using communication chan-

nels between agents has been investigated in [83, 127]. In [91], an extensively scaled version of the PPO algorithm has been used to play a multi-character strategy game, called Dota 2. OpenAI Five introduced a hyperparameter called team spirit, which is responsible for weighting individual characters' rewards versus the average rewards of a team member. Agents were trained by self-play with a pool of old versions of themselves, assuring always appropriate difficulty level of a game. No additional mechanisms, except for the massive training scale, have been used to address the multi-agent aspects of an opponent. A trained set of five neural networks presented superhuman abilities in competition with professional Dota 2 players, showing cooperation skills among team members. The same training approach has been used in [7], where two teams competed in the hide-and-seek game. The environment has been designed as a set of scenarios in which different objects could be used as cover by hiders and by seekers to reach the hiders team. The main phenomena of training were the emergence of a series of strategies and counter-strategies along with training by both teams, creating complex tasks curriculum from relatively simple game dynamics. A method utilising the actor-critic approach, in which the critic has access to all agent's observations during training, has been proposed in [69]. As the actor network is consuming only local observations data, the method did not require any communication between agents. The method has proven to work well for both cooperative and competitive interaction models. An interesting observation has been made in [20], where the issue of cooperation with the suboptimal (or just not aligned) human agent was highlighted. The authors showed that in cooperative environments, where agents trained through self-play are paired with human agents, performance is significantly worse. The authors also introduced the concept of adapting to human gameplay, which results in better coordination with humans and more robust policies.

2.3 Autonomous Driving

2.3.1 History of Autonomous Driving

The work on self-driving cars and AD has been carried out since 1939, with the first promising experiments taking place in the 1950s and since then research has continued. The first truly self-driving and autonomous vehicles was created by Carnegie Mellon University's Navlab and ALV projects in 1984 [55], followed by the Eureka PROMETHEUS Project, which was a joint effort of Mercedes Benz and Bundeswehr University in Munich in 1987. Since those two events, multiple automotive companies and research organisations have worked on autonomous driving technology, often resulting in functional prototypes. One of the important milestones in AV development was the DARPA challenges. The first two events concerned navigation in a desert environment [132, 136, 140], while the third, called the DARPA Urban Challenge, was held in a closed urban-like area with a focus on multi-vehicle coordination [40, 81, 141]. These events gathered many people who today are leading the development of autonomous driving technology. In 2010, the major automotive companies began to invest heavily in AD research and development. Companies such as Waymo (a subsidiary of Google), Tesla, and Uber along with traditional automakers such as Mercedes-Benz, BMW, and GM became actively involved in autonomous driving projects. Highlighting some of the important milestones, in 2013 Mercedes-Benz drove the Bertha Benz memorial route autonomously [171], while in

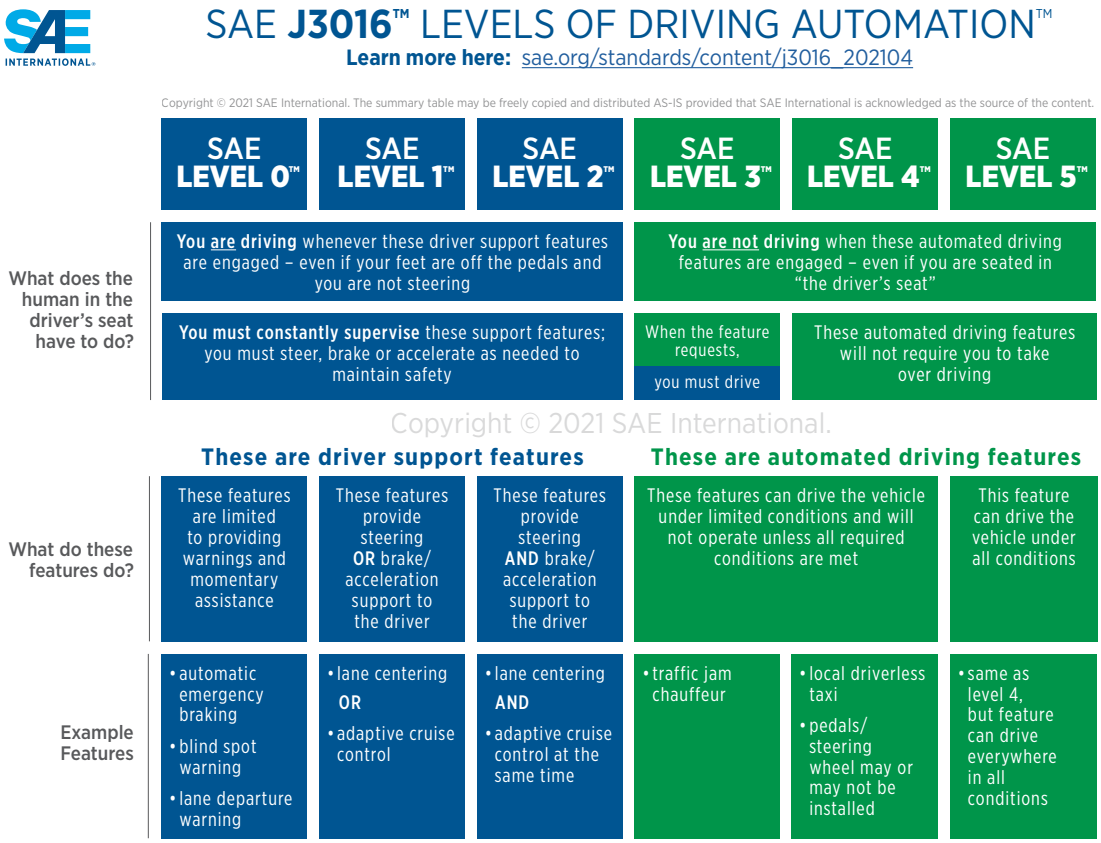


Figure 2.2: Breakdown of SAE levels of driving automation [1]

2015 the automotive supplier Delphi Automotive PLC (not Aptiv) completed the longest automated drive in North America, travelling coast to coast from San Fransisco to New York [31].

One of the key drivers of autonomous driving progress has been advances in sensors and perception. Sensors such as LIDAR, radar, and cameras, currently are a crucial part of the AD stack, providing a 360-degree perception of the car's surroundings. To accompany that, progress made in the field of machine learning and artificial intelligence allowed for processing and interpreting the raw sensor data, enabling automatic detection of road structure, objects on the road, and other key elements important from the AD perspective.

As autonomous driving progressed, governments and regulatory bodies began to address the safety and legal aspects of it. In some countries, including the United States, China, and Germany, guidelines and bills specific to autonomous vehicles have been introduced. Additionally, international organisations such as Society of Automotive Engineers (SAE) defined the standardisation and taxonomy of autonomy levels to classify the capabilities and responsibilities of a given self-driving system [1] (see Figure 2.2 for a detailed explanation of each level).

Over the years of development, two distinctive paths toward fully autonomous driving have been established. The first one, favoured by technology companies, is coupled with a newly introduced transporta-

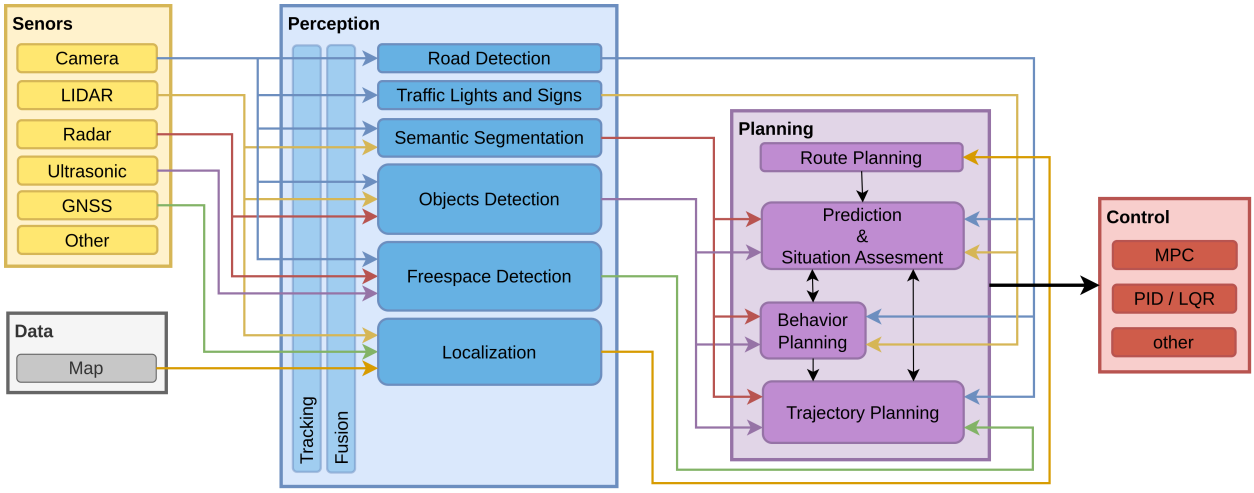


Figure 2.3: Typical autonomous driving system architecture.

tion concept, called Mobility on Demand (MoD), which provides accessible transportation services on an as-needed basis. The heavy dependence on digital platforms, shared mobility, and integration and interoperability of various transport modes should result in efficient use of resources, reduced congestion, and improved user experience. With relaxed costs requirements and smaller scale with better control over the fleet, more advanced approaches might be tested, from the hardware and data perspective. Currently, Alphabet's autonomous driving division, Waymo, has one of the largest autonomous driving fleets, currently reaching around 700 vehicles and providing its ride-hailing service in three locations in the United States. Other technology companies, such as Motional and Cruise, are conducting similar tests in some of the largest cities [29, 84].

In the second, more bottom-up approach, vehicle manufacturers are steadily growing the autonomous capabilities of cars from their offerings, prioritising more cost-efficient and scalable solutions. Development of feature functions is also partly driven by regulatory bodies, including the Euro NCAP organisation, which enforces the existence of some active safety features or assesses vehicle safety based on the performance of those. The current state-of-the-art includes the first successful realisations of the SAE Level 3 systems. In 2020 Honda introduced to Japan the first L3 system, called Sensing Elite, in a limited number of cars available for lease, which allows the driver to not pay attention to the surrounding under certain conditions of traffic jams on the highway [48]. Mercedes introduced a similar L3 highway traffic jam system to Germany in 2022, with plans to spread it to other countries [37]. At the same time, manufacturers are also heavily invested in the development of so-called L2+ systems, which expand the capabilities of the AD system while keeping the driver responsible and in the loop.

2.3.2 Typical System Architecture and Components of AD System

Over the decades, a large number of different architectures have been proposed to tackle the problem of autonomous driving. Without a one-and-only right way to define the system, some components and concepts reappear in almost all proposed solutions.

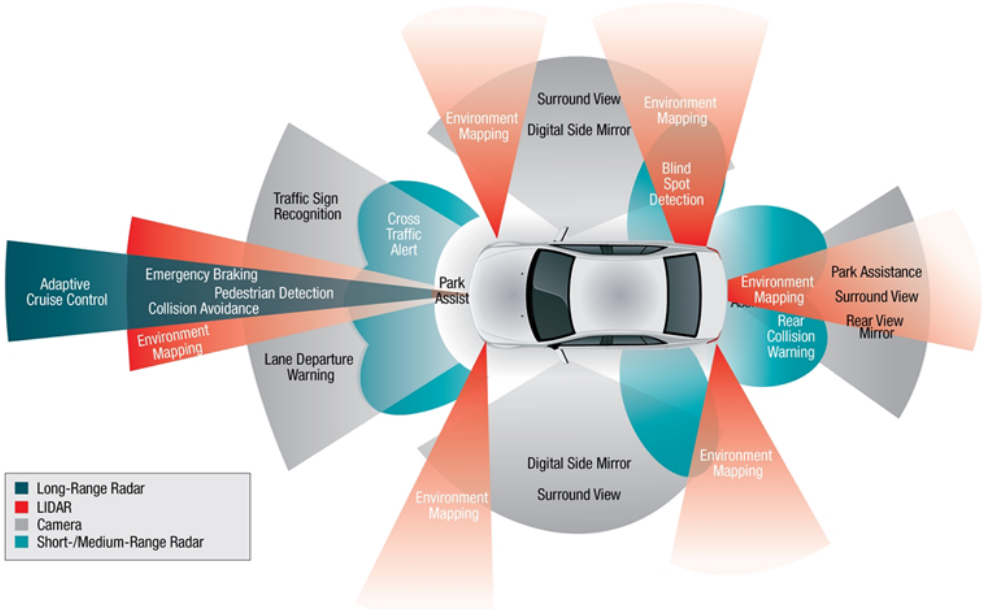


Figure 2.4: Example sensor coverage along with features which its supports [109]

In Figure 2.3, a high-level version of the typical system architecture of the autonomous driving system is presented. First of all, the car must collect information about the surroundings. To do so, numerous different types of sensors are used. Car might also use the data, like a map, saved on-board or accessible online. All this information is used in different perception modules, allowing one to detect different kinds of surrounding elements, presenting them in different forms. The process of perception almost always includes tracking and data fusion between different sources, which first and foremost allows for improvement in detection performance. With a good representation of the environment around the car, planning takes place, including understanding where to drive (route planning), what kind of situation we are in (situation assessment), what other traffic participants will do (prediction) and how to act accordingly (behaviour and trajectory planning). With defined action, most often defined in a form of trajectory or set of reference set points, control blocks are responsible to translate it to actuators input format, which results in ability to regulate movement of the car.

In Sections 2.3.3 to 2.3.8, more details about all these blocks are provided, with the most typical methods used currently in both research and production environments.

2.3.3 Sensors

To realise any ADAS or AD feature, the first and foremost car has to perceive its surroundings and understand the context in which it is in. In this regard, the capabilities of the system are strongly connected with the car's functional capabilities in the end. In almost all cases, except for minor research activities [158], perception is treated as a separate task with separate defined objectives. With a world model created on such a base, the features functions of the given AD system operate. Figure 2.4 presents an example of the sensor coverage of the high-end setup, along with potential feature functions supported by each sensor unit.

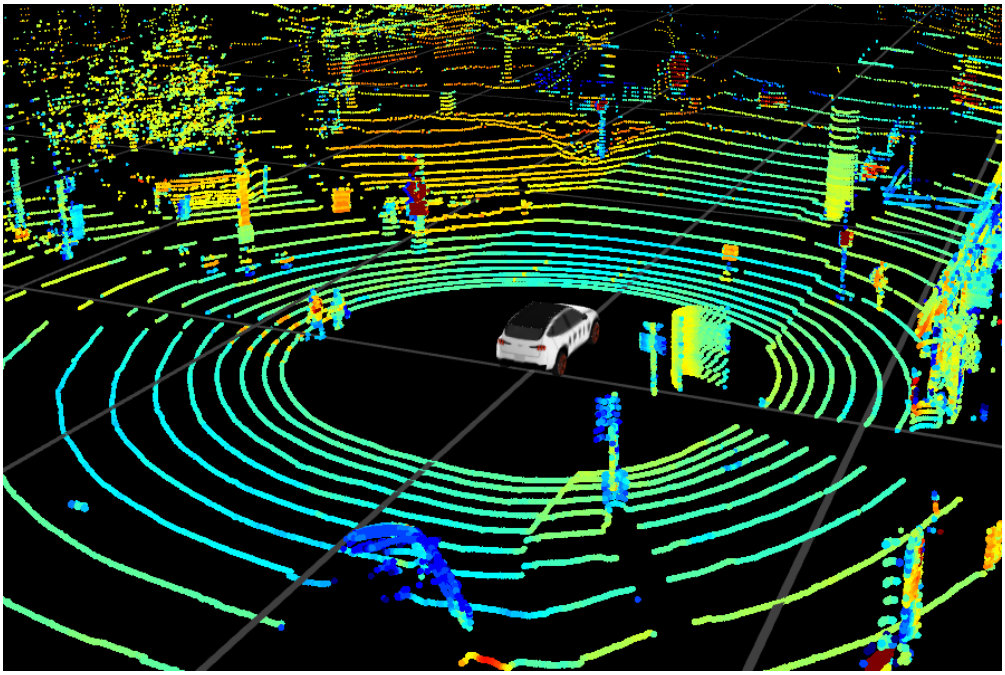


Figure 2.5: Measurement of an urban scene with the use of reference LIDAR sensor mounted on the roof of the test vehicle.

Lidar

The sensor that provides the most direct and one of the richest representations of the environment of the vehicle is the Light Detection and Ranging (LIDAR) sensor. The LIDAR principle of operation consists of determining the ranges by targeting objects or surfaces with a laser beam and measuring the time of flight of that beam to an object and back. LIDAR sensors provide a 3D detection map which can be later used to perceive the world in the form of a semantic segmentation map or entity detection in vector form.

Different types of LIDARs have been proposed. The most distinguishable one is the mechanical type, in which a rotating mirror is used to direct the laser beam in multiple directions. With Micro-Electromechanical System (MEMS) setup, micro mirrors placed directly on a chip are controlled by electromagnetic force. The Flash LIDARs diffuses the laser beam to cover the entire scene at once and then detects the reflected beam using a 2D detector array.

The main advantages of the LIDAR sensor include high precision and resolution, direct 3D position measurement, and a large field of view with a relatively wide range. One of the severe disadvantages is the lack of resilience to weather conditions, especially heavy rain. Additionally, due to the complicated mechanical construction, LIDAR sensors are still considered a costly solution compared to other kinds of sensors, therefore, their application is limited to commercial MoD services and high-end vehicles.

Radar

The other sensor that found its usage in the automotive domain is the radar. This well-known technology relies on sending and receiving electromagnetic waves, which on the way are reflected from obstacles. Through this process, obstacles can be detected and located in the sensor field of view. Additionally, one of

Figure 2.6: The Aptiv's SRR6 sensor, which is most often mounted in the corners of the vehicles [103]

the biggest advantages of automotive radar sensors is the ability to directly measure radial velocity, often called the range rate. An image of a typical radar sensor is presented in Figure 2.6.

In the automotive setting, the most commonly used radars are the Frequency Modulated Continuous Wave (FMCW) ones, with a frequency band in the range between 76 and 81 GHz. There are also multiple types of radar depending on the use case, starting from Short Range Radar (SRR) with a large field of view, high resolution, and relatively short range (up to 70 metres) to Long Range Radar (LRR) with a narrow field of view but a maximum detection range, even up to 250 metres. The next step in radar evolution is the introduction of imaging radar technology, which, except for the standard measurement of range, direction, and relative velocity, also provides vertical information and generally presents significantly better resolution. This results in a 4D view of the world, allowing one to determine the over- and under-drivability of obstacles, better detection of road contours, and building more contextual information about the environment.

The pros of the radar sensor include being more robust to weather conditions, long-range detection, direct measurement of position and relative velocity, and straightforward integration with a relatively small cost per unit. With regard to the cons, the processing of radar signals is rather complex for both angle measurement and object classification.

Camera

Human drivers first and foremost use their vision to perceive the environment around the car and assess the situation. As all the road infrastructure is represented graphically, including traffic signs, traffic lights, and road markings, the use of vision systems is a necessity.

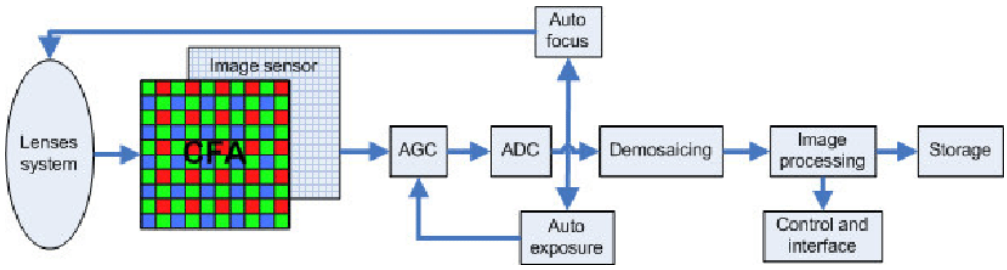


Figure 2.7: Camera hardware system schematics [101]

The typical camera sensor schematics are presented in Figure 2.7. First, the lens system forms and projects the image onto the image sensor, which converts visual data to electrical signals, with Complementary Metal Oxide Semiconductor (CMOS) technology being the most popular solution. Later, after Auto Gain Control (AGC) adjusts the brightness of the image, an analogue-to-digital converter translates the continuous electrical voltage into a discrete form, which heads to an image signal processor, responsible for basic operations such as demosaicing, noise reduction, white balancing, and others. Several control operations, including exposure adjustment and auto focus, are also executed along the way.

Although cameras produce rich visual information, are cost-effective, and provide high resolution, captured raw images can not be directly used in further processing and require extensive and elaborate processing to extract semantic information. This includes depth perception challenges in which perception must rely on visual cues, such as object size, perspective, context, or motion parallax, to locate and classify objects. Fortunately, the rapid development and progress in the fields of machine learning and artificial intelligence enabled the realisation of those tasks in real-time onboard vehicles, immensely increasing the performance of depth estimation and object detection. Nevertheless, vision systems are subject to limited perception range and are vulnerable to adverse weather and lighting conditions.

Initially, camera systems were used mostly in forward vision applications and as an aid to a driver during reversal. Currently, its utilisation has increased significantly. To provide a better field of view in front of the car, some manufacturers have used systems with multiple cameras, each with different focal lengths. Recently, single-camera front-view systems with increased resolution have been put into production. Stereovision cameras are used as well as a front-vision system, however, mono-camera systems do dominate the market. In high-end variants, cameras have been mounted as well near pillars and in the rear, allowing to observe cross traffic and cover intersection scenarios. In parking applications, 360-degree camera setups based on fish-eye cameras allow for representing the closest car surroundings from a birds-eye perspective, which can be used both by the driver directly or by automatic parking feature. Vision systems have also found their place within the vehicle. Driver and cabin monitoring systems are becoming a requirement, allowing the detection of seat occupancy, seatbelt fastening status, or driver state and alertness level. Gesture recognition systems are also used as an alternative to control cars' onboard systems.

Ultrasonic

The ultrasonic sensors are widely used as cost-effective solutions for parking aid, maneuvering assistance, and low-speed emergency braking. They operate by producing the sound wave and listening for the echo

reflected from potential obstacles in the detection cone. The sound wave is created with the use of a piezo-electric transducer, which converts electrical current to mechanical wave, and is later perceived with the use of the same effect. The time between sending and receiving the sound pulse is proportional to the distance to the obstacle. The effective range of detection starts at a few centimetres and ends at several metres. The advantages of ultrasonic sensors are their low cost, allowing for the placement of multiple sensors around the vehicle, and relatively good resolution in close range. Disadvantages include short detection range, limiting the use of such sensors only to low-speed maneuvering, and ambiguity of azimuth of the reflected object.

Others

Other sensor types are also in use in AD and ADAS use cases. The Inertial Measurement Unit (IMU) sensors, consisting of accelerometers and gyroscopes, measure vehicle linear and rotational movements, allowing them to provide more precise information about vehicle position, orientation, and motion. For a similar purpose, wheel speed sensors measure the rotational speed of each wheel, allowing them to detect a potential slippage and estimate the host's vehicle dynamics. To be able to localise, Global Navigation Satellite Systems (GNSS) sensors are used to receive the signals from satellites, allowing them to determine the vehicle's position and velocity. In earlier days of ADAS system development, simple laser sensors, with only a few laser beams, were used in ACC systems for distance measurement to cars in front of the ego vehicle. In premium segments, infrared sensors are used to detect Vulnerable Road Users (VRUs) under night conditions. There is also a wide range of other sensors used in today's vehicles as well, but their impact on ADAS and AD systems are neglectable.

2.3.4 Perception Data Representation

As sensors collect the raw data stream, these data have to be processed to a usable format from a functional perspective. Such processing of the data streams might be done for a single sensor, or by fusing multiple sensors' outputs, potentially coming from different domains. This could also be done at the low level, in the so-called Low-Level Fusion (LLF). The way how processing is done varies from sensor to sensor, however, a trend in using machine learning methodologies is visible in all domains.

Pre-processed data are represented in multiple forms, useful for different kinds of application. Objects, including cars and VRUs, are most often represented in the form of a list of data structures, including object position, dynamic state, class, and other features (such as blinker status for a car).

Road, including information about lane markers, road edges, and barriers, is often directly related to traffic signs and traffic light information, allowing to gather all road and legal-related data in a single place. The shape of the road is most often modelled as polynomials, however other kinds of curve formats, like splines, are also in use. In more advanced systems, where crossroads are also modelled, specific lane segments are interconnected with each other in a graph manner. A good example of road data representation, mainly used for simulation purposes, is the OpenDrive standard [5].

Such representations of objects and road data allow us to easily implement rule-based systems and interconnect the information with each other, defining, for example, lane assignment for a car or providing the status of a correct traffic light for an ego vehicle.

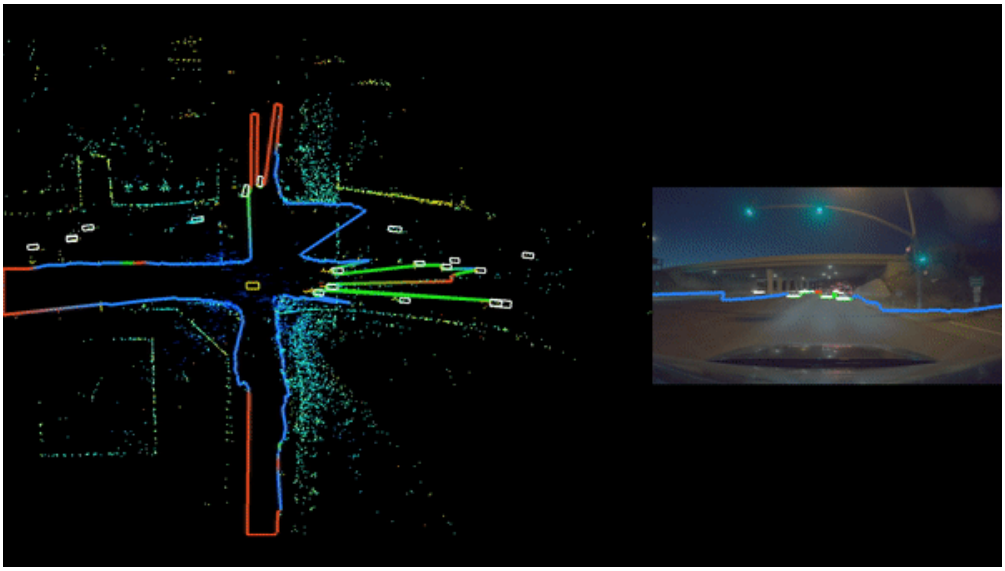


Figure 2.8: Example freespace representation [33]

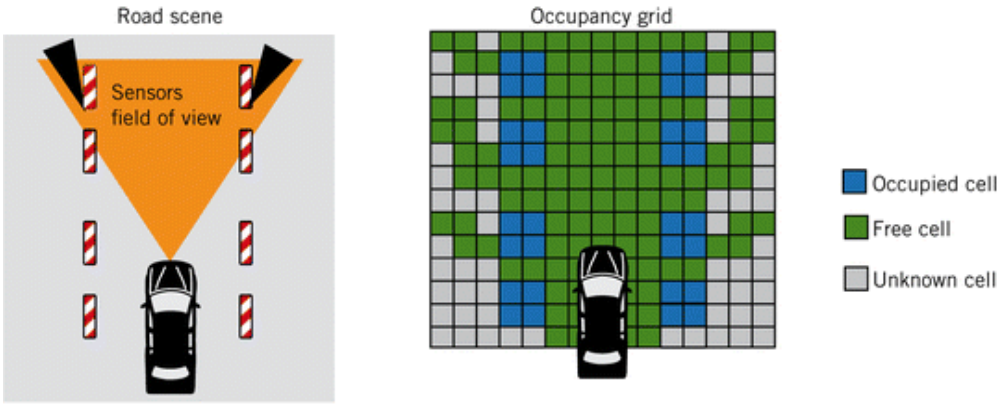


Figure 2.9: Example grid representation [43]. In a single frame, some cells might have unknown statuses, but as the car travels, it might update and track those cells by observing those from different perspectives.

At the same time, the representation of complex road structures, where lanes are not clearly defined, aligning objects' positions accordingly might be difficult with such data representation. Scenarios in which more free-form planning is required, including unstructured intersections, parking lots, or narrow road passages, often require more flexible representation of the environment. This depiction includes freespace, where the driving area is detected from the perspective of the ego car in a line-of-sight manner (see Figure 2.8). Another way of representation is the grid (Figure 2.9), defined as a cell map, where each of them can be classified as being in one of the listed states. Such a representation enables easier tracking and might provide more information about the occluded areas in a given moment, but requires much more computation power to process. Both two- and three-dimensional grids are used, with the latter having the benefit of representing hilly road sections or discriminating between the overdriveable and non-overdrivable areas. Grids as well might include only static context or include as well dynamic object representation within.

In the end, different data representations are often used within a single software stack, allowing the use of different interfaces for different features or scenarios and increasing the robustness of the system.

2.3.5 Localization and Mapping

While driving, humans often rely on their past experiences and memories to recognise patterns of reappearing infrastructure to predict how to navigate complex intersections and act efficiently based on imperfect information. As these abilities are natural to human intelligence, they are hard to realise in a digital system, which is safety critical.

One of the concerns focuses on the locality of perception around the car, which results in a lack of precise information on how the road looks beyond the sensor's field of view. It is possible to define the policy that detects and understands all the splits, merges, and other, often occluded features of the road. It seems however much more straightforward to harvest the data in the form of the map and base the decision-making on that source as well, and do not require reasoning from vague and incomplete information.

To effectively use the map, the car must localise itself within it with a given precision. Global Navigation Satellite Systems (GNSS), including the most recognisable Global Positioning System (GPS), use the constellation of satellites to provide positioning information. Positioning accuracy is subject to noise, which originates from time measurement errors, different wave propagation velocities through space, signal multipath effect, and imprecision in satellite position measurements. As the precision of the standard GPS, which fluctuates around a few metres, is enough for navigation and routing tasks, it is not enough for precise localisation within the lane. To alleviate some of the noise, the concept of differential GPS (DGPS) has been introduced. The DGPS introduces a stationary reference station, placed in a precisely measured location, which determines and sends the differential corrections to another, moveable, receiver, which could be a car. This methodology allows an increase in the accuracy from a few metres to 0.5 - 2 metres. The Real-Time Kinematic measurements (RTK) are currently the most accurate method which can be obtained in real time, which extends the DGPS concept by focussing on the satellite signal's carrier wave, allowing to reduce the position estimation error to centimetres. Furthermore, all of the above methods can use inertial navigation systems based on odometry, allowing one to estimate position changes based on measurements of its dynamic state and tracking methodology, such as Kalman Filter (KF) [53].

Current autonomous driving research uses different types of maps, each with its own pros and cons. Research carried out in geofenced areas often relies on high definition maps (HD-Maps), which the first concept was introduced by Mercedes-Benz [171]. Most often defined with the use of high-precision lidar and camera sensors, they contain very detailed information about static objects such as lane geometries, buildings, traffic signs, and lights along with their properties. Map details allow not only to navigate but also to localise based on the matching of perceived features with those in the map. HD-Maps, however, require expensive equipment to be collected, which impacts their update abilities and scalability potential.

The Simultaneous Localisation and Mapping (SLAM) methodologies combine map creation with localisation tasks [86, 165]. A promising approach that uses this concept with crowd-sourced information has been proposed by the automotive supplier MobilEye. The Road Experience Management (REM) [78] system utilises vision as a primary source of data and benefits from the use of the fleet of end-users' cars, which provides a processed (therefore light) representation of the road. Later, data is aggregated and sent back to users' vehicles in the form of a map, allowing them to enable more features and provide high-precision localisation.

2.3.6 Tracking and Fusion

Perceiving the scene is connected with the inherent noise in detection and is prone to occlusions, often of different kinds for different sensor domains. The standard approach to improving the quality of the estimate focuses on the tracking and fusion of data. Tracking is defined as using a series of readings from a single sensor source, while fusion uses multiple sources. In addition to improving the state estimation quality of measurements, tracking allows one to acquire signals which are not directly measured (like the speed of a vehicle detected by a vision system). What is more, tracking and fusion allow one to provide quality of estimation, which might be used to filter out false positive activations of the system (like Autonomous Emergency Braking (AEB) system braking in front of ghost object) and might "hold up" the object detection even if it is no longer in sensors field-of-view.

Tracking and fusion systems are used in most perception tasks, from object detection to traffic signs. Most obvious example is detection of other vehicles in the scene by given sensor or set of sensors, followed by tracking or fusion algorithm allowing us to estimate state of detected object and provide theirs coherent representation. Both tracking and fusion are very often quite similar to each other with respect to the mathematical concepts on which they stand. The most common approach to tracking a system with linear dynamics and Gaussian noise is KF [53]. For systems with a nonlinear dynamic or observation model, extensions of standard KF have been introduced, such as Extended Kalman Filter (EKF) [2] and Unscented Kalman Filter (UKF) [52]. All of these variants are divided into two steps: the prediction step, where the state and its uncertainty of a given object are updated to a given time moment, and the update step, where new measurements are incorporated into the estimation. Additional steps which needs to be covered as well is object to measurement association, allowing to pick measurement to update correct tracked object or object management, controlling when and how objects are being created and removed. Recently, machine learning-based tracking methods have also started to play an important role in object tracking [104, 156].

The localisation task also is subject to data fusion and tracking. The baseline position on the location of the vehicle is taken from the GNSS system, which is later supported by consideration of motion model measurements and perception-based localisation, including lidar or vision data sources. Perception-based methods are relaying on identification of characteristic points in the environment and localising the car with respect to them with the use of a map. Due to the strong multimodal, non-Gaussian probability distribution of estimated position, they often use particle filter concepts [4]. The main principle of operation relies on spawning a large number of particles, each representing a virtual car placed on a predefined map. Later, each particle is moved accordingly to the real movement of the vehicle, and the rightness of the particle position is estimated based on the similarity between the simulated sensor reading and a real one. The weights of the particles, corresponding to the rightness of their position, is used to estimate final position estimation of the filter. Those approaches are incorporated into the SLAM techniques, introduced in Section 2.3.5.

2.3.7 Situation Assessment, Prediction, and Planning

When knowing what surrounds the ego vehicle and where it is located, the system has to make decisions about what to do. The general term used to describe this part of the system is often referred to as trajectory planning, but encapsulates much more than only planning a trajectory as such. There are numerous

approaches and configurations which are in use currently. Additionally, each of the elements of planning system are highly interconnected to each other, mostly due to closed-loop nature of the system, therefore the architectures are not feedforward in its kind. Most of solutions, however, include blocks realising the function of situation assessment, prediction, behaviour, and trajectory planning.

Prediction and Situation Assessment

To add contextual and functional information to the perception of the ego environment, a situation assessment is performed. An important part of this assessment is the prediction of behaviour and trajectory, which aims to predict future movements of all other road participants beyond their kinematic state. To do so, prediction methods utilise the context information, including other road users participants, road structure, local traffic rules and customs, or even time of a day. The ideal result is a multimodal set of prediction which is dependent on each other, often with equally strong dependence on our own planned actions. With that in mind, in planning the behaviour or trajectory, algorithms aim at finding a solution to multi-agent, closed-loop problem where prediction of other road users and our planned movements fits together and fulfils set of constrains.

There are multiple types of prediction methods, ranging from model-based to data-driven ones. The simplest solutions relay on the current kinematic state of tracked object, and extrapolating its movement to the future. However, such a prediction might be valid only in a limited time frame and in scenarios that do not involve abrupt changes in movement. Model-based approaches usually define common behaviours of the vehicle, like going straight or turning right on the intersection, changing lane, or speeding or slowing down, and trying to identify them in current object behaviour. The authors of [61] provided an extensive survey on prediction and situation assessment methods.

The approaches which currently gain a lot of attention are the data-driven ones based on neural network predictors. Most often, they encode the scene in predefined format (for example, a semantic segmentation map) and try to predict the set of most probable trajectories for each of the objects [19, 32, 100, 110].

The situation assessment is highly interlinked with the prediction task itself as it tries to grasp contextual information about the scenarios in which the ego car is in. It is most often realised as a set of heuristics, which tries to detect if a given scenario is nominal or dangerous, or what is the resulting priority on the intersection. The identification of such traits is later used in nominal trajectory planning, and might be treated as indications of potential firing safety systems, such as AEB.

Behavior Planning

Behaviour planning term is most often referred to the planning of vehicle movement using a high-level command language, which is later translated or implemented in the form of trajectories. Such planned behaviours may involve manoeuvre-based planning, where the actions relate to commanding lane change, lane following, or turning on an intersection, or be setpoint-based directions, indicating the reference speed or amount of bias within the current lane. In most of the cases, the behaviour planning block is not directly responsible for ensuring safety and relays on safe realisation of its output by path and trajectory planning module.

As high-level definition of behaviour is convenient, in the same time it is limiting in intended ego behaviour representations. Implementation of a functional bottleneck in the form of high-level behaviour planner is sometimes omitted in approaches which can directly plan trajectory itself.

An extended review of behaviour planning methods might be found as an introduction to Chapter 3, in Section 3.3, which aimed to put in context the reinforcement learning behaviour planning module introduced there.

Path and Trajectory Planning

The path and trajectory planning modules aim at generating geometrical shapes which the ego should follow. The *path* can be interpreted as a line to follow with any dynamic profile, defined primary with the use of static context. In situations where dynamic context is not existing, like isolated parking manoeuvre or maneuvering in a closed area, the path is a reasonable way of expressing car behaviour (see Chapter 4 for an example of utilisation of the path interface). In case of dynamic objects presence in the scene, a path might still be planned to define the reference line to follow. The plan expressed as a path must be followed by a definition of dynamic movement, defining a concrete velocity profile along it. In cases of heavy interactiveness with traffic, such as urban driving, path planning starts to lose applicability, as it cannot model those interactions well. Path can be well described as a set of points or as a parametric curve.

The *trajectory* is a function of time, which precisely defines the reference position in each moment, and by this velocity and acceleration profiles, it might therefore be directly used as a control reference. As it defines the movement of the ego vehicle to the full extent, it can be used in dynamic environments to model interactions with other vehicles. Due to this, trajectory planning is the method of choice for most autonomous driving methods.

There are numerous methods of planning path and trajectories for autonomous driving use, while most of the methods tries to optimise a cost function while adhering to defined constrains. In [154] a sampling method has been proposed that focuses on defining the initial and end state of the vehicle and fitting the optimal jerk polynomial to it. In Mercedes research [170], authors derived trajectories with local, continues approach which minimises objective dependent on safety, comfort and usefulness while passing set of constrains including vehicle kinematics and dynamics as well as static and dynamic elements of the scene. In [68], Model-Predictive Control (MPC)-based system which automatically decides on the modes of manoeuvres has been presented. In [138], we proposed a method for planning a trajectory that expanded the potential for the execution of emergency manoeuvres.

Currently, trajectory generation research is dominated by AI-based solutions, which tightly ties it to prediction methods. An end-to-end solution, accepting raw LIDAR data and HD map and providing object detection with their predictions as well as cost volume defining the goodness of each position to reach the planning horizon, has been described in [164]. A prediction-based scalable system that allows car dynamics to be incorporated has been presented in [107].

More detailed reviews of motion planning methods can be found in [28, 42, 163]. Reviews of the literature related to these topics are also included in Sections 3.3, 4.3, and 5.3.

2.3.8 Control

The trajectory, which is the most frequently result of the trajectory planning module described above, has to finally be performed by the car actuators. Often, the reference trajectory is already generalised in control parameters space, therefore it is realisable from a car kinematic perspective, but this is not always the case. Such systems most often generate control trajectories from a reference path or a trajectory based on MPC approaches [162]. The use of a mathematical model of a car and a prediction engine enables the generation of trajectories with many control options and potential constraints. The additional benefit of defining control trajectories includes straightforward handling of potential delays in the system.

In the more basic systems, such as entry-level Adaptive Cruise Control (ACC) features, where the trajectory is not planned and the system operates directly on brakes and throttle signals, simple and adequate controllers are used. The most common include the Proportional Integral Derivative (PID) controller and Linear-Quadratic Regulator (LQR), with their derivatives [39]. Still, a lot of customisation is made specifically for the automotive use case, allowing for better behaviour of the final system. As an example of such adaptation, during vehicle cut-in in front of the ACC-regulated car controlled by PID regulator, the integral part of it is often reset to accommodate faster response to a nonlinear change in the system input.

Chapter 3

Goal-based Behaviour Planning With Manoeuvre and Desired Speed Control

3.1 Introduction

One of the challenges of automated driving capability is successful navigation from point A to point B. The system is most often provided with a navigation, which, based on localisation information, provides a list of instructions to follow the selected route. These instructions can be represented as lane-based goals and in most cases are associated with decision points such as splits or intersections. Those lane-based goals are defined to indicate on which lanes we should position ourselves in a given distance, and which additional information about the desired kind of manoeuvres to be executed, in cases where multiple manoeuvres can be performed from a given lane (like driving straight or taking a right turn). In the end, following the route, the car should perform such manoeuvres to end up in the correct lanes at specific locations.

One of the most straightforward policies to follow such a route is to perform accordingly left and right lane change manoeuvres until we will arrive at the correct lane and to stay on it until we will pass defined goal and move to the next one. However, such a policy might not be sufficient in real-life scenarios. As an example, road policies in many countries indicate that cars should stay in the right lane whenever possible. Furthermore, lane navigation decisions should be made in the context of current traffic, which might not allow performing specific lane change manoeuvres at a given time (Figure 3.1) or may cause a situation where leaving the desired lane from the perspective of the route is a reasonable option (Figure 3.2).

As the driving setting is inherently closed-loop and involves many unobserved states, like the level of aggressiveness of a given driver or their own goals, the correct decision in a lot of cases is not straightforward. The optimal policy might drastically change even with small perturbations of the same scenario, as well as is defined based on the current objective, which should represent a balance between comfort, safety, and effectiveness. Due to the plurality of possible scenarios, the obvious tendency is to use data-driven methods, such as machine learning. The closed-loop property of the system suggests the use of the reinforcement learning methodology. At the same time, the rule-based approach brings many benefits and might rule out a lot of decisions for free (like not changing the lane to a non-existent one).

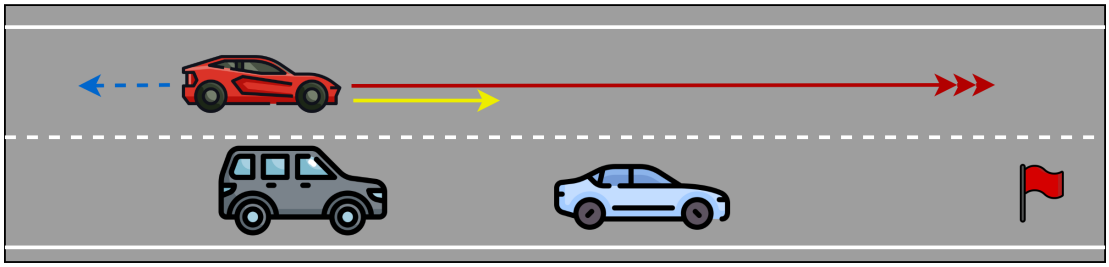


Figure 3.1: The ego (red car) goal is on the right lane and there are multiple strategies for how to get to that lane. In the most aggressive setting (red), ego may speed up and try to squeeze in front of the blue car, while risking missing the goal. Another option (yellow) is to maintain the speed and try to negotiate the space with the grey car, hoping it will be not too assertive and will let the ego car in. In the most conservative option (blue), ego may drop behind the grey car and execute lane change there, however, this may have an impact on fast lane (right) flow.

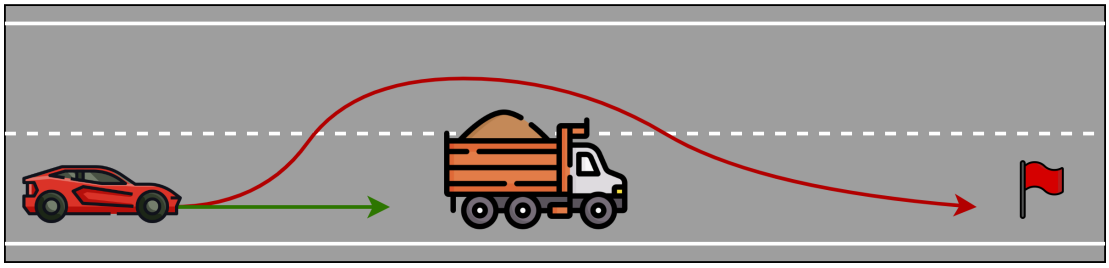


Figure 3.2: The ego car (red) is driving in the correct lane but it is stuck behind a slow-moving truck. The safe option is to stay in the correct lane and slowly continue until reaching the goal (green trajectory). Ego may as well leave the goal lane, overtake the truck and go back to the right lane (red trajectory).

Because of that, a highway navigation problem will be presented with the proposed hybrid system, which will incorporate both a reinforcement learning-based policy and a rule-based system for meeting constraints and trajectory planning.

3.2 Problem Statement

3.2.1 Problem Formulation

In this chapter, a reinforcement learning methodology was applied to produce a policy controlling a car using a high-level interface. The policy, being part of the control system of the modelled ego vehicle, will interact with the design environment, which will model the problem of reaching lane-based goal in multi-lane highway scenarios.

The problem will be represented as Partial Observable Markov Decision Process (POMDP), however, the partial observability issue will not be addressed in a specific way. Each episode starts with spawning an ego car controlled with the RL policy on the highway with other vehicles on it. The scenario, which consists of predefined traffic congestion, number of lanes, and lane-based goal (located on a specific lane at a given distance in front of the ego), is randomly selected. To generate action, RL policy uses observation of the environment, consisting of ego's internal state, desired goal encoding, state of surrounding objects and road structure information. The policy is responsible for defining the action in the form of the car behaviour, which consists of discrete manoeuvre type and setpoint of velocity. Rest of the car control algorithm is within the simulated environment.

The crucial part of the environment is the traffic simulation engine itself, but the environment also includes methods for the interpretation of actions and the creation of observations. In the first step, the defined action is interpreted and realised with the use of the trajectory generation module. Depending on the setup, this action is interpreted in different ways, which is a subject of evaluation in the research. The common element in all setups is moving the ego vehicle accordingly to a defined trajectory coupled with simulation of movement of other road users, which are controlled in a closed-loop manner. In the end, the resulting state of the whole simulation is represented in the form of observation of the environment.

The terminal states of the simulation include reaching the desired goal or missing it, being involved in a collision, or leaving the boundaries of the road. A reward signal is composed with elements promoting reaching the goal as fast as possible, while maximising the comfort of drive described in a form of acceleration magnitude and avoiding undesirable terminal states.

A set of experiments has been conducted that allowed the verification of positive and negative aspects of the introduced mechanisms. For being specific, the agent have been trained with and without the proposed deterministic mechanisms, and the effects of introducing the deterministic constraints after the training without them were examined.

3.2.2 Assumptions and Limitations

In the research conducted, the problem has been simplified by making the following assumptions, which could be limiting when trying to deploy the proposed methods in real-world applications.

- Trajectory planning was realised in two axes separately (first for longitudinal axis and later based on that in lateral one), modelling car as a point (its centre) with specific distance limits representing car dimensions. This was a reasonable assumption in scenarios where movement is dominated by longitudinal part and ego car stays within safe limits of dynamic movement. However, in case of slow moving scenarios the kinematic relationship between orientation and ego movement would need to be handled with the use of more complex motion model.
- For trajectory planning, simple prediction based on the extrapolation of the dynamic state of each object was used. This way of performing the prediction is only valid in short-term horizons or in very stable scenes. Other, more complex methods might be used to address this fact, although a decision has been made not to explore this dimension in the research.
- The simulation of highway scenarios only included a straight segment of the road. It can be argued, however, that by representing the road in the Frenet coordinate system this limitation is negligible for experiment, as even a curve-shaped road would be presented in the same way.
- By simulating the highway environment, some of the existing traffic rules have been represented. There is, however, a significant amount of rules which have not been expressed in training setup, where a lot of them vary depending on specific country. Making sure the agent fulfils all of them accordingly is significant development and legislative effort.
- The policy has access to perfect information about the state of all road users and infrastructure, which is an unrealistic assumption in the case of real-world sensors, which are associated with noise. The issue of modelling perception errors and being robust in case of their existence needs to be addressed prior to moving the solution to a production environment.

3.3 Prior Art

The concept of behaviour planning was originally drawn from research conducted during the DARPA Urban Challenge and was the result of the definition of the standard AD architecture, which has been mentioned in Section 2.3.2. During this research, to narrow the search space for optimal trajectory due to limited computation power and to straightforwardly accommodate traffic rules, the need to control the car at high level arised.

Initially, behaviour planning models used the concept of Finite State Machine (FSM), which has been constructed with a set of states relating to specific modes of driving, between which the algorithm can transit based on a list of conditions [146]. For example, in [35, 81] FSM have been used to control modes in which car is present. The application of FSM in real world scenarios has been shown in [171]. In practise, however, the FSM approach turns out to be unsuitable for the generation of doe behaviour due to poor explainability in complex scenarios, maintainability in case of a need to refine existing behaviour, and scalability to cover a plethora of cases. Behaviour-based systems work by decoupling the decision process into atomic tasks and formulated behaviour in a bottom-up approach. Examples of this include subsumption architectures [18], voting systems [105] or activation networks [71]. In [96], a hierarchical behaviour-based architecture

has been presented to make tactical and strategic decisions, which allows for a combination of different techniques for different levels or scenarios.

The application of reinforcement learning methodology to behaviour planning has initially been studied in the field. In [94], we conducted initial research on behaviour planning, where we used DQN and SAC algorithms to control the car according to Responsibility-Sensitive Safety (RSS) formulations in a similar set of scenarios. Additionally, in the article, we provide a comparison to naively trained low-level control with the use of the SAC training algorithm. Similar research has been conducted in [88], where the authors added simple rules that later override the decisions of the RL agent if necessary. In [74], the behaviour planning agent was trained in a two-lane setting with the Deep Q-Learning (DQN) algorithm to maximise speed and avoid collisions. In a series of articles, the company MobilEye has evaluated the use of reinforcement learning in the semantic control interface, where the agents' actions were carried out according to the state of other road users [118, 119]. During that work, safety rules have been formulated and are currently followed with legislative efforts. In [98], the potential promises and challenges of applying reinforcement learning to automated vehicle motion planning have been summarised.

It is also important to note that with recent growth in computation power and popularisation of data-driven methods, the behaviour planning function is often heavily minimised in scope, as planning occurs as a single step with the use of simultaneous prediction and planning methods [19, 100]. In that case, the behaviour definition often defines the constraints or goals for planning.

Review of other behavior planning methods, which found its application in autonomous driving, has been provided as well in [42, 163].

3.4 Behavior Planning Environment Description

As described in Section 2.1, the reinforcement learning process is formulated as the interaction of the agent with the environment to find a policy that maximises the discounted accumulated reward. From the perspective of the search for optimal policy or training algorithm development, the environment is treated as a constant benchmark used for evaluation. This is an analogy to training data sets used for the evaluation of supervised learning algorithms and models. Nevertheless, when the objective is the application of reinforcement learning methods to an existing real-world problem, such as automated driving, the design of the environment itself presumably becomes the most extensive research and development area. In the context of presented application, such a design would include a form of perception of the outer world and creating its observation, a way of interpreting of agent's action, the definition of traffic, covered scenarios during the training, additional, non-trainable elements of the policy, and finally definition of the reward signal. These design choices would have implications on the properties of the environment, such as its transparency, safety assurance, required information availability, and controllability of the car, which will have a direct impact on training efficiency and overall functionality-wise performance of the policy.

In the following sections, the most important elements of the environment are presented and described.

3.4.1 TrafficAI Simulation Tool

To train the reinforcement learning agent, an appropriate traffic simulation tool must be selected. During the preparation for the research, a list of requirements which such software should fulfil was prepared.

- Simulation shall allow for full control over it, allowing one to control it via the Python language interface, including scene initialisation, stepping through simulation, and controlling car behaviour.
- Software must allow closed-loop simulation with behaviour models of other road users.
- The software shall be available with a beneficial licencing model, allowing one to run simulation in multiple instances at once.
- The solution must be lightweight and fast, which would allow to run it on a single CPU without usage of GPU.
- Simulation shall be able to generate randomised scenarios, prescribed in a high-level manner.
- Simulation shall be able to read in map data, or generate it randomly.
- Simulation shall be able to simulate reading from different virtual sensors located in a car.
- Basic visualisation of the simulation state must be available.

Based on the research, the simulation engine that was selected as a base was TrafficAI [135]. The tool was originally developed by the Simteract company and was adopted by the same supplier to meet all of the listed requirements as a joint effort of Simteract and Aptiv.

TrafficAI is a multi-agent, closed-loop simulation of the traffic environment, both for the highway and urban scenarios. With respect to the static environment, the simulation includes a multiple-lane road simulation with features such as junctions, merges, or splits, along with traffic signs, traffic lights, or pedestrian crossings. Such road structures may be created both manually by prescribing a parametrizable configuration file or may be based on real-world map sources, such as OpenDrive or Open Street Map. When it comes to the dynamic part, multiple users of different types, such as cars or trucks, can be simulated.

Controlling of an agent may be realised by direct state setting, using kinematic model simulation, or with the use of a dynamic model of a car. Agent control may be delegated as well to the simulation engine. In such a case, a parametrizable behaviour model is used to decide on agents' actions. By such a parameterisation, different driver types, such as aggressive or rookie, can be defined.

The tool also allows for defining a basic portfolio of simulated sensors, which handles both field-of-view limitations and occlusions. By placing the sensors in specific locations in the car, we may represent the target car set-up that we are trying to recreate. Later, while using simulation, users may query-specific sensor stream or sensor portfolio about specific objects (other road users) detected by it.

During the co-development of TrafficAI, an idea of the *high-level scenario* has been introduced, which allows defining the properties of both road infrastructure and traffic in parametric form. With the use of this, users may define features such as a sequence of road structure elements, the density of traffic per each

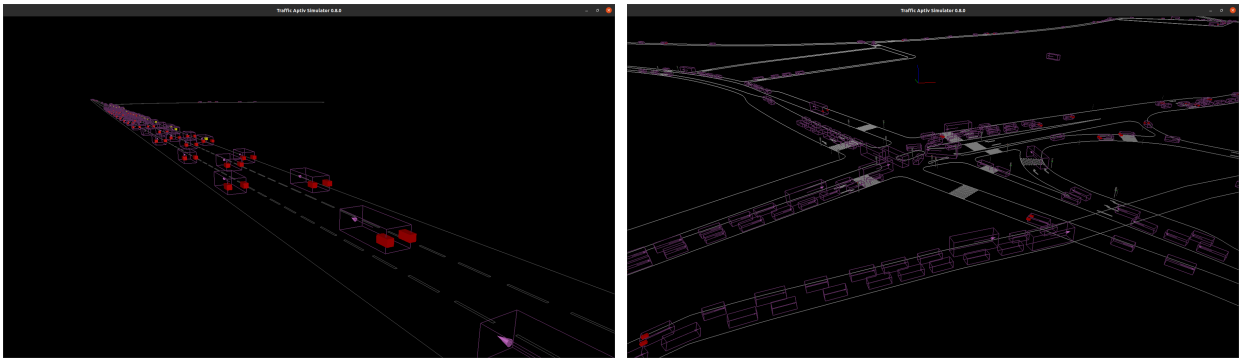


Figure 3.3: Visualized TrafficAI simulation. Road structures are presented in white; cars in the form of bounding boxes are presented in magenta. On the left, is a basic use case of highway environment simulation. On the right, the more complex urban scenario with high-density traffic, junction and overpass.

road or lane, a portfolio of driver types along with their navigation goals. On the basis of that, variations of on-road scenarios can be automatically generated, still allowing for the closed-loop behaviour of other road users.

The tool also allows for basic visualisation, representing cars as bounding boxes and drawing a road in the form of lines.

The main contributing changes to the simulation involved better support of reinforcement learning application needs and closing the gap to automotive-grade simulation tools. The list of changes included an option to add artificial sensors, a new definition of scenarios (also known as the High-Level Scenario), adding a dynamic model for agents' movement simulation, lightweight visualisation, and scenario replay features. With these adaptations, the TrafficAI simulation tool has become a solid training and testing ground for the application and has gained the potential to be used as well as a general tool for the development of automated driving features.

Having all the elements listed above, the user may step through the simulation. In each step, the simulation is queried about the road and traffic features around the ego vehicle, which may come from both reference data and sensor models. On the basis of that, the decision may be made on how we would like to act in response to it. This process emulates the behaviour of the onboard system in a car with ADAS or automated driving capabilities. For the presented use case, simulation is used as a part of the reinforcement learning environment, with specific additions around it, explained in detail in further sections.

3.4.2 TrafficAIEnv

With TrafficAI simulation as the core of the test environment, a general reinforcement learning environment, TrafficAIEnv, has been created [92]. It wrapped the simulation and adds to it specific dynamics. A very high-level overview of it is presented in Figure 3.4. TrafficAI implements both standards, open source OpenAI Gym Environment [17] interface, and it may be extended to a multi-agent setting, supporting the RLlib [65] interface version of such environment. In both settings, the two most important functionalities are *reset* and *step* methods. The first one is called always at the beginning of an episode, and it is responsible for setting up a whole new episode, which is supposed to be simulated, ending up with returning initial observation for the

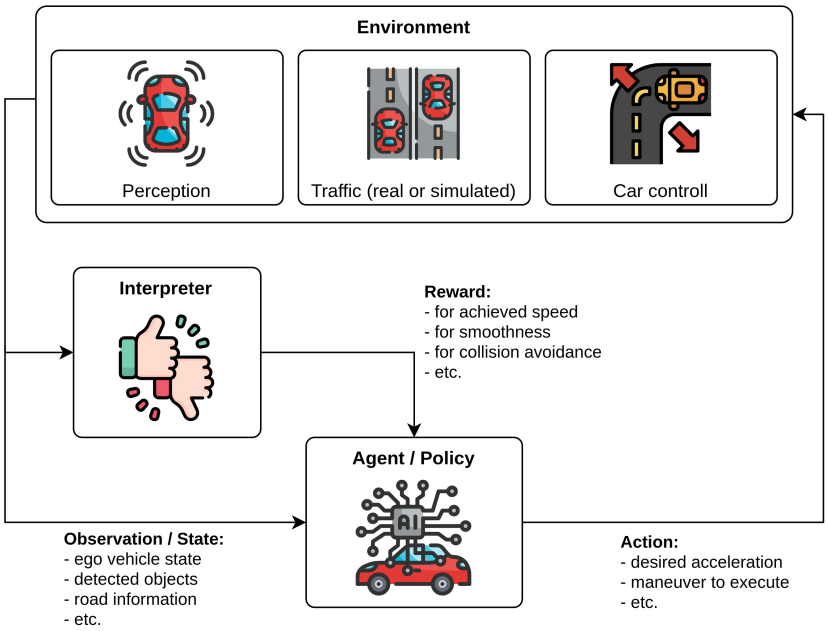


Figure 3.4: The building blocks and scheme of agent (policy) interaction with the designed environment from its perspective. In each timestamp, based on the ego car's perception systems, which eventually form environment observation, the policy decides on action to execute. Action is defined as an acceleration command, manoeuvre to execute, or analogues control signal. This action is further interpreted and parsed by trajectory generation and control blocks. Next, with low-level control defined for the ego, behaviour is executed within the traffic, which is simulated or the real one. Then, the ego car's perception systems are queried again, resulting in a new state (observation) for the next time instance. Along with this new observation, the policy decision, and in general environment state, is evaluated and summarised in the form of a reward signal, whose value depends on such qualities as achieved speed, smoothness, and safety.

agent. The *step* method is responsible for interpreting the consumed agent's action and moving the whole environment to the next time instance, while producing the next observation, reward signal, and indication if the current episode has terminated.

Because during the project many different settings would and should be tested, the elements of the internal environment have been designed to be very general and configurable. To support this requirement, a concept of pipelines operating in states has been introduced and implemented. The initial idea came from [7], however, it has been refined and extended.

Each agent, which is supposed to be controlled by a reinforcement learning algorithm, has a state, which is a dictionary gathering such information as current position, goal, perceived road, and other road users from its perspective, etc. For general information about the environment, such as the simulation handle or update time, the special placeholder is reserved as another artificial agent. On the basis of that, different pipelines are defined, which are the sequences of function calls. Those calls may be characterised as environmental ones, which are supposed to modify the general state of the environment, and agent-specific ones, which update the state of a specific agent. An example of the first type is simulation update, which happens once for all agents, while queering observation for a given agent is an instance of an agent-specific one. Depending on

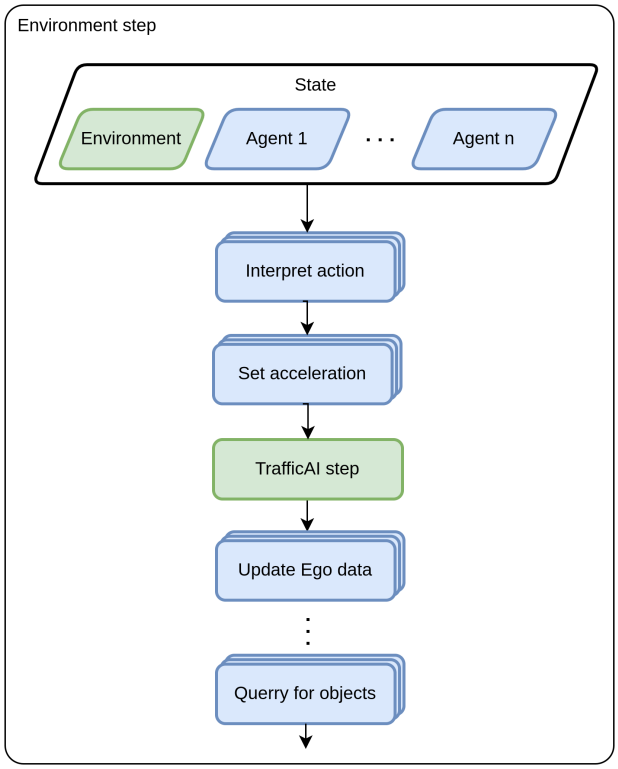


Figure 3.5: An example of an environment pipeline, executed for step function of the environment. The green colour indicates general data and pipeline steps of the environment, while blue corresponds to agent-specific elements. In presented examples, agent-specific modules such as *Interpret action* or *Query for objects* are executed for each agent by consuming the corresponding agent state and general environment one, while steps such as *TrafficAI step* once per the whole environment.

the type, each function call accepts a set of parameters required by it and updates other elements of the state. The calls are then executed in sequence for each agent or environment.

Based on pipeline concepts different processes happening within the environment are realised, such as initialisation, step function, observation creation, or visualisation. By different definitions of such processes, different concepts for observation shapes, action interpretations or internal agent dynamics may be tested. The reward calculation process is handled in a slightly different manner, to allow for the easy accumulation of rewards of a specific agent and cross-agent operations, like depending on one agent's rewards on another agent's performance.

In addition, the environments have been wrapped to simulate multiple different scenarios. By that, multiple-variable-difficulty functional setups can be simulated. Additionally, the curriculum learning [126] approach can be realised, gradually increasing the difficulty of the environment by changing the distribution of different scenarios.

The general flow of information within the environment is presented in Figure 3.6. This scheme and its realisation vary depending on the concrete setup, but is general enough to present dynamics and steps executed within the environment. Specific observation and reward mechanisms are presented in the following sections, which correspond to concrete functional setups.

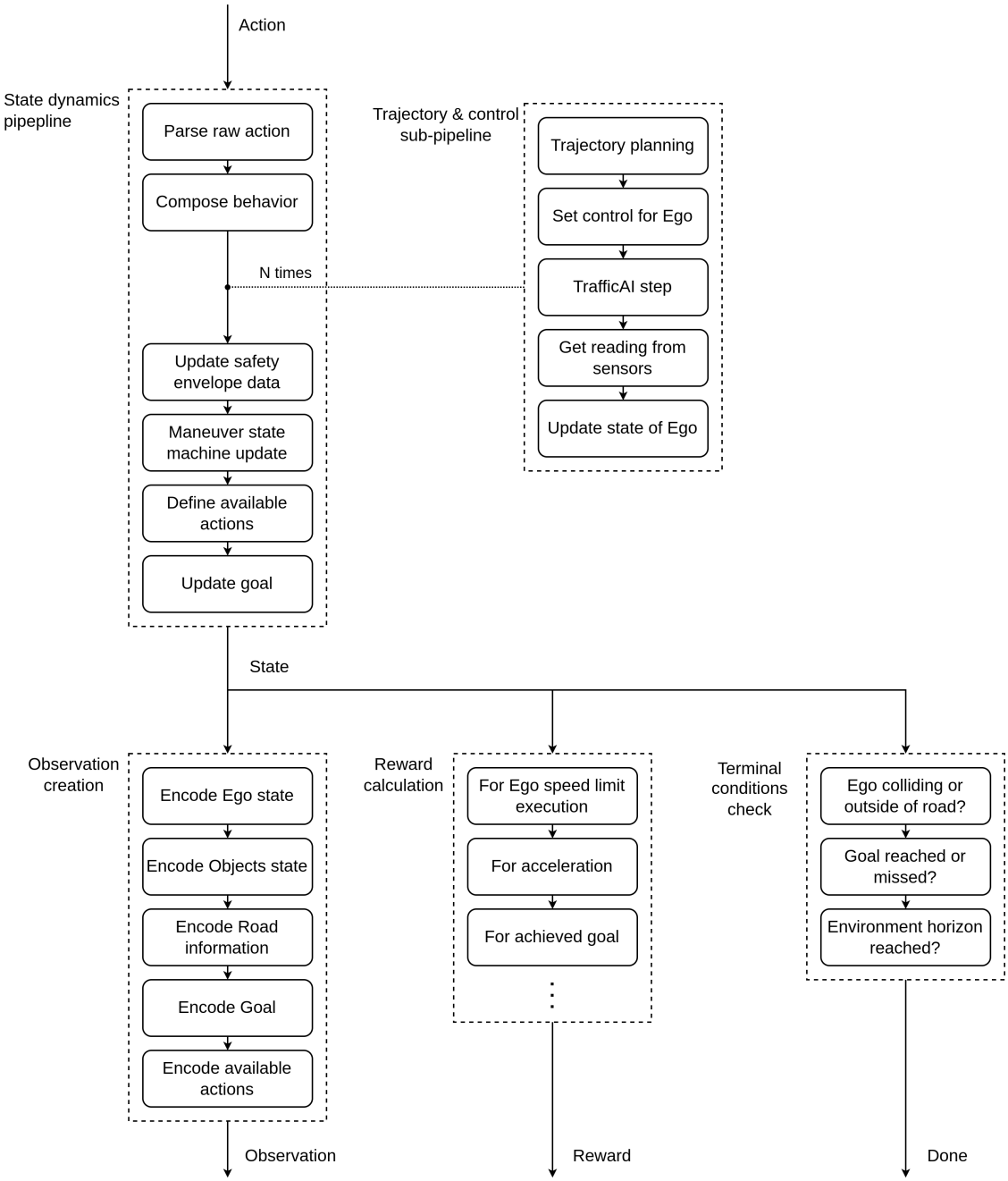


Figure 3.6: Graphics represent the general flow of information in the TrafficAI Environment in a step function. First, the transition to the next time instance is realized by a series of steps, consisting of action parsing, execution of that action in the loop in *Trajectory and control sub-pipeline* and data gathering afterwards. Later, based on the updated state the new observation is created, along with recalculated reward and an indication of episode termination.

3.4.3 Observation Space

To be able to run the NN-based policy, the state of the environment must be encoded in the appropriate format. Its structure is presented in Table 3.1. All values were normalised to the range $< 0, 1 >$ or $< -1, 1 >$ or were a set of discrete values. Firstly, the state of the ego vehicle itself was encoded, summarising its dynamic state, history of executed manoeuvres, and providing information about the availability of lane change. Secondly, the goals state have been parsed. In the test scenarios presented in this work, only situations with a single goal in a single lane have been tested. The state of each goal consists of distance to it and indication which lanes are the target ones (which enables the option to have more than one possible lane to reach). Next, the state of dynamic objects around the ego car has been encoded. Each encoding consisted of the indication of the validity of the given object, its relative position and dynamic state, its dimensions, and the discrete position indication, including its lane, being in front or in the back, and the order with respect to other objects. The last information was the status of the object blinker. The road was represented in a form of a list of lanes states, each including information about its relative position to the vehicle, when it opens and closes and about availability of its adjacent lanes. This information was important in cases where scenarios with lanes openings and closings were simulated. Lane information was indicated in the available manoeuvre mask, which was used later in the action selection mechanism.

The proposed observation space was used in all experiments, although in some parts of it was not effectively used in specific scenarios (like the ones without opening and closing lanes). the model, however, was used in the general version, while understanding that some of its input might not be used in a narrowed scenario portfolio.

3.4.4 Action Space

As already mentioned in introduction, the RL-policy was assigned the role of behaviour planner. In the case, the behaviour was defined as the manoeuvre to execute along with the velocity set point. The action space for the experiments has been designed as a multi-discrete head, therefore the network was providing independent probability distributions for manoeuvre and velocity parts. The details of the action space are presented in Table 3.2.

3.4.5 Reward Function

Reward function designed had to be composed of few specific components responsible for different qualities of the desired behaviour. The final reward r has been calculated as the sum of the weighted components c_i according to Equation 3.1. The specific components of the rewards have been listed in Table 3.3.

$$r(s, a, s') = \sum_{i=0,1,...,N} c_i \tag{3.1}$$

The base component of the reward function is the execution of the speed limit, which provides an incentive to get to the goal lane as fast as possible with respect to the speed limit. Rest of the components by their purely negative effect on the return are used to shape ego behaviour and limit undesirable situations.

Table 3.1 Observation space used for behavior planning scenarios.

Values	**Space**	**Range**	**Description**
ego	[10]		State of the ego vehicle
d		$< -1, 1 >$	Lateral position in the Frenet coordinate system
v_s		$< 0, 1 >$	Longitudinal velocity in the Frenet coordinate system
v_s^{limit}		$< 0, 1 >$	Absolute velocity saturation with respect to speed limit
v_d		$< -1, 1 >$	Lateral velocity in the Frenet coordinate system
a_s		$< -1, 1 >$	Longitudinal acceleration in the Frenet coordinate system
a_d		$< -1, 1 >$	Lateral acceleration in the Frenet coordinate system
θ		$< -1, 1 >$	Orientation in the Frenet coordinate system
m		$\{0, 1, ..., 5\}$	Encoding of the last manoeuvre executed by the ego
l^{left}		$\{0, 1\}$	Left lane change available
l^{right}		$\{0, 1\}$	Right lane change available
goals	[2, 8]		Vector of goals on the road
goal	[8]		Goal description
s^{g}	[1]	$< 0, 1 >$	Distance to the goal
e^{g}	[7]	$\{0, 1\}$	Lane goals vector
objects	[10, 14]		List of states of objects around the ego
object	[14]		State of a single object
e_k		$\{0, 1\}$	Indication if the k-th object is valid
s_k		$< -1, 1 >$	Longitudinal position of the k-th object in the Frenet coordinate system
d_k		$< -1, 1 >$	Lateral position of k-th object in the Frenet coordinate system
w_k		$< 0, 1 >$	Width of the k-th object
l_k		$< 0, 1 >$	Length of the k-th object
v_k^{s}		$< 0, 1 >$	Longitudinal velocity of the k-th object in the Frenet coordinate system
v_k^{d}		$< 0, 1 >$	Lateral velocity of the k-th object in the Frenet coordinate system
a_k^{s}		$< -1, 1 >$	Longitudinal acceleration of the k-th object in the Frenet coordinate system
a_k^{d}		$< -1, 1 >$	Lateral acceleration of the k-th object in the Frenet coordinate system
θ_k		$< -1, 1 >$	Orientation of the k-th object in the Frenet coordinate system
l_k^{rel}		$\{0, 1, ..., 9\}$	Relative id of the lane on which k-th object is
p_k		$\{0, 1\}$	Indication if the k-th object is in front or in the back from ego
o_k		$\{0, 1, ..., 9\}$	Indication at which position k-th object is (is it first, second, etc.)
b_k		$\{0, 1, 2\}$	Encoding of the blinker status of k-th object: off, left or right
lanes	[7, 8]		Lanes description
lane	[8]		Description of a single lane
e_i		$\{0, 1\}$	Indication if i-th lane is valid
l_i^{rel}		$\{0, 1, ..., 6\}$	Relative position of i-th lane
o_i^{in}		$< -1, 1 >$	Information about when the i-th lane opens relative to the ego position
c_i^{in}		$< -1, 1 >$	Information about when the i-th lane is closing relative to ego position
$a_i^{\text{left,in}}$		$< -1, 1 >$	Indication in which distance left lane w.r.t. i-th lane will be available from it
$a_i^{\text{left,to}}$		$< -1, 1 >$	Indication to which distance left lane w.r.t. i-th lane will be available from it
$a_i^{\text{right,in}}$		$< -1, 1 >$	Indication in which distance right lane w.r.t. i-th lane will be available from it
$a_i^{\text{right,to}}$		$< -1, 1 >$	Indication to which distance right lane w.r.t. i-th lane will be available from it
maneuvers	[6]	$\{0, 1\}$	Mask vector of available maneuvers

Table 3.2 Action space used in behavior planning scenarios. Both maneuver and velocity represent probability distributions over discrete set of actions.

Action	**Space**	**Description**
maneuver	[6]	State of the ego vehicle
FOLLOW_LANE		Stay within current lane
PREPARE_FOR_LANE_CHANGE_LEFT		Turn on left indicator and execute the follow-lane manoeuvre
PREPARE_FOR_LANE_CHANGE_RIGHT		Turn on right indicator and execute follow-lane manoeuvre
LANE_CHANGE_LEFT		Execute lane change to the left lane
LANE_CHANGE_RIGHT		Execute lane change to the right lane
ABORT_LANE_CHANGE		Go back to the origin lane during lane change and prepare for lane change
velocity	[7]	Vector of velocity deltas
-3		
-2		
-1		
0		Delta velocities added to the current ego speed to derive a new velocity setpoint
1		
2		
3		

Table 3.3 Reward components used in behaviour planning experiments.

Component (c)	**Range**	**Description**
speed limit execution	$< 0, 1 >$	Average speed with respect to the speed limit, provided that the agent successfully arrives at the goal. When the agent misses the goal, the reward is set to 0.
negative acceleration	$< -0.054, 0 >$	Negative reward for acquiring negative acceleration (braking). Value provided in each step, saturated at -$3\mathrm{m}/\mathrm{s}^2$
maneuver execution	$< -0.01, 0.0 >$	A negative reward provided when the agent changes the manoeuvre, otherwise zero. They are used to minimise instances of lane change manoeuvres. Values for specific maneuvers: - Follow Lane: 0.0 - Prepare for Lane Change: -0.0001 - Lane Change: -0.0001 - Abort Lane Change: -0.01
emergency manoeuvre	$< -0.01, 0.0 >$	Negative reward for emergency manoeuvre execution. Equal to -0.01 at each step when an emergency brake must be applied, therefore, more important in the need for severe brake, otherwise 0.
terminal states	$< -1.0, 0.0 >$	Negative reward of -1.0 for one of the terminal states ending the episode, including collision or getting out of the road by invalid manoeuvre. Otherwise, 0.

This includes limiting braking, the number of manoeuvre changes, the need to execute an emergency braking manoeuvre, and ending up in any undesirable terminal states.

The reward function has been kept in the same form for all experiments, but due to the changes introduced in the action selection mechanism, trajectory generation mode or manoeuvres FSM some of its components were inactive or much less impactful.

3.5 Hybrid System Architecture

As already noticed in Section 3.1, there are obvious benefits of using deterministic, rule-based methods as well as AI-based methods in behaviour planning. Unfortunately, combining these two approaches is not trivial. The difficulty arises in fact because the agent has to train itself in a closed-loop system, which might be heavily impacted by those deterministic rules. This stands contrary to system design where each block has a specific task to execute which is well defined and independent except for interface reliance.

Imagine a well-known feature function of adaptive cruise control, where the speed of the car is regulated based on user input but at the same time might be decreased in case of a slower moving vehicle in front of the ego car. In such a system, two distinctive parts can be defined: one is responsible for the velocity set point, which is the desired speed of travelling, and one aiming at keeping that reference unless objects in front of the ego car does not enforce its limitation. The first part might take into consideration the driver preferences, consecutive speed limits, or comfortable, maximum velocities in curves. The second part interprets the derived set point velocity as an desire, and by understanding the dynamics of the car tries to keep it, while in the same time is responsible for limiting the ego speed to keep safe distance to objects in front.

In case of a rule-based system, the first part does not really care if its speed limit is reached by the second part. Although, if we would like to model the first part of the system as RL agent, returned action in the form of velocity setpoint should be executed in predefined way. In the other case, the agent starts to lose understanding of how given actions impact its state and are realised. From the RL agent's perspective, driving behind some slow-moving vehicle and returning any velocity setpoint above current speed has no real effect on the car's behaviour, as it is limited by the ACC part of the system. Of course, the machine learning process will try to deal with that uncertainty, but in the same time the noisiness and lack of transparency introduced by such system design seriously hinders the already challenging optimisation process.

Such system design choices are not only visible in velocity control, but have its own form in manoeuvre selection or defining the acceleration profile. In following sections, we will go through proposed methods and design choices which aims at keeping the environment as transparent as possible but which preserve the ability to inject rules into the system and ensure compliance with them. The hybrid system will be designed, according to the common AD system architecture presented as part of Figure 2.3. The RL-based module will be made responsible for behaviour planning and define the manoeuvre to execute and the velocity at which the car should travel. The deterministic methods will be used both prior to call to AI module and afterwards, and will pre-define the possible actions, ensure safety of planning, and execute RL-defined behaviour.

3.5.1 Safety Envelope and Responsibility-Based Safety Framework

One of the main roles of the deterministic part is to define the rules-based constraints for the policy. Some of the driving rules or traffic laws can be defined straightforwardly and might be assumed ultimate, with the exception of eventual perception errors and safety-critical scenarios. Under nominal driving conditions and with reasonably good performance perception, crossing a solid line or running a red light should never happen and should be excluded from action space.

Having that in mind, the rules that apply to simulated scenarios have been listed, which are applicable to the highway driving conditions:

R1. Do not change lane to non-existing one

R2. Do not cross solid line

R3. Do not cross the speed limit

R4. Do not change the lane when it is unsafe

R5. Do not change lane to one which is ending

R6. Do not drive to close to the car in front

Some of those rules (the first three) are rather straightforward and clear, but the rest needs to be more precisely defined to determine exactly what should be the minimum safe distance to the car in front. To do so, basic formulations from Responsibility-Sensitive Safety (RSS) [118] have been used, which is the mathematical model for the definition of safety constraints.

The RSS framework embeds the understanding that ultimate safety is impossible to achieve and the main focus point should be avoiding causing collisions. Following that, clear rules have to be established to assign guilt (i.e., responsibility) for each event on road and make sure that the ego car will never be the responsible agent. The RSS is introducing a set of five parameterizable rules based on common sense and practise, allowing one to define what it means to drive safely. In the event of braking any of those rules, the ego vehicle would be obligated to perform the emergency manoeuvre, which is a possible "way out" from a dangerous situation. The framework is built and parameterized on the notion of the reasonable worst-case scenario of what might happen on the road, rather than statistical analysis. The framework does not pay special attention to possible perception errors in the system, which should be taken care of in a separate manner.

In case of highway use, the first rule was used primarily, which defines the safe distance to be kept to the car in front. The commonly known "Two Second Rule" is substituted with precise mathematical formulation, taking into consideration the observed as well as assumed kinematic properties of the cars. Equation

$$d_{\min} = v_r \rho + \frac{1}{2} a_{\max,\text{accel}} \rho^2 + \frac{\left(v_r + \rho a_{\max,\text{accel}}\right)^2}{2 a_{\min,\text{brake}}} - \frac{{v_f}^2}{2 a_{\max,\text{brake}}} \tag{3.2}$$

shows the calculation of the minimum safe distance to the leading vehicle, where v_r is the relative speed, ρ is a response time, $a_{\max,\text{accel}}$ is the maximum acceleration which can be applied during reaction time by

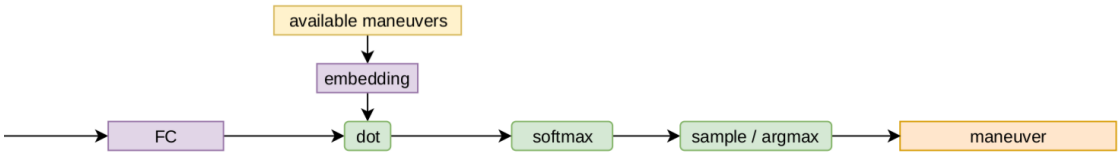


Figure 3.7: The mechanism is used to define the probability distribution only over available actions. The general state of the neural network passes through the Fully-Connected (FC) layer and it is multiplied in a dot-product fashion with the embedding of available manoeuvres, resulting in a vector which assigns value to each of them. After processing this vector with the softmax layer, the result is interpreted as a probability distribution, from which one may select specific action accordingly to its wish (sampling, argmax).

the ego while $a_{\mathrm{min,brake}}$ and $a_{\mathrm{max,brake}}$ are, respectively, the minimum braking deceleration which must be applied as an emergency braking and the maximum braking deceleration which we can expect from a leading car.

Rule number two, which defines the minimum lateral distance along with the lateral emergency braking, has been used when other vehicle movement should trigger an automatic lane change manoeuvre.

The rules mentioned above have been used to define the constraints for Adaptive Cruise Control minimum distance keeping, as well as when it is safe to perform an automatic lane change in the presence of other vehicles in the target lane. In the next subsection, a description of how those rule-based constraints can be injected into the neural network and utilised in the overall system is provided.

3.5.2 Deterministic Available Actions Definition

Having the constraints defined, one has to decide how to utilise them in the system. As already described in Section 3.5, simply downselecting the actions from the NN-based policy afterwards is problematic, as it introduces inconsistency in selected agent's action and its execution, resulting in noise and high variance in the gradient. That problem will be primarily visible during the training, less during the evaluation phase, unless the policy is recurrent or uses the actions history at the output.

To address this, in [95] the concept of constraining the output of the policy neural network to only predefined actions was proposed. The general idea of a specific mechanism of the NN architecture has been taken from [91], where the linear projector over the available actions has been used in the action head. In the original paper, this mechanism was used to limit the number of actions to choose from to suppress an extensive and pointless exploration of the action space. In this case, however, instead of heuristics, the deterministic rules defined and introduced in Section 3.5.1 have been used. With that approach, there was no need to define another reward component for selecting an unavailable action, which as a result does not put stress on gradient calculation. The mechanism introduced in NN is presented in Figure 3.7.

One of the identified issues of taking such an approach is generalisation. Imagine a state s_{l} in which the actions of specific agents are limited by deterministic rules and the state s_{nl} being very closely related to s_{l}, but in which all actions are available. Since in s_{l} the agent cannot select forbidden actions, state s_{nl} cannot benefit from this knowledge because the gradient is never backpropagated in those cases. Putting this in plain English, since an agent cannot make bad decisions, it does not learn from them, so it cannot devalue

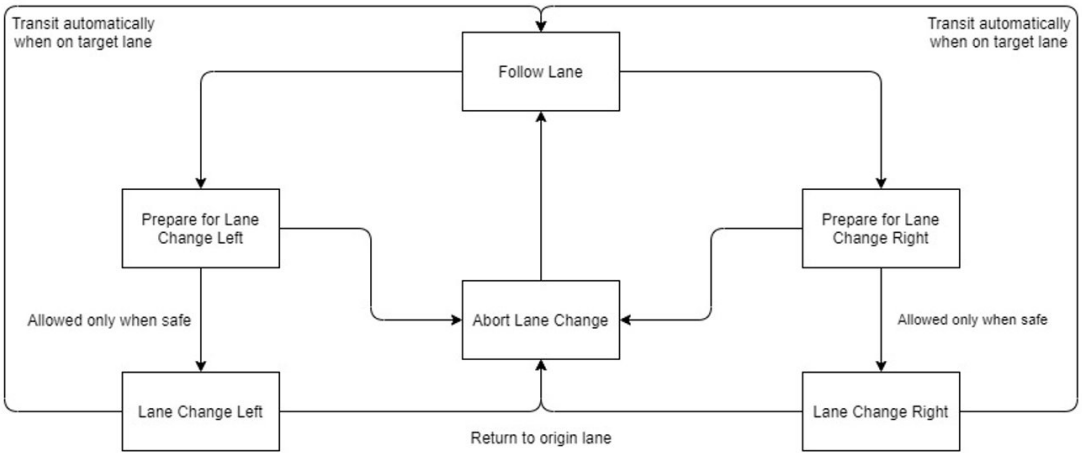


Figure 3.8: Maneuvers Finite State Machine which has been used in combination with action selection by RL agent.

the available decisions similar to forbidden ones. This issue becomes even more problematic in the case of velocity control, which will be described in the following section.

3.5.3 Maneuver State Machine

Understanding that the execution of selected manoeuvres on the road must be ordered and structured, an additional module for manoeuvre planning that has been based on FSM concept has been introduced. The basic structure of manoeuvre FSM was used to dictate what kind of manoeuvre can be executed based on the previous state, to introduce time-based requirements (for example, how long car should keep lane after a lane change) and to easier control lane change execution. The responsibility of performing transitions to specific states has been assigned to the RL agent, with the exception of situations where only one action could be selected. Mechanism of FSM and RL action have been combined accordingly to the method presented in Section 3.5.2. The FSM has been reporting the current state (manoeuvre) and what kind of manoeuvres are available from it. This information was provided to the neural network in the form of a mask vector, which narrowed the possible actions to select. With standard sampling mechanism concrete action has been selected from narrowed action distribution, and used in another step to make a transition to another state. At the same time, the selected manoeuvre was to conditioning the trajectory planning module to generate the appropriate path.

The structure of FSM used in the proposed approach is presented in Figure 3.8. Six distinctive manoeuvres (states) have been defined, consisting of following the path, preparing for a lane change (left and right), lane change itself (left and right), and aborting a lane change. Arrows represent potential transitions between manoeuvres. Additionally, transition availability has been as well masked with the use of current context, which included such rules as

- The existence of the target lane was necessary to allow preparation for the lane change and manoeuvres of the lane change itself to a given side. In the situation in which the ego car was travelling on the edge of the road, the lane change manoeuvres that would put him outside it were masked out.

- After entering some manoeuvres, timing requirements was introduced which did not allowed one to exit given state before predefined time. For example, after going back to the lane following the manoeuvre, the agent was forced to stay in this state for at least one second. This allowed to stabilise the car behaviour.

- Lane change manoeuvres were masked out if they were not safe to perform due to traffic. The RSS distance rules were used to check if entering the lane agent will not cause a hazardous situation and if it was the case, the lane change manoeuvre has been forbidden.

The introduced concept of manoeuvres supervision allows one to better understand the meaning of specific actions for the RL process. Without such a mechanism, the execution of different actions might result in a similar end behaviour of the ego vehicle and might be misleading (imagine a case where the agent selects a change in the right lane during execution of the change in the left lane). It allows as well to more clearly execute manoeuvres with deterministic approach later.

3.5.4 Transparent Speed Control Design

Controlling the speed of a car in combination with any other mechanism of such steering (such as the ACC control introduced in 3.5.5) raises quite a lot of obstacles to the transparency of the control.

The most straightforward idea is to control the velocity setpoint going into the ACC trajectory generation controller. To ensure generality, the target speed has been provided with reference to the speed limit in discrete form. Although the end behaviour of the agent was correct, allowing one to navigate effectively through the lanes, the action was changing quite a lot and was unstable. The reason for that was lack of transparency, as all "slow-down action" was executed in the same way. As an example, in case the car drives 50 m/s, providing a new set point of 30 m/s or 20 m/s results in the same severe deceleration.

To address that, the concept of delta speed control has been introduced. With that, the agent is tasked with providing acceleration-like command which is combined with the desired speed limit. The output then is added to the current ego speed and is treated as a setpoint. Additionally, different maximum acceleration levels are associated with those actions. Due to that, all actions are meaningfully different and might be associated with the desired aggressiveness level. Moreover, the same mechanism considering availability of the actions was used to pre-select only available delta speeds, utilising information about current speed limits and possible executions of emergency manoeuvres. Figure 3.9 presents how the mechanism works.

During the evaluations, the problem of multimodality of the speed distribution has been discovered. To effectively address cases with multiple, equally good actions (see Figure 3.10), the obvious choice is the discrete action space. Here, however, the problem of continuity of the strategy arises. As we would like to represent the multimodality of action distribution in a given scene, we would also like to make sure that we will select one of those strategies and stick to it to see the effects of that decision. Situations where the agent jumps from speed-up to slow-down actions, resulting in roughly the same speed and a quite uncomfortable driving experience, should be avoided. In such a case, working with policy-based methods, returning the probability distribution over actions, is problematic in the case if sampling is enabled. When sampling is disabled and the agent uses the argmax action instead, a non-natural behaviour and suboptimal

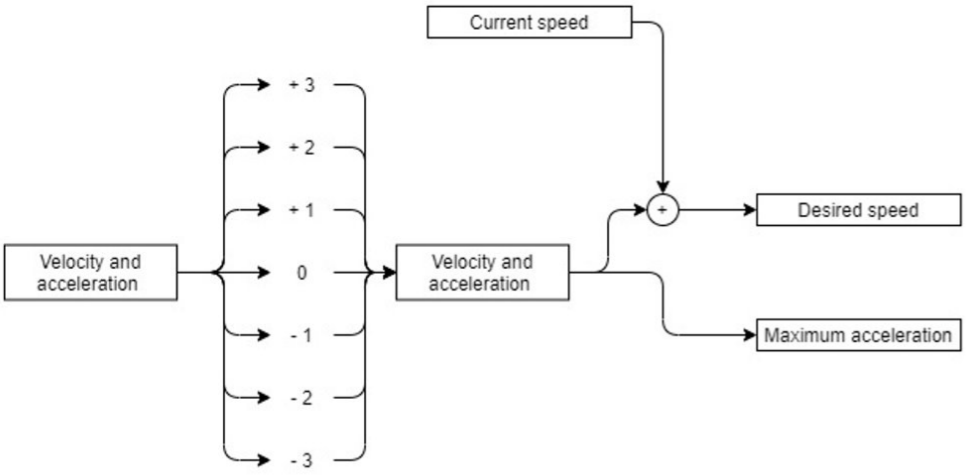


Figure 3.9: The delta speed mechanism. Agent action is interpreted in the same time as desired increment of speed and maximum acceleration level. In that way, agent output preconditions the adaptive cruise control mechanism. Higher absolute acceleration values are associated with more aggressive driving.

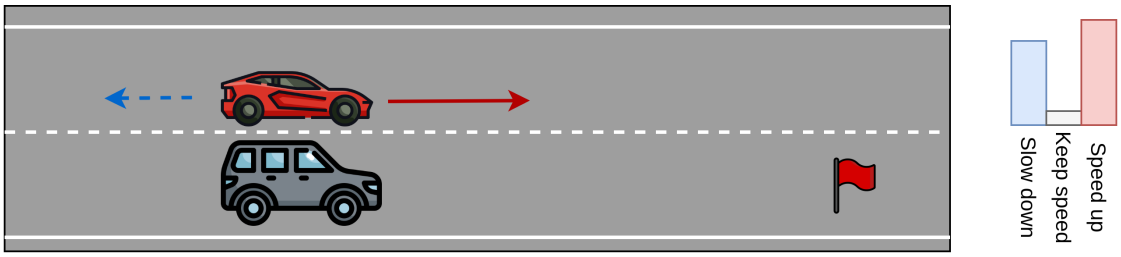


Figure 3.10: Blocked lane change scenario. In this concrete situation, we would like to make sure that action distribution for slow-down, keep-speed and speed-up actions looks more-or-less as one presented in the right side of the image. In the same time however, when the agent decides to select given strategy, we would like to make sure that he will stick to it and do not flicker between slow-down and speed-up actions any more.

performance were observed. The desired property of the system suggests using value-based methods, like DQN, in which the algorithm is not directly optimising the action distribution but the corresponding value-function approximations. This approach detaches the action selection mechanism and exploration from the learning objective itself, which is why switching off exploration at the evaluation time does not impact the policy performance.

3.5.5 Trajectory Generation Module

After defining the behaviour of a car, the model-based algorithm for trajectory planning was used. As a trajectory generation concept, an idea based on pairwise const-jerk velocity cruise algorithm was used. The solution has been inspired by the work done in [154], where two six-order polynomials have been used to plan the motion independently on two separate longitudinal and lateral axes. In the presented case, motion has been planed in the form of a sequence of polynomials, representing the velocity profiles of longitudinal and lateral movement in the Frenet Coordinate System.

The high-level description of the trajectory generation block has been provided as Algorithm 1. Primary input to the trajectory planning algorithm is the target state of the ego vehicle, including its required position, velocity, and acceleration in both axes. Deriving this target state is shared responsibility of RL-agent, which provides velocity setpoint and manoeuvre to execute and set of heuristics, which selects appropriate vehicles to regulate on according to given manoeuvre, by calculating safe positions behind them. Future targets (such as desired position and speed after speed equalisation) are acquired with assumptions of kinematic extrapolation of movements of other objects, which is used to predict their future states. If no target vehicles are significant from planning perspective, the target state does not include positional requirement and focuses purely on keeping appropriate velocity. In case of some manoeuvres, like lane change, ego might need to check regulation on two different target states: in the cars in front in current and target lane. In such a case, the algorithm selects a more conservative velocity profile (one which minimises the velocity value). In all cases, constant jerk is used to generate acceleration profiles, which of course results in velocity and positional trajectories.

After defining the target state or set of them, the process of trajectory generation begins. Depending on the situation, this generation is divided into multiple steps. In the most general case, the ego needs to plan two phases of speed change. As an example, consider a situation in which the ego car drives too close to the vehicle in front, while the agent target speed is still higher than the speed of the vehicle in front (see Figure 3.11). This case might happen during severe braking of the front car or a cut-in scenario. First, the ego needs to slow down below the speed of the vehicle in front, increasing the distance to it. To do so, the first phase is planned to acquire "dropping speed". In the next step, ego plans the last phase of manoeuvre where it speeds up to equalise the speed with the target. By understanding how much distance will be covered in the last phase, the algorithm can establish when the manoeuvre should start. The last step is to plan the cruising phase, the length of which depends on how far the ego car should drop behind the target in front. When checking the distances covered in the first and last phases, the duration of the cruising phase is established. Durations of all specific phases of planning are derived with use of straightforward analytic equations of motion, allowing one to calculate the time of each specific manoeuvre. Polynomials are used to plan each segment of the trajectory, with assurance that the dynamic states at the ends of those segments are equalised.

In a similar fashion other manoeuvres are planned, although they might be composed only from subset of described above phases depending on objects existence, the intention regarding speed setpoint, the current speed and phase of manoeuvre. It is important to note that trajectory planning algorithms details are straightforward but cumbersome, therefore, algorithm is presented only in high-level.

The trajectory is planned for the time needed to reach the target state, and could be prolonged if it is shorter than the planning horizon. For the designed system, the behaviour planning block was called every 0.5 seconds. The trajectory planning block has been called each 0.25 seconds and therefore executed the same behaviour twice. Motion simulation has been done with 0.05 seconds interval, and with the same interval the trajectory to set ego state has been sampled accordingly.

Furthermore, according to RSS rules [118], an emergency manoeuvre has been defined as the application of heavy braking. This was triggered when the ego car violated the safety distance in front of us, defined by

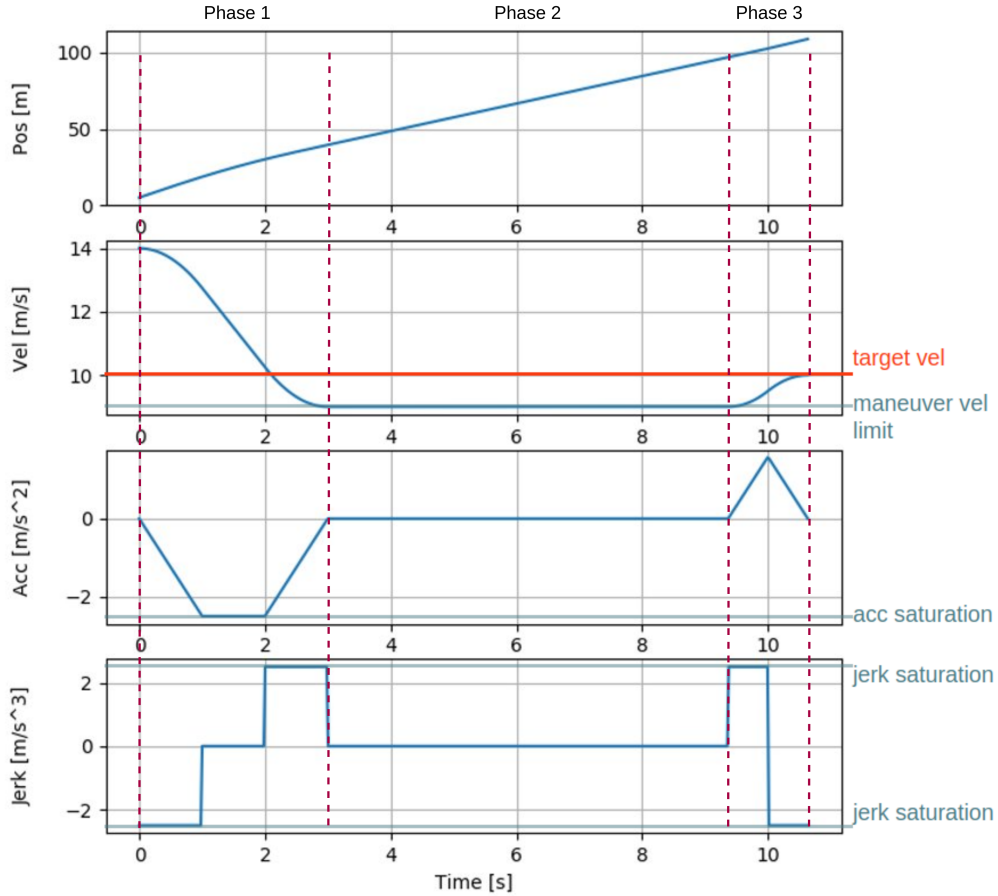


Figure 3.11: Desired trajectory profiles for scenario in which car is too close to car in the front, therefore needs to slow down and then equalize the speed at correct distance.

Algorithm 1 Pairwise Const-Jerk Velocity Cruise Algorithm

Initialize the rl-policy π

Call the policy π to acquire desired maneuver m_π and velocity setpoint v_π

Select set of target vehicles $\{o_1, o_2, ..., o_k\}$ which an ego car need to regulate on depending on the maneuver m_π

if target vehicles are present **then** ▷ Continue in Adaptive Cruise Control mode

  for each valid object o^i **do**

    Calculate desired position s_d^i and desired velocity v_d^i for on each of object o^i

    Plan trajectories with end state s_d^i and v_d^i

  end for

  Create trajectory elements which minimizes the position value across valid time range

else ▷ Continue in Cruise Control mode

  Mark policy velocity setpoint v_π to desired velocity v_d

  Plan trajectory with end state v_d without constrains on concrete position

end if

using RSS rule one. This situation is termed a *safety violation*, and measures the frequency with which the agent ends up in this state.

3.6 Policy Optimisation

3.6.1 RLLib Reinforcement Learning Library

As the research was focused on reinforcement learning methodology, appropriate software was required that implemented the specific algorithms. To focus the attention on research scope, connected with application of autonomous driving, the decision has been made to use a third-party library, which implemented the desired algorithms in a scalable way.

The library that has been chosen was RLLib [65], which has been built on the Ray framework [153]. As a base, the tool introduced the efficient and scalable concept for distributing computing across different processes and nodes, which have to communicate with each other. This stays in contrast to more common applications, in which big jobs consist of many smaller tasks which can be executed in parallel more or less independently.

RLLib package implements many different reinforcement learning algorithms which are optimised from the point of view of performance. It allows for easy scaling-up of experiments to thousands of workers. RLLib supports the most common deep learning frameworks, which are PyTorch [99] and Tensorflow [73]. It also allows for hyperparameters tuning sessions with the use of its package Tune.

RLLib Trainers classes define and coordinate distributed workflows, consisting of running rollouts, responsible for experience collection, and optimisers, aiming at improving the policy. The policies define how an agent acts in an environment, supporting different cases such as single-agent gym-like environments, vectorised environments, and multi-agent environments. The policy evaluation part, running in rollout workers, is responsible for collecting experiences and implementing the agent-environment interaction loops. Those experiences are represented in the form of batch samples that encode one or more fragments of trajectories. The interaction between the elements mentioned is defined in the form of execution plans, which prescribe the sequence of steps that must be executed sequentially in the training process or in parallel by many experience collecting actors.

3.6.2 Training Infrastructure

To efficiently run the experiments, specific computational resources were required. For that purpose, an on-premise cluster of computers has been used, including nodes with CPU and GPU units. The cluster was constructed with about 380 CPUs and up to 15 GPUs of different class. The cluster is managed with the SLURM workload manager [124]. Each training starts with defining the resources, including the number of CPUs, GPUs, and RAM memory requirements. Later phase is managed by RLLib and Ray framework itself, and coordinates tasks from a single job, executed potentially on different nodes.

3.6.3 Proximal Policy Optimisation Algorithm

The algorithm used in the described experiments was PPO [115], which is part of the policy gradient algorithms. PPO is an on-policy algorithm and can be used for environments with continuous and discrete action spaces. The core idea behind the implementation of the PPO algorithm answering the question of how big steps can be made in policy improvement while not going too far and collapsing performance.

PPO can be defined in two main variants: PPO-Penalty and PPO-Clip. The *PPO-Penalty* solves the KL-constrained by approximation similarly to TRPO [114], however, instead of using it as hard constrain, penalises for the KL-divergence in the objective function, while adjusting the penalty coefficient during the training automatically. The *PPO-Clip* variant does not utilise the concept of KL-divergence term and does not have constraints, but instead relies on the specific clipping mechanism within the objective function to eliminate any reasons for a new policy to get too far away from the old one.

The PPO-Clip version updates the policy parameters θ via

$$\theta_{k+1} = \underset{\theta}{\operatorname{argmax}}\, \mathbb{E}_{s,a\ \pi_{\theta_k}} \left[l(s, a, \theta_k, \theta) \right], \tag{3.3}$$

which is typically done by taking multiple steps, often in a mini-batch manner. By defining the probability ratio between the new and the old policy as

$$\rho(\theta) = \frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)}, \tag{3.4}$$

we can define the l objective as

$$l(s, a, \theta_k, \theta) = \min \left[\rho_t(\theta) A^{\pi_{\theta_k}}(s, a),\ \operatorname{clip}\left(\rho_t(\theta), 1-\epsilon, 1+\epsilon\right) A^{\pi_{\theta_k}}(s, a) \right]. \tag{3.5}$$

The first term inside the $\min$ function is a surrogate objective familiar to the TRPO algorithm. The second one modifies this objective by clipping the probability ratio to the $[1-\epsilon, 1+\epsilon]$ range, which makes sure that update of policy's parameters will not be too large. The clip function is defined as

$$\operatorname{clip}\left(x, x_{\min}, x_{\max}\right) = \max(\min(x, x_{\max}), x_{\min}). \tag{3.6}$$

Then, the minimum of those two terms results in a lower bound on the unclipped surrogate objective.

For the PPO-Penalty variant, the objective includes the term penalizing from going away from the original policy

$$l(s, a, \theta_k, \theta) = r_t(\theta) A^{\pi_{\theta_k}}(s, a) - \beta KL \left[\pi_{\theta_k}(\cdot|s), \pi_\theta(\cdot|s) \right]. \tag{3.7}$$

The β parameters are automatically adjusted during the course of training in a straightforward comparison to some target KL divergence. On the basis of experiments, however, the PPO-Clip variant yields better results. For advantage, any method for its estimation can be used, but most commonly, value function estimation is utilised for that purpose.

The pseudocode of PPO with clip variant of objective is presented in the following scheme.

PPO uses and optimises stochastic policy in an on-policy way, thus exploration is embedded within the optimised policy, and exploitation-exploration trade-off depends on initial conditions and training procedure. In most cases, the policy becomes less random during training, so it is more likely that it will stuck in local

Algorithm 2 PPO-Clip

Get initial policy parameters θ and value function parameters ϕ

$\theta_0 \leftarrow \theta$

$\phi_0 \leftarrow \phi$

for iteration $i = 1, 2, ...$ **do**

  for actor $a = 1, 2, ..., N$ **do**

    Collect set of experiences $D_i = \tau_k$ by running π_{θ_i} in environment for T timestamps

    Compute rewards-to-go $\hat{R}_t$

    Compute advantages estimates $\hat{A}_1, \hat{A}_2, ..., \hat{A}_T$, using value function estimate v_ϕ

  end for

  Update the policy π_θ by maximizing the surrogate loss value:

  $\theta_{i+1} \leftarrow \text{argmax}_\theta \frac{1}{|D_i|NT} \sum_{\tau \in D_i} \sum_{t=0}^{T} l(s_t, a_t, \theta_k, \theta)$

  Run regression on mean-squared error for value function:

  $\phi_{k+1} \leftarrow \text{argmin}_\phi \frac{1}{|D_i|NT} \sum_{\tau \in D_i} \sum_{t=0}^{T} \left(V_\phi(s_t) - \hat{R}_t \right)^2$

end for

optima as training progresses. On the other hand, stability of optimisation is often also a source of consistent improvement of optimisation objectives, with rare moments of performance drop.

3.6.4 Neural Network Architecture and Training Details

The neural network architecture that has been used for the agent policy is presented in Figure 3.12 and was based on the architecture of [91]. Processing starts from the ego-related and goal observation. Later, data describing each individual object and lane are processed by a corresponding block, mostly composed of Fully-Connected (FC) layers. The challenge of changeable number of objects and lanes across the scenes was solved by a max-pooling operation performed after the Rectified Linear Unit (ReLU) activation function. With that design, invalid objects, padded with zeros, were excluded from further processing. Categorical data types, such as the last executed manoeuvre or blinker status, are introduced into the processing through embeddings. The output of each individual block is then concatenated and pushed through the Long Short-Term Memory (LSTM) unit. As the final stage of the calculation, the action head provides separate action distributions for manoeuvre and velocity actions. Additionally, the mechanism introduced in Section 3.5.2 is present at the bottom of the architecture, both for the manoeuvre and velocity action heads.

For training, the PPO algorithm has been employed with parameters which are listed in Table 3.4. Trainings have been conducted in the local cluster and have been run with 90 rollout workers and a single trainer process with a GPU unit. Decisions to stop training were made when performance, expressed as mean reward, had stopped changing.

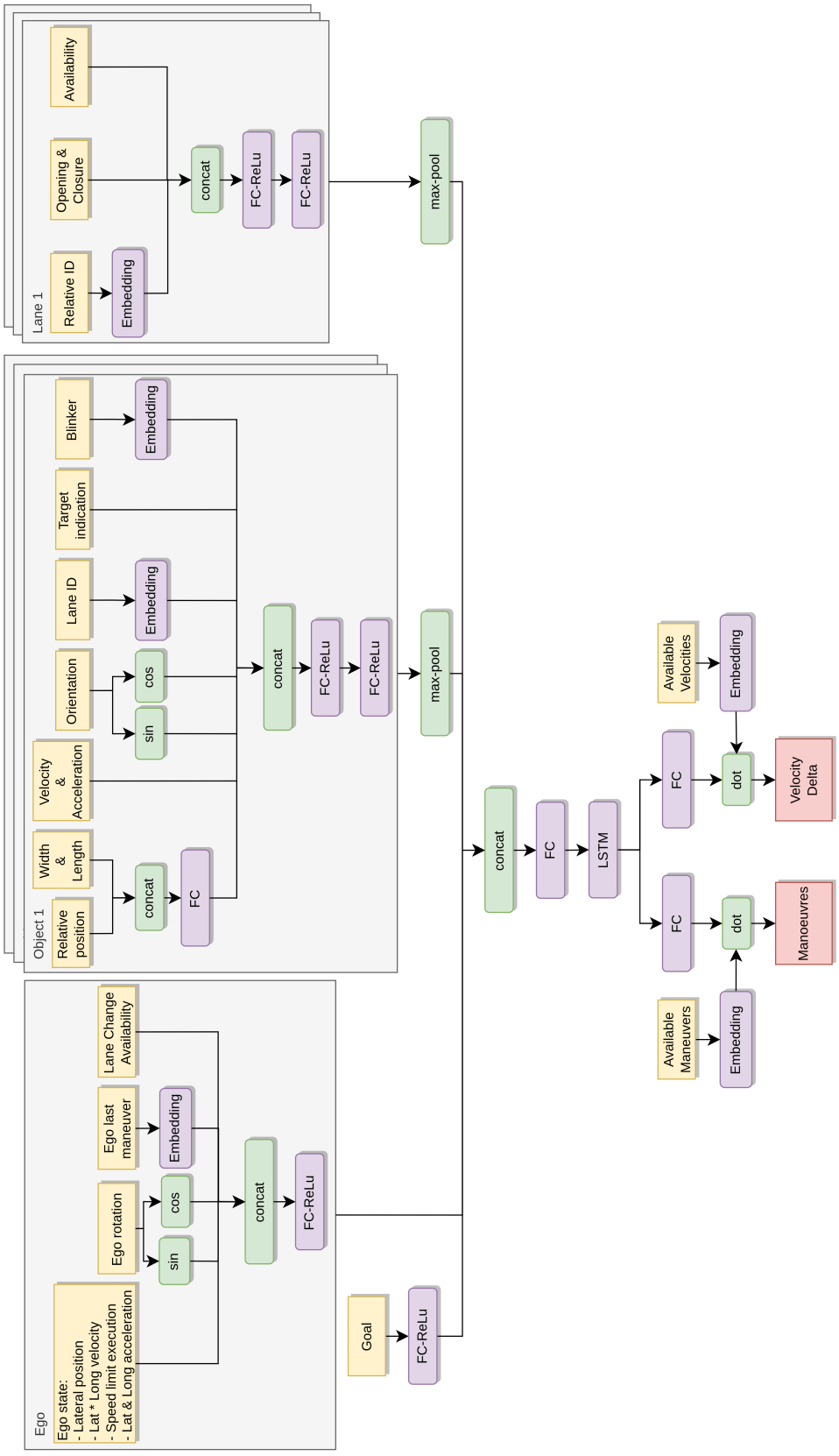


Figure 3.12: Neural Network architecture used in behaviour planning. Yellow blocks indicates inputs to neural network. Violet elements represents learnable parameters of the Neural Network (NN), while green blocks are deterministic, mathematical operations. Blue items represents data embedding along the processing, while red ones indicates the output from Neural Network (NN).

Table 3.4 Parameters of PPO algorithm used in behavior planning experiments.

Parameter	**Value**
train batch size	300,000
number of sgd iterations	5
discount parameter γ	0.995
GAE parameter λ	0.95
kl coefficient β	0.0
clipping parameter ϵ	0.2
gradient clipping	3.0
learning rate	5×10^{-5}

Table 3.5 Scenario distribution in the environment used both for training and evaluation of behaviour planner agents.

Scenario	**No Traffic**	**Light Traffic**	**Medium Traffic**	**Heavy Traffic**
scenario percentage [%]	10%	20%	40%	30%
congestion - time interval between new car spawning [s]	-	8 - 12 s	1 - 8 s	1 - 2 s

3.7 Results

To evaluate the mechanisms introduced, the agent responsible for behaviour generation was trained in two sets of conditions:

- **FSM ACC On** in which the FSM to control the execution of manoeuvres and suppress the selection of forbidden ones was used and with the ACC control on, which adapted the ego's speed in case of slower driving car in front.
- **FSM ACC Off** where the agent could freely select manoeuvres and the ACC control was off, therefore RL agent defines the set velocity for standard Cruise Control (CC).

In both sets of conditions, the reward function was defined in the same way and the emergency manoeuvre triggering was active. Both trainings were carried out in the same environments, where a straight road segment with a different number of lanes was simulated. In each episode, the goal lane has been generated at a distance between 100 and 1000 metres. Four scenarios with traffic alteration have been simulated, generated by providing the time range between the spawning of new agents at each lane origin (see Table 3.5 for more details).

Evaluation has been done in original setups, on randomly generated 500 scenarios. Evaluation in one additional setting was made as well:

- **FSM ACC Off in FSM ACC On Setup** where policy trained in FSM ACC Off setup was evaluated with settings from FSM ACC On configuration.

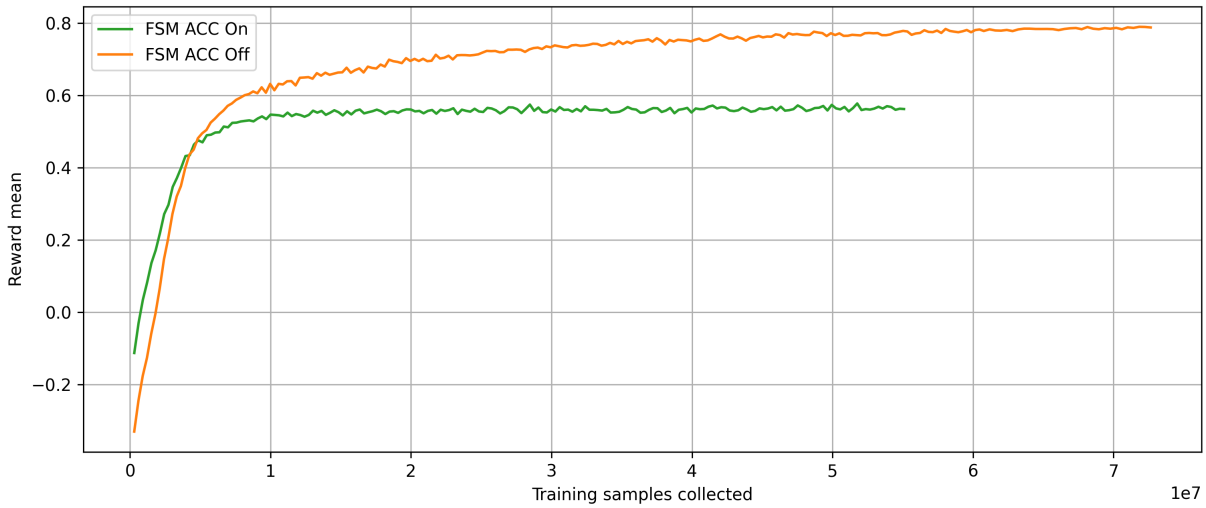


Figure 3.13: Mean reward plot of two training experiments.

By this evaluation, the check if mentioned mechanisms can be introduced after the training, and therefore filter unwanted actions from happening afterwards was performed.

In Figure 3.13 the mean reward scores (the average reward obtained by the agent along the training epochs) for two performed trainings are presented. Looking at the values of the reward function alone, the FSM ACC Off training ended with better performance.

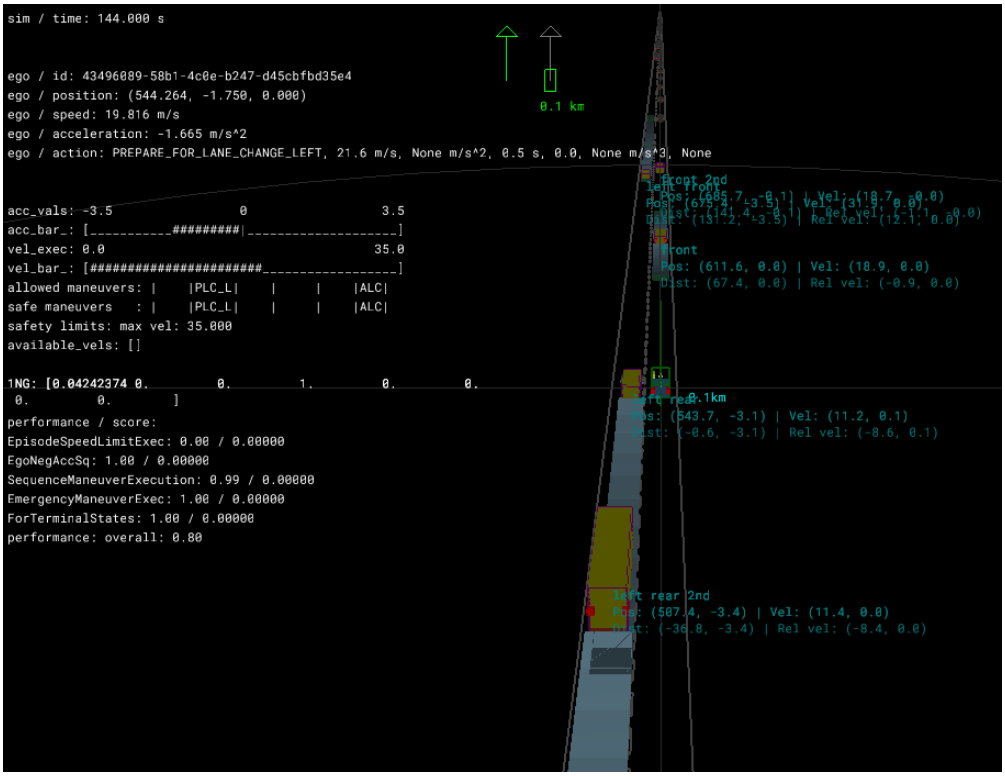
In Figures 3.14, 3.15, 3.16, and 3.17, examples of evaluations of the FSM ACC On agent have been presented. At the top, the green arrow indicates the target lane with a green rectangle showing the current position of the vehicle with the distance to the goal (similar to how most common navigation systems work). A set of metrics is displayed on the left side, with indication of ego current action (ego / action:) or currently allowed manoeuvres (allowed manoeuvres:).

To gain more insight into agent behaviour and performance differences, Table 3.6 presents numerical metrics that describe the agents' behaviour in the three experiments mentioned above.

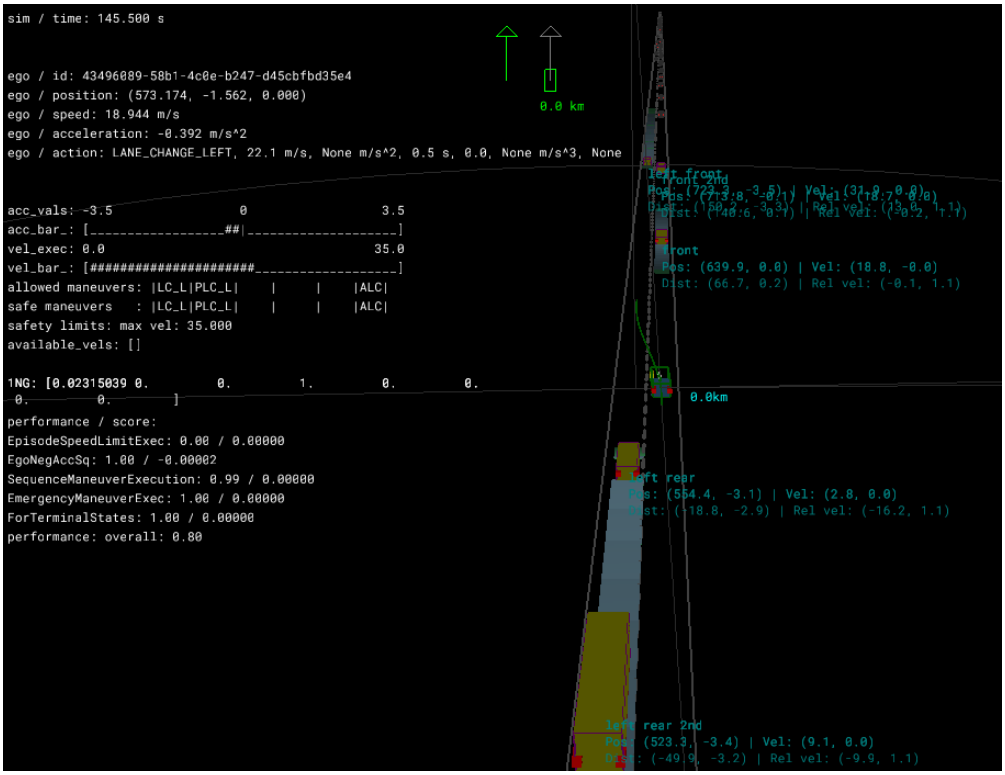
By careful analysis, the following conclusions might be drawn. From a goal-setting perspective, the FSM ACC On agent showed the best performance. It has also caused the least number of collisions. FSM ACC Off agent, which had more direct control over the agent speed, minimised the mean absolute acceleration and at the same time presented a slightly better mean velocity. The lower reward mean scores acquired by the FSM ACC On agent caused by the enforced FSM manoeuvre mechanism, which resulted in a higher penalty on manoeuvre execution. One also argue that the FSM mechanism improves the efficiency of lane change manoeuvres and stabilised the action selection process. A smaller amount of manoeuvre changes and less time spent in lane change, along with a higher probability of reaching the goal, support this claim.

Applying the mechanism after training, as in the FSM ACC Off in FSM ACC On experiment, has a negative effect on performance in most metrics. Therefore, one may argue that the introduction of the proposed deterministic mechanism during training improves general performance, although it must be done with care and the potential redesign of the reward functions.

Based on the metrics, potential problems have also been identified, which was most visible in the FSM ACC Off experiment. Looking at the manoeuvre distribution and visually reviewing agent performance, the

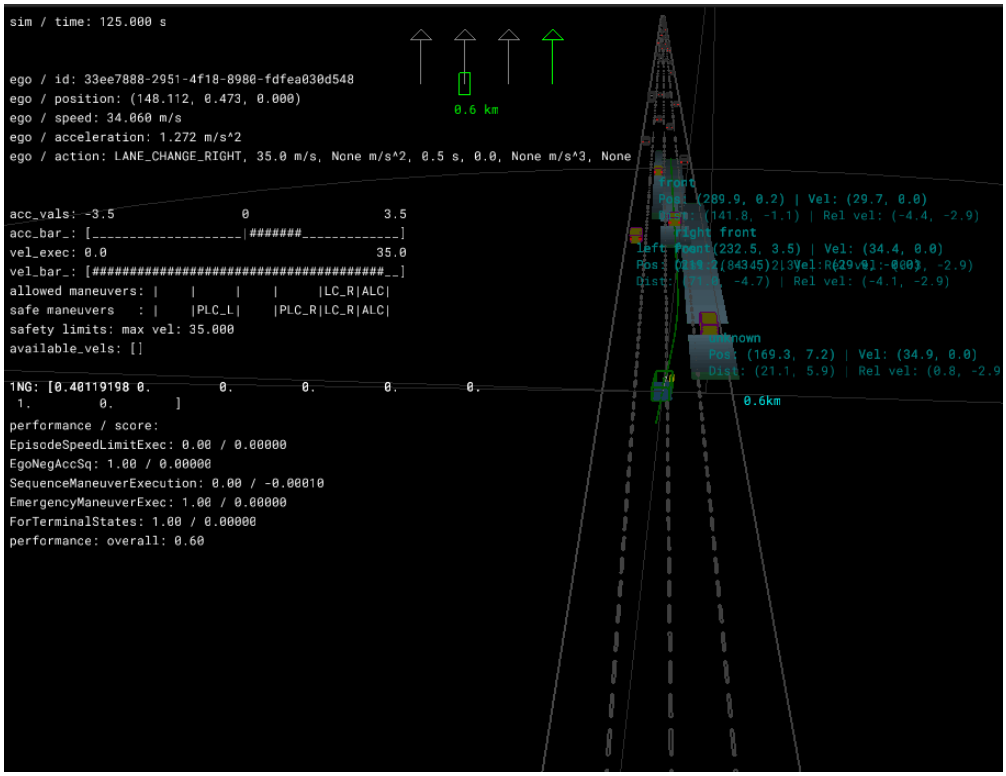


(a)

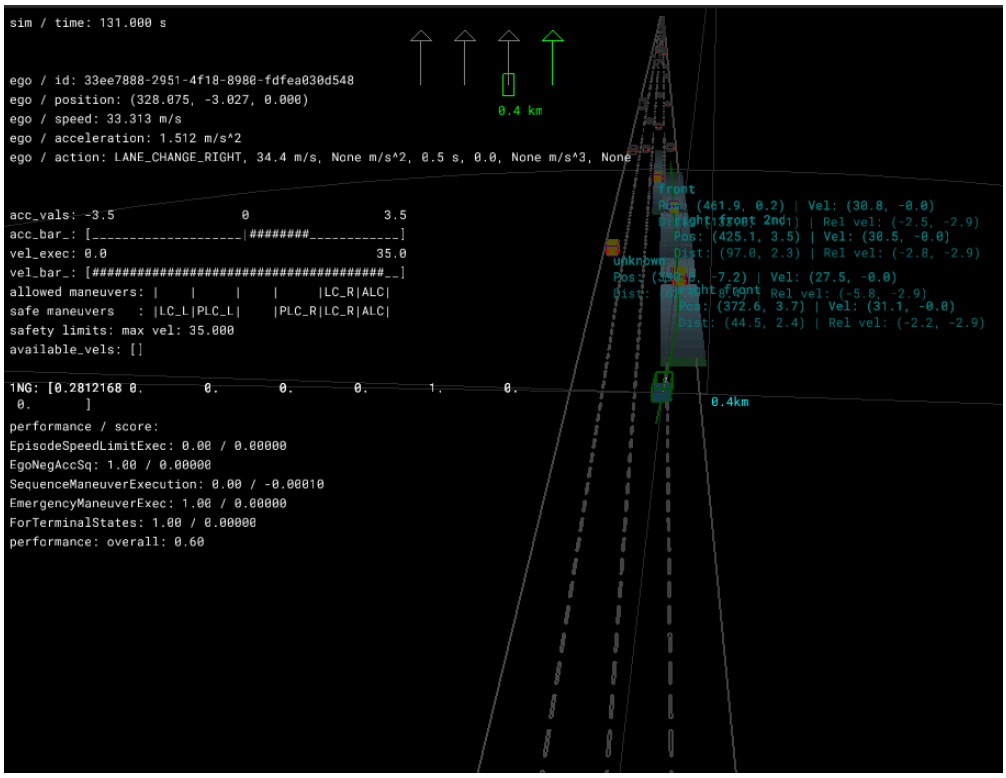


(b)

Figure 3.14: Scenario 1: The ego car equalises its speed with the left car (Subfigure (a)) to squeeze-in in front of it (Subfigure (b)).

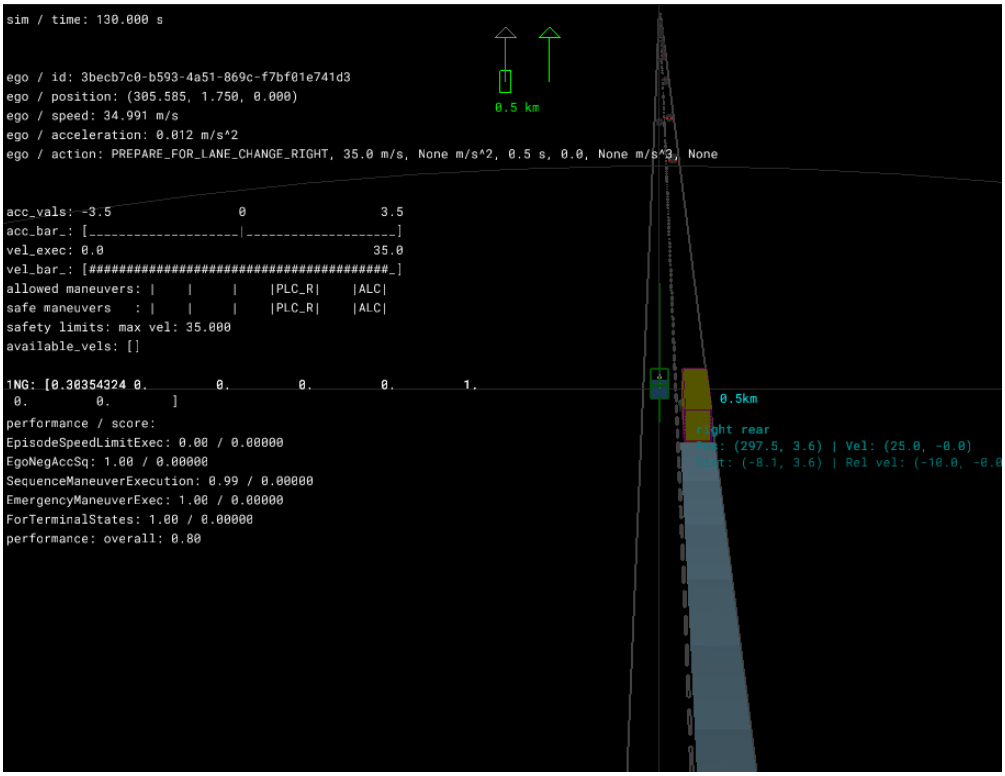


(a)

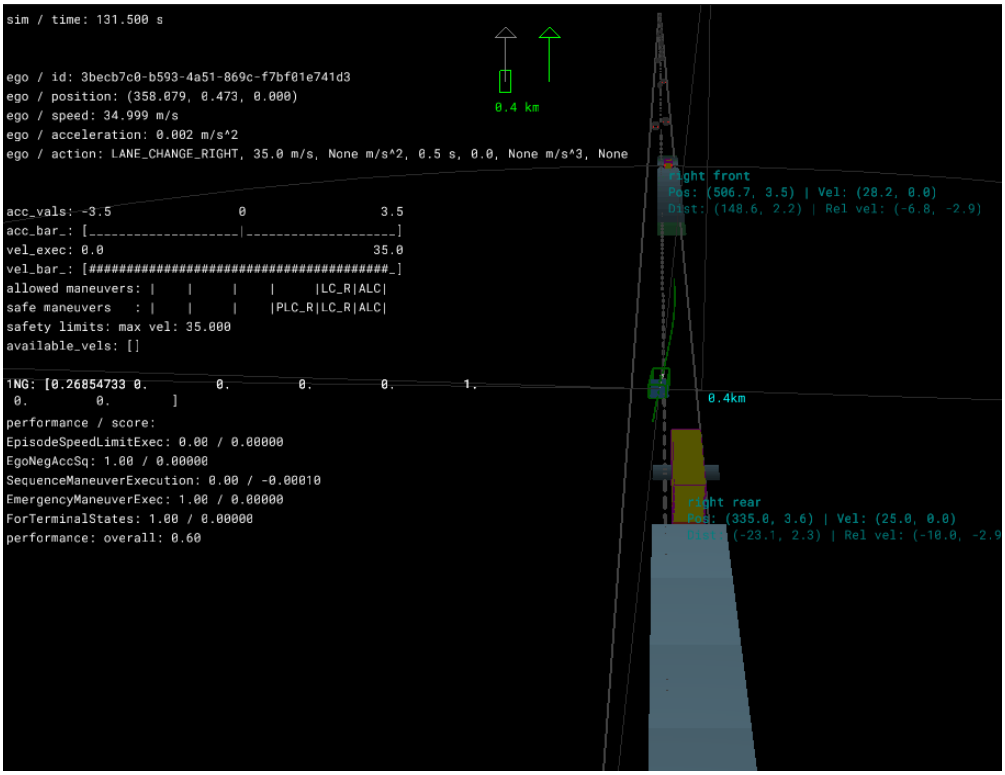


(b)

Figure 3.15: Scenario 2: Ego car must change lane twice to the right.



(a)



(b)

Figure 3.16: Scenario 3: Ego car keeps left lane and high speed to overtake a truck (Subfigure (a)) to later change lane to the right (Subfigure (b))

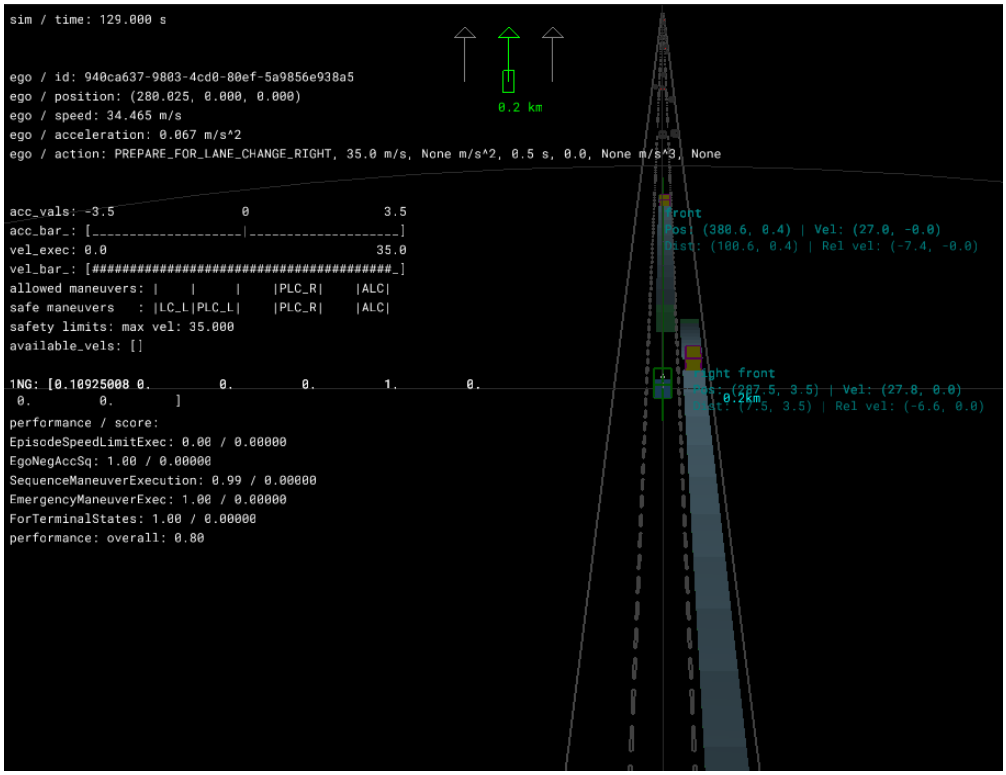


Figure 3.17: Scenario 4: Issue with realization of Prepare for Lane Change Right maneuver. Even thought agent does not want to change lane to the right (it is already on correct one), it continues to drive in wrong maneuver it until it reaches the goal.

Table 3.6 KPIs calculated for three experiments with a behavior planner agent.

Experiment Name	**FSM ACC On**	**FSM ACC Off**	**FSM ACC Off in FSM ACC On**
goal reached [%]	**99.2**	98.6	97.8
goal missed [%]	**0.4**	0.6	1.0
collision [%]	**0.4**	0.8	1.2
outside of road [%]	0.0	0.0	0.0
safety violation [%]	1.12	1.59	**0.85**
velocity mean [m/s]	27.21	**27.72**	26.03
velocity std [m/s]	4.55	3.435	4.58
acceleration mean $[m/s^2]$	1.33	**0.957**	1.3
acceleration std $[m/s^2]$	1.57	1.23	1.56
manoeuvre change count	2.16	4.02	1.82
follow lane [%]	58.7	24.5	59.6
prepare for lane change left [%]	12.1	1.8	4.33
prepare for lane change right [%]	25.0	63.4	33.2
lane change left [%]	2.34	4.74	1.60
lane change right [%]	1.81	5.6	1.04
abort lane change [%]	0.05	**0.0**	0.2

conclusion that the agent was often stuck in preparation for the lane change right manoeuvre was made, although still with the intention of just following the lane. The reason was that there were no functional differences in the execution of preparation for lane change and follow lane manoeuvres. As this is not the problem from end behaviour, it is raising issues from integration and explainability perspective.

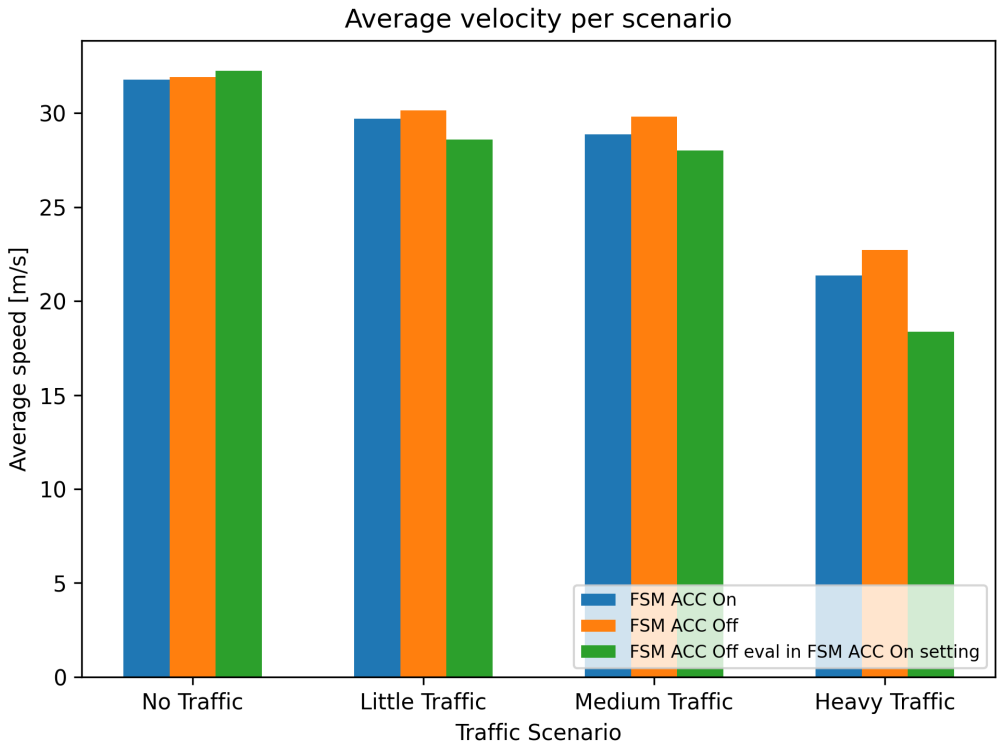
In Figure 3.18 two basic metrics, average speed and goal achievement, are presented, in different traffic scenarios. With this, one may argue that the heavier the traffic, the more deterministic rules had a stronger negative impact on the average speed, most probably caused by the lack of lane change rules enforced by the FSM formulation. It may be also stated that applying the rules afterward (FSM ACC Off eval in FSM ACC On setting) significantly reduces capabilities of the system in most congested scenarios.

3.8 Discussion and Further Work

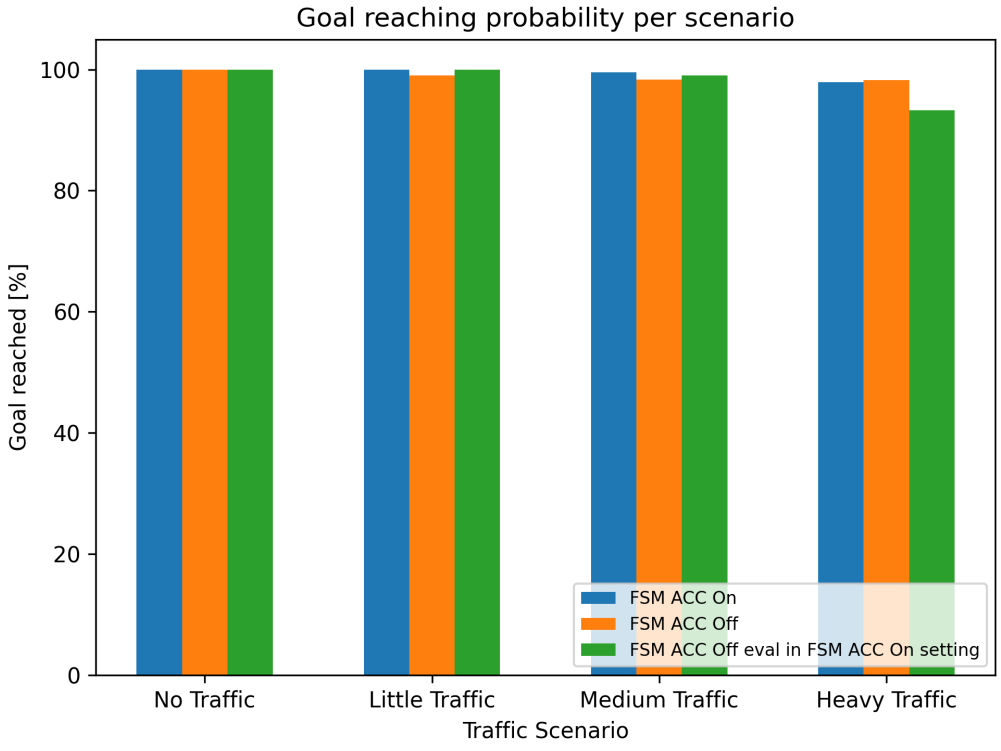
Analysis of results shows that reinforcement learning is capable of controlling agents with a high-level interface. Additionally, it was shown that the introduction of a deterministic mechanism at the time of training works better than introducing them afterward. Assuming that deterministic rules are required in the end, a recommendation of carefully introducing them during the training can be made.

One of the conclusions from the executed trainings and visual evaluation of the agent behaviour was the problem of fine-tuning the rewards. The more complex the reward function, the harder it was to control the agent behaviour by its changes. Methods which aims at estimating reward function with the use of other neural-network models, for instance supporting those actions which resemble previously seen behaviours, are attractive research direction and might help in defining the target behaviour system designer expects. Furthermore, it is recommended to build smaller modules with embedded Neural Network (NN) models is recommended, in which the desired behaviour can be defined more clearly.

The problem of merging experience-driven behaviour with traffic rules should still be treated as open. Although in the research presented an effort was made to integrate some of the traffic rules within the system, real-world scenarios will require a much more complex and elaborate set of rules, which will pose new challenges during integration of those with planning algorithms. Recent development efforts from MobilEye and Tesla, where planning was done at the level of deciding interactions between road users, seems to be a preferred way to solve these issues.



(a)



(b)

Figure 3.18: The most important metrics that describe the behaviour of the agent, presented in different traffic scenarios.

Chapter 4

Parking

4.1 Introduction

Another essential element in the desired portfolio of automated driving skills is parking a car. As each journey starts and finishes with such a manoeuvre, it is desirable to perform it without a driver's involvement, especially as many drivers struggle with more complex manoeuvres and tight parking spaces.

Manoeuvring in a parking environment is quite different from driving on a highway. First, it involves much more free-movement planning, which is less constrained by arbitrary rules or a portfolio of manoeuvres to execute. This means that there are fewer constraints that the agent should (or must) follow, giving him more freedom in planning. This results in a more clearly defined reward function, which does not have to be described as a composition of many objectives. Secondly, car movement is much more driven by its kinematics, and its precise dimensions play an essential role in manoeuvring in tight spaces. Next, assuming that no other actors are present in the scene (or their presence is handled outside of parking module), the environment dynamics depends purely on agent movement and scene perception, making it more accordant with Markov property. Finally, parking involves low-speed manoeuvring, resulting in less severe consequences of eventual error. This plays an important factor in legislation and the easier path to commercialising given solutions.

Taking all of the above features into account, it seems that reinforcement learning methods are a good fit for parking scenarios. With a much clearer objective, a less complex environment with fewer deterministic rules to follow, and more combinatorial planning, it looks like RL-based algorithms might result in better performance and less computation power than standard methods.

4.2 Problem Formulation and Assumptions

4.2.1 Problem Formulation

The focus of the following experiments is to derive the control policy (resulting in a path) for the parking slot planning problem, including three scenarios: perpendicular parking, parking at an angle and parallel parking. The trained RL policy will be used to acquire the path based on iterative neural network inference.

The problem is framed as POMDP, although in the training process the partial observability feature is not addressed in any specific way. At the start of each episode, the environment will generate an instance of scenario, including the initial position of the ego vehicle, the details of the goal, and the set of obstacles. The process of scenario generation is controlled by environment configuration, which might change during training to present the agent with scenarios of different difficulty. The agent is identified with the ego vehicle. The agent policy, based on observation of the ego's relative position to the goal and obstacles, will derive action, which includes movement and turn angle. In response, the environment will simulate the movement of the ego car, check for any potential collisions with obstacles, and verify if the goal position has been reached. From there, the environment is able to provide a new set of observations for the agent. By the iterative process of action selection and environment simulation, the path is created, which, in positive case, leads to the goal position. The event of collision, goal reaching, or exceeding the maximum number of simulation steps results in episode termination. The agent is rewarded in the sparse manner at the end of an episode, with positive reinforcement for goal reaching and zero otherwise. By reward design, the agent is also encouraged to minimise the number of direction changes.

PPO algorithm will be used to train the policy. Comparison of two distinct neural network architectures that define the agent policy by comparing its functional performance, robustness, and execution times will be made. Additionally, an evaluation of the proposed solution will be presented based on real world data.

4.2.2 Assumptions and Limitation

During parking experiments, the following limitations have been accepted and specific assumptions have been made, listed below:

- Vehicle movement will be simulated with planar kinematic model. It might be argued that there is no need to simulate dynamic part, as the path planning problem should not be concerned with that aspect by operating in very limited dynamic constraints. Dynamic aspects can be addressed later in the control process.

- Both the ego vehicle and obstacles are represented in a form of polygons, which might potentially limit the way of representation of the scene. One may argue, however, that most of the scenarios might be well modelled with polygon representation, therefore this limitation should not be seen as severe obstacle in application trials.

- Perfect knowledge about the environment is assumed within the constraints of defined observation interface, without noise modelling in a space of observation creation or motion execution. This may limit potential transfer to real-world applications, however resulting solution should be efficient enough to be able to re-plan the path during maneuver execution, therefore adjust in case of discovered perception issue later.

4.3 Prior Art

Planning a motion for the autonomous vehicle during parking manoeuvres in a much less structured environment than highway scenarios requires the use of targeted methods. One of the first successful implementations of autonomous driving systems operating in urban and parking domains was presented during the 2007 DARPA challenge. In one of them, a variant of the A* algorithm and non-linear numeric optimisation have been combined in a two-phase planning algorithm [35]. A similar hierarchical approach with the use of optimisation-based collision avoidance has been described in [167]. Another hierarchical system, consisting of an imaginative model for anticipating results before parking, an improved rapid-exploring random tree (RTT) for planning a trajectory and a path smoothing module, has been introduced in [36]. An extended version of the RTT mechanism, the bidirectional rapidly-exploring random trees, aiming at improving the consistency and quality of the generated path coupled with the parking-orientated model predictive controller is described in [51]. In [21], the authors introduced a method called Orientation-Aware Space Exploration Guided Heuristic Search, allowing to gather knowledge about driving direction, which might be later used as a heuristic in the search phase of planning manoeuvres. The authors of [120] introduce the system for coordinated parking of vehicle fleets, consisting of centralised spot allocation and path planning executed by a so-called coordinator and decentralised collision avoidance performed by each vehicle. A distributed system for valet parking in multi-story parking connected with charging the vehicle phase has been summarized in [58]. The practical application of visual SLAM has been presented in [137] which has been used for the trained parking application, allowing a car to park itself in known, frequently visited locations. In [121], the neural network methodology has been used to predict the intent and movement of other road users in a parking lot, which could be crucial to planning the movement of the ego itself.

Reinforcement learning methodologies have also been successfully used in parking domains. An end-to-end solution, where reinforcement learning agents directly controlled the angle of the steering wheel along with the model-based tracking of the parking slot has been described in [166]. In [11], the deep deterministic policy gradient algorithm has been used to train a reverse parking manoeuvre. In [125], the successful application of the DDPG training algorithm for parking vehicles in tree scenarios (parallel, perpendicular, and at an angle) has been presented. The use of a multi-agent training algorithm for the problem of online parking assignment, operating with both connected and non-connected cars, has been proposed in [168].

4.4 Parking Slot Environment

For the training of agents in a parking environment, a parking simulation has been implemented in the Python programming language. The main elements of it involved an ego car whose movement could be controlled by defined action, a set of obstacles, each represented as a polygon, and the goal, defined as desired position and orientation. On the basis of that, the mechanisms defining the dynamic relationships were implemented, including car movement simulation, collision detection, and reaching the goal by ego. In the end, the rendering option was added allowing for visual inspection of the given scenario and the performance of the ego car. Having such simulation in place, specific functionalities for reinforcement learning methodol-

ogy were added, such as observation creation mechanism, reward definition, and termination check. The elements listed above are described in greater detail below.

4.4.1 Environment Class Structure

To ease further experiments definition and benefit from learning lessons while implementing the TrafficAI environment (Section 3.4.2), the decision has been made to simplify its general design. As design dependent on pipelines is proven to be very customisable and scalable, it turned out to be hard to debug and test. At first glance, it was often not clear which elements of the pipeline change which parts of states and the overflow of information was not well presented. To improve this, parking experiments used the general environment class, which was the composition of other predefined modules, however, which could be shared among different experiments.

The new definition of the environment already assumed the possibility of simulation of multiple agents at once. To start with, each environment has to define the State in the form of the data class, with predefined slots for general and agent-specific data, both for active and terminated agents. General State introduced as well placeholders for common data, such as agent id, done flag and action. To initialise the state, the class Scenario was introduced to define the given configuration, which should deterministically define a given variation of a use case. With that, all interaction between each agent would have to be defined in the EnvDynamics class, in the form of a step function. Here, all changes and evolution of the state have been implemented. This class is also responsible for initialising state based on scenario, raw action decoding, and checking the terminal condition of each agent. To define the way of representing the State to the agent user had to implement class Observer, parsing State to vector representation consumable by neural network along with observation space definition. The Critic part was responsible for calculating the reward function based on State, while the Viewer task was rendering the current scene. With this design, all relations between elements of Environment were clear, and the code was quite reusable. It allowed freedom of definition, which was often necessary to implement specific mechanisms.

Based on the general template, the ParkingSlotEnv has been created, implementing all elements accordingly to the parking use case. Specific elements of that environment are described in the following sections.

4.4.2 Path Planning Ego Motion Model

Planning parking manoeuvres can be performed with the use of different interfaces and motion models. One of the most common motion models used in many robotic applications is the bicycle model [102], where the state is defined as position, velocity and optionally higher-order derivatives like acceleration. The control interface most commonly includes acceleration and steering angle. In the parking use case, however, the eventual interaction with other movable objects is often handled externally, while manoeuvres can be executed with different velocity profiles. Because of that, tracking the dynamic state (such as velocity) is not especially necessary at the level of planning parking manoeuvres. Planning a path (list of reference points), coherent with the kinematic model constraints, should be the goal of planning, allowing further modules to plan specific execution of it in time.

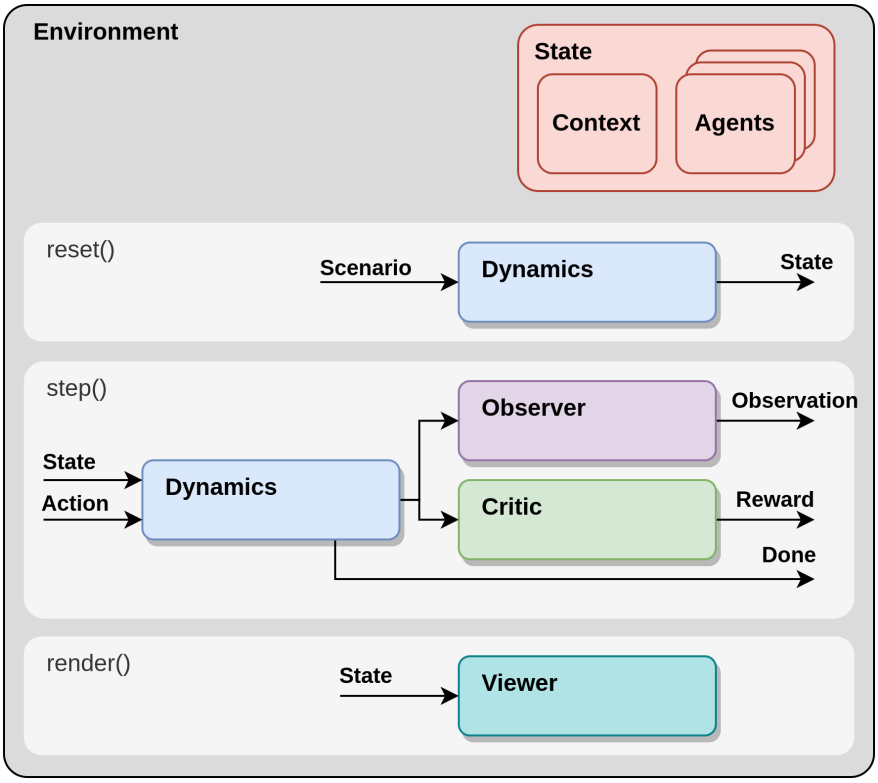


Figure 4.1: Environment template with three interface methods: reset, step and render

Having understood this, a decision has been made that in the case of this experiment, the parking module will output the path. Its shape will be the result of a series of interactions between the agent and the environment, which, in other words, would be the trajectory of actions taken during a single episode. As the scene did not include any other moving objects, the problem might be treated as one where the dynamic function is known, which allows for precise and repeatable simulation of environment dynamics. At each step, the agent is asked to select the action from the discrete space, where each action is a combination of a specific wheel angle and a travel distance. Then, the simulation of the movement of the ego is executed and the agent based on the observation (described in Section 4.5) could select the next action.

The motion model used for the simulation uses a control vector consisting of a turn angle value and distance to travel. The model has been based on the formulation of the bicycle model [70], while detailed equations of motion have been derived based on geometric analysis.

The model of a car is presented in Figure 4.2. First, based on the wheelbase of the car L and the angle of rotation of the wheels δ one may derive the turn radius r, which is equal to

$$r = L \tan\left(90^\circ - \delta\right) = \frac{L}{\tan \delta}. \tag{4.1}$$

When the turn radius has been established, by using the formula for the arc length,

$$s = r\Delta\psi, \tag{4.2}$$

the delta in the orientation of the car $\Delta\psi$ can be calculated as

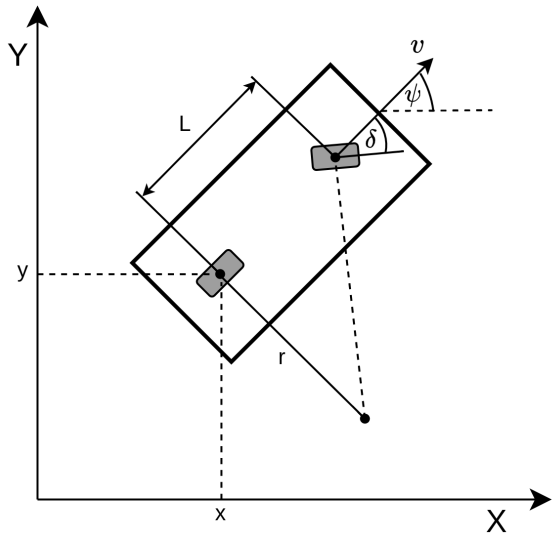


Figure 4.2: Graphical representation of kinematic motion model used for simulation.

$$\Delta\psi = \frac{s}{r} = \frac{s_t \tan\delta_t}{L}. \tag{4.3}$$

and use it in the final motion equation (4.8).

The movement of the car on the X and Y axes is based on translation by the vector $\vec{p}$. The angle of the vector can be derived as the average orientation angle ψ^{avg}

$$\psi^{\text{avg}} = \frac{\psi_t + \psi_{t+1}}{2} = \psi_t + \frac{\Delta\psi}{2} = \psi + \frac{s_t \tan\delta_t}{2L}, \tag{4.4}$$

while its length can be derived by analysing the geometric relationship in an isosceles triangle (see Figure 4.3 for reference).

$$|\vec{p}| = \frac{2L}{\tan\delta_t} \sin\left(\frac{s_t \tan\delta_t}{2L}\right) \tag{4.5}$$

Finally, a set of equations that describe the car motion can be represented as

$$x_{t+1} = x_t + |\vec{p}| \cos\psi^{\text{avg}} = x_t + \frac{2L}{\tan\delta_t} \sin\left(\frac{s_t \tan\delta_t}{2L}\right) \cos\left(\psi + \frac{s_t \tan\delta_t}{2L}\right) \tag{4.6}$$

$$y_{t+1} = x_t + |\vec{p}| \sin\psi^{\text{avg}} = y_t + \frac{2L}{\tan\delta_t} \sin\left(\frac{s_t \tan\delta_t}{2L}\right) \sin\left(\psi + \frac{s_t \tan\delta_t}{2L}\right) \tag{4.7}$$

$$\psi_{t+1} = \psi_t + \frac{s_t \tan\delta_t}{L} \tag{4.8}$$

with control variables, including distance s to move along the curve and wheels' angle δ. The system is also parameterized by car dimensions and state variables, where x and y are ego positions in the two-dimensional plane, ψ is its orientation, and L is a wheelbase. The distance s_t could be both positive and negative, depending on the direction of movement. The motion model takes a simplified form when the turn angle δ equals 0 (the car is moving straight).

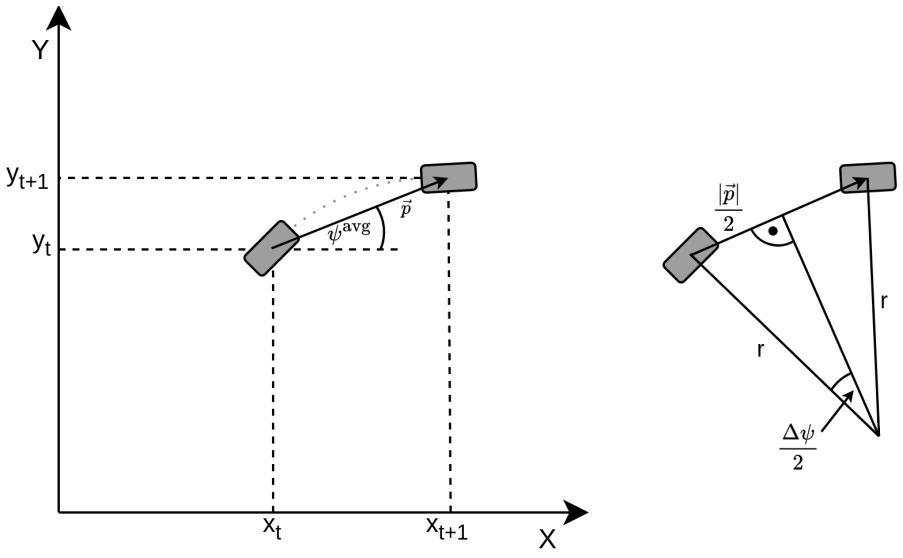


Figure 4.3: Graphical representation of simulated movement of a imaginary rear wheel of bicycle model from time t to time $t + 1$

With this model in mind, the action space has been designed as cross-product of movements provided in metres and wheel angles provided in degrees:

$$\mathcal{A} = \mathcal{A}^s \times \mathcal{A}^\delta, \tag{4.9}$$

where

$$\mathcal{A}^s = \{-1.0, -0.25, 0.25, 1.0\}, \tag{4.10}$$

$$\mathcal{A}^\delta = \{-28, -24, ..., 24, 28\}. \tag{4.11}$$

4.4.3 Obstacles and Collision Detection Mechanism

To define the parking scenario, except for the ego itself and the goal, obstacles are added to create a natural representation of a parking lot. Obstacles should represent both parked cars and elements of infrastructure, such as kerbs or barriers. They are defined as polygons with a potentially unlimited number of vertices.

Then, the collision detection mechanism has been implemented. The ego was represented as a polygon with vertices defined as its four corners. The general idea of this module was to cross-check if any vertex of the first polygon lies within the second polygon, and vice versa. To check if a given point lies within the polygon, the cross-product between each vector defining the border of the polygon and a vector going from each vertex of the polygon to the point under test has been calculated. In case when all resulting cross-products have the same sign, the point lies within the polygon. A graphical representation of the method is presented in Figure 4.4.

The described method has some limitations, including the requirement that the obstacles (polygons) have to be convex. Another one is the potential failure of collision detection in cases where no points of polygons

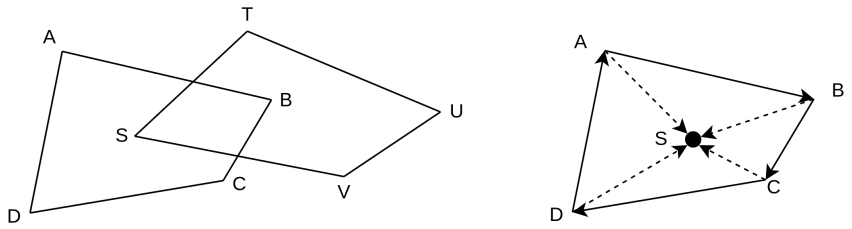
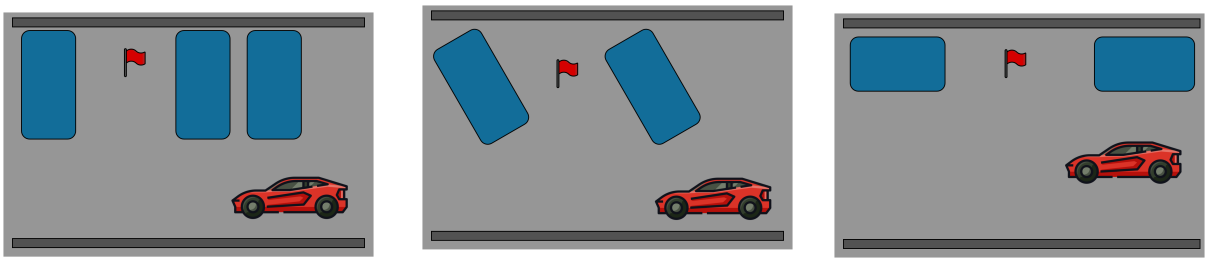


Figure 4.4: To check whether any polygons are in a collision, cross-check is done if any vertex of polygon $ABCD$ lies within polygon $STUV$, and vice-versa. To do so, for each point the cross-product between border vectors and vectors going from the origin of the border vector to the point under test is calculated. In the case presented here, it is done such for pairs $\vec{AB} - \vec{AS}$, $\vec{BC} - \vec{BS}$, and so on. If the sign of all cross-products is the same it means that the point S lies within the polygon $ABCD$.



(a) (b) (c)

Figure 4.5: Three families of parking scenarios. In Figure (a) represents perpendicular parking, where the ego has to park nose-in or rear-in. Figure (b) shows the parking at an angle, used when the road is narrow and perpendicular parking would be difficult. Lastly, in (c), the parallel parking case is presented, which is useful in most narrow streets. Scenarios vary in detail, such as the amount of space, the initial position of the ego vehicle, etc.

lie within the other one, but still its borders cross, which might happen in cases of "very pointy" polygons. However, those limitations, keeping in mind the simulated scenarios and a rather small ego's move distance, do not have a negative effect on function and training itself. Additionally, to further improve the safety of collision detection, the middle points of each obstacle have been added to cross-check the procedure.

4.4.4 Parking Scenarios

Having basic structures ready, the scenario can be defined which in the case of the formulation of the parking problem is a variation of the situation (functional use case) that the agent has to solve during the episode. The scenario has been encoded with a set of parameters, listed in Table 4.1.

This definition has been used to represent all three families of use cases: perpendicular parking, parking at an angle, and parallel parking, presented in Figure 4.13.

Table 4.1 Structure of scenario class, based on which episodes in parking environment are created.

Values	**Description**
Initial position	initial position of Ego car
x_0	initial position on the x axis
y_0	initial position on the y axis
ψ_0	initial orientation
Car parameters	parameters of the ego vehicle used for simulation of kinematic motion
L	wheelbase; distance between two axles
width	width of ego vehicle
length	length of ego vehicle
$\delta_{\max}$	absolute maximum angle of the front wheels
axle to front	distance from rear axle to front of the vehicle
axle to rear	distance from the rear to rear axle
Goal	goal description
x_g	goal position on the x axis
y_g	goal position on the y axis
ψ_g	goal orientation
position tolerance	positional tolerance between the goal orientation and ego orientation to reach the goal
orientation tolerance	orientation tolerance between goal orientation and ego orientation to reach the goal
Obstacles	list of obstacles present in the scene, each represented as a list of points creating them
Points	list of points defining each obstacle
x_j	position of j-th obstacle point in x axes
y_j	position of j-th obstacle point in y axes

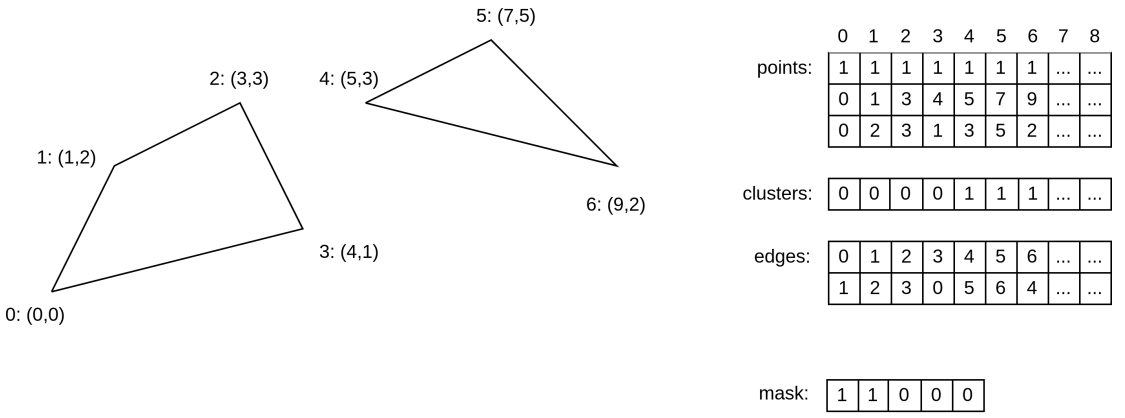


Figure 4.6: Sample of graph objects and their observation for graph neural network used in parking application.

4.5 Observation Space and Corresponding Neural Network Design

During the experiments with the parking environment, multiple ways of presenting the scene to the agent and the results of different network designs were tested. Initial, simplistic experiments with encoding the goal and specific points of obstacles defining the parking spot in a form of a flat vector with standard fully-connected layers failed. The most probable reason for that was the imposed ordering of points and because of the lack of generalisation while the agent was moving through the scene.

In the end, reasonable performance of the agent with two designs was achieved. The first one has been encoding the obstacles present in the scene as graphs, where later the Graph Neural Network has been utilised to process this input. The second one utilised the idea of LIDAR-like observation, where multiple rays go from the ego to the first encountered obstacles, measuring free space. Both approaches are described in detail below.

4.5.1 Graph Representation of the Scene

Most of the elements within the parking environment have been defined as polygons or might be interpreted as ones (like the shape of the ego). Because of that, the natural approach would be to observe those elements in their original form. However, this requires the use of the specific network architecture, allowing a generalised way of processing, independent of input ordering and the number of points to process.

As a result of selecting a Graph Neural Network as agent architecture, observation space has to be designed specifically for this design choice. Figure 4.6 presents a sample encoding of the objects, while Table 4.2 presents the structure of observation used in this experiment. First, all nodes of polygons in the scene are encoded as a long list of points with specific qualities such as type and position. The cluster vector indicates which nodes belong to which cluster, while the edges inform about interconnections between the nodes. Finally, the mask vector indicates the valid clusters, which are later processed with a multi-head attention mechanism.

Table 4.2 Structure of parking observation when using graph neural network.

Values	Space	Range	Description
context			contextual information about the goal and the ego vehicle
$\psi_{\mathrm{g}}^{\sin}$		$< -1, 1 >$	sinus value of the orientation of the goal in VCS
$\psi_{\mathrm{g}}^{\cos}$		$< -1, 1 >$	cosine value of the orientation of the goal in VCS
x_g		$< -1, 1 >$	relative x position of the goal in VCS
y_g		$< -1, 1 >$	relative x position of the goal in VCS
nodes	[24, 3]		list of nodes descriptions
node	[3]		single node description
x_k		$< -1, 1 >$	x position of the k-th node in VCS
y_k		$< -1, 1 >$	y position of the k-th node in VCS
t_k		$\{1, 2, 3\}$	type of nodes
edges indexes	[24, 2]		list of pairs of indexes indicating connection between nodes
edge indexes	[2]		single pair of indexes
start idx		$\{1, 2, ..., 24\}$	index of the start node for the edge
end idx		$\{1, 2, ..., 24\}$	index of the end node for edge
clusters	[24]		association of nodes with the clusters
c_k		$\{1, 2, ..., 6\}$	cluster id of k-th node
mask	[6]		
v_i		$\{0, 1\}$	validity of i-th cluster

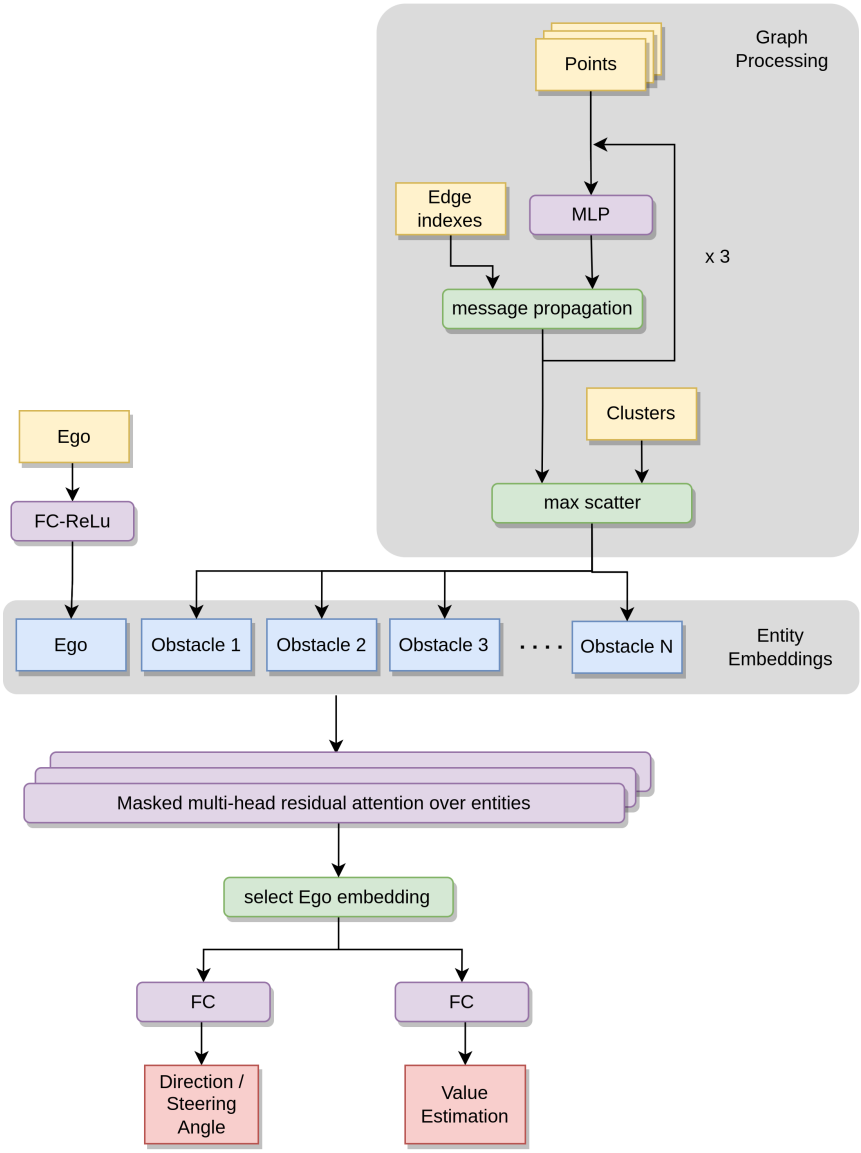


Figure 4.7: Architecture of graph neural network based policy. Color scheme follow the one introduced in Figure 3.12.

Based on such a prepared observation, an architecture based on graph NN was applied (Figure 4.7). First, the ego data along with the description of the goal is embedded as the first token, with the use of a single FC layer with ReLU activation function. Tokens describing obstacles are embedded with Multilayer Perceptron (MLP) and graph-based processing, which concatenates embeddings of points according to edge indexes in a loop (in case of neural network architecture used within 3 iterations), allowing to encode longer sequences of polygon vertexes with each iteration. Later, with the use of max scatter operation, embeddings are squeezed to corresponding obstacles embedding with the use of cluster indexing. Next, all entities (ego and obstacles) are processed with masked multi-head attention, allowing to mask of not valid obstacles. Based on the embedding of the ego after processing through the attention mechanism, the action distribution and the value estimation are calculated by the FC layers.

Table 4.3 Freespace observation. All values are normalized.

Values	Space	Range	Description
context	[4]		contextual information about the goal and the ego vehicle
$\psi_{\mathrm{g}}^{\sin}$		$< -1, 1 >$	sinus value of the orientation of the goal in VCS
$\psi_{\mathrm{g}}^{\cos}$		$< -1, 1 >$	cosine value of the orientation of the goal in VCS
x_{g}		$< -1, 1 >$	relative x position of the goal in VCS
y_{g}		$< -1, 1 >$	relative x position of the goal in VCS
freespace	[50]		freespace described as a set of distance measurements around the car
d_i		$< 0, 1 >$	freespace at i-th azimuth to closest obstacle or other agent; saturated at 50 meters

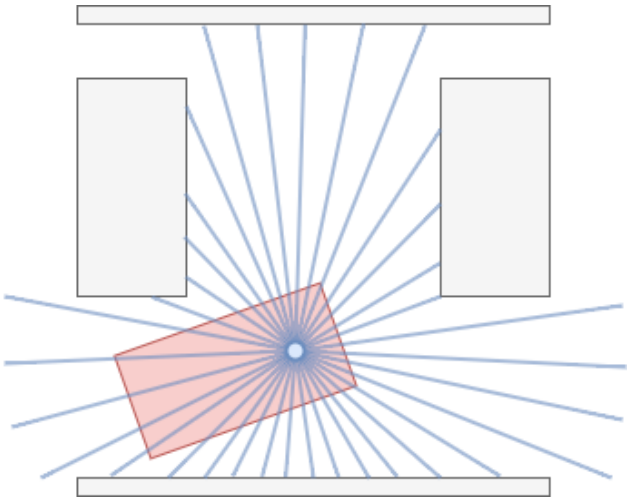


Figure 4.8: Visualisation of freespace measurement. Ego car is marked as red rectangle, with blue freespace rays, casted from its middle at evenly spread angles. The rays report distance to closest obstacles (grey objects) at given azimuth. For sake of clarity, smaller number of rays is presented in the image than in experiments.

4.5.2 Free Space Observation

As parking involves manoeuvring closely to obstacles, the sense of free space around the car is important information while planning the motion. As graph neural networks are good at representing general information, such information might be provided to an agent in a simpler form.

Taking inspiration from [7], LIDAR-like distance measurements were used, originating from the centre of the rear axis with the spread of angles around the car. The measured distance is saturated to a predefined value (see Figure 4.8 for a graphical representation of the measurement). In the experiments, 50 evenly spread rays around the car were used. Additionally, information about the goal's position and orientation (in the form of sine and cosine of angle difference) relative to the ego was provided.

Based on that, a straightforward neural network architecture has been proposed, mainly with the aim of high throughput and sample efficiency during training, presented in Figure 4.9. The architecture included basic, one-layer circular convolutional (Conv) processing of the ray measurement. Later, the output from

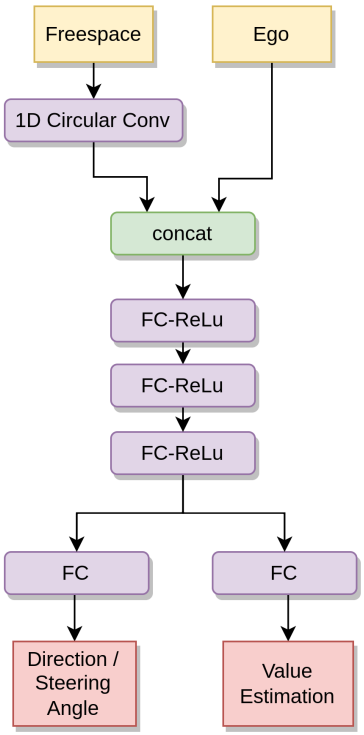


Figure 4.9: Architecture of neural network consuming freespace rays as primary information about surrounding obstacles. The color coding is the same as in Figure 3.12.

convolution is concatenated with context information about the goal, and processed with a multi-layer fully-connected block. In the end, action distribution and value function are calculated respectively.

4.6 Reward Design

Parking a car is clearly an episodic use case, as reaching the destination spot is a straightforward definition of termination. In the parking spot configuration, the decision has been made to use purely sparse rewards. Most of the time, the reward was defined in the same way, except for some experiments in the initial training phases. The agent has been receiving a +1 reward when reaching the goal and 0 otherwise. To minimise the count of direction changes, each such event has been punished with a -0.1 reward and added to the final reward, but only after successfully reaching the goal. In this case, the reward has been set to a minimum of 0.3, to ensure an indication of reaching the goal, even in the case of multiple direction changes along the way. The reward function has been presented in equation 4.12, with n^{dir} standing for the number of direction changes.

$$r(s,a,s') = \begin{cases} \max\left(1 - 0.1 \times n^{\text{dir}}, 0.3\right), & \text{if goal is reached} \\ 0, & \text{otherwise} \end{cases} \tag{4.12}$$

Initial experiments have shown that initial training phases often suffered from weak positive reinforcement. The successful start of the training procedure was based on the correspondence of the magnitude of the reward for success, the discount factor γ, and the qualities of the scenarios, defining the number of suc-

cessful episodes with the random policy in the early stage. The conclusion was made that reward shaping in the early phases of training may ease this initial struggle. The tried changes, as well as other ways of dealing with early training, have been presented in Sections 4.7.1 and 4.7.2.

4.7 Policy Optimisation

For training, most of the experiments used the PPO algorithm [115].

The successful application of reinforcement learning methodologies to listed environments was not straightforward and required a bit of work and customisation. Most of them dealt with the initial phases of training and the ability to back-propagate positive, sparse rewards for early-stage random policy.

4.7.1 Initial Phases of Training

In the early stages of training, with randomly initialised weights of the policy, agents were rarely able to reach the target position successfully, resulting in weak positive reinforcement. With the initial reward definition, the collision of the agent has been associated with a negative reward of -1, which dominated the episodes with an untrained policy. One way of overcoming this issue was to eliminate negative reward (-1) in case of collision. With this design, the agent was only rewarded in non-zero manner for arriving at the goal position, which did not discourage further exploration.

Another way that has been tried and worked well was by crafting scenarios to increase the probability of reaching the goal successfully, even with a random policy. That was achieved by moving the agent closer to the goal, relaxing goals' tolerances, and at the same time moving obstacles away.

The last way of dealing with the initial phases of training, which played as well the role of continuous difficulty increase accordingly to agent performance, was the introduction of curriculum, described in the next section.

4.7.2 Curriculum to the Rescue

As humans learn a new skill, one of the efficient training methodologies is to start with a relatively easy task and then gradually increase its difficulty, with a rate adequate to the student's current competency. With such an approach, a student is always challenged with tasks at the edge of his capability, allowing him to perform relatively well based on his current skill set and at the same time learn something new. As it turns out, an analogous approach, called curriculum learning, is similarly effective in artificial neural network training, resulting in faster convergence or in some cases even enabling it.

A good overview of older curriculum methods has been presented in [14], while [89] surveyed curriculum methods designed specifically for reinforcement learning domains. The implemented solution used in this research resembles most of the solutions presented in [10, 148].

In experiments, the approach of a teacher-guided curriculum was followed. The student, associated with the parking agent, is presented with tasks coming from the scenario generator. The curriculum is realised by controlling the scenario generator to produce tasks at the appropriate level of difficulty. Algorithm 3 represents the methodology to control the difficulty of scenarios.

Algorithm 3 Parking curriculum

Get initial scenarios generators configurations c

$c_0 \leftarrow c$

Set scenario difficulty update value Δc

Set minimum performance to update difficulty $r^{\min}$

Set threshold value for not decreasing difficulty ϵ

for iteration $i = 1, 2, ...$ **do**

  Collect set of experiences D_i with scenario generations configuration c_i

  Optimize the policy π by running PPO algorithm

  for scenario $s = 1, 2, ..., K$ **do**

    Select experiences D_i^s which correspond to scenario s

    Calculate mean $\mu_i^{\mathrm{r,s}}$ and standard deviation $\sigma_i^{\mathrm{r,s}}$ of rewards for D_i^s

    if $\sigma_i^{\mathrm{r,s}} < \sigma^{\max}$ **then** ▷ Achieved stable performance

      if $\mu_i^{\mathrm{r,s}} >= \mu_{i-1}^{\mathrm{r,s}}$ & $\mu_i^{\mathrm{r,s}} >= r^{\min}$ **then** ▷ Achieved better performance

        Increase difficulty of scenario s:

        $c_{i+1}^s \leftarrow c_i^s + \Delta c^s$

      else if $\mu_i^{\mathrm{r,s}} >= \mu_{i-1}^{\mathrm{r,s}} - \epsilon$ **then** ▷ Achieved worse performance, but within margin

        Proceed with the same difficulty

        $c_{i+1}^s \leftarrow c_i^s$

      else ▷ Achieved worse performance

        Decrease difficulty of scenario s:

        $c_{i+1}^s \leftarrow c_i^s - \Delta c^s$

      end if

    else ▷ Achieved unstable performance

      Decrease difficulty of scenario s:

      $c_{i+1}^s \leftarrow c_i^s - \Delta c^s$

    end if

  end for

end for

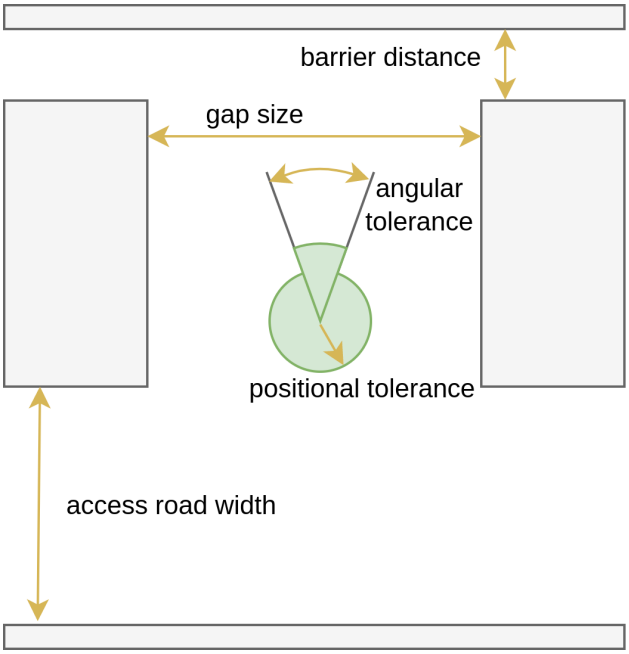


Figure 4.10: Definition of control parameter by which curriculum steers the difficulty of scenarios.

Table 4.4 Configuration of curriculum algorithm for parking use cases.

	Parallel			Perpendicular			At Angle		
difficulty	easiest	update	hardest	easiest	update	hardest	easiest	update	hardest
positional tolerance [m]	2	$\pm$ 0.1	0.25	2	$\pm$ 0.1	0.25	2.0	$\pm$ 0.1	0.25
angular tolerance [deg]	36	$\pm$ 5	3	36	$\pm$ 5	3	36	$\pm$ 5	3
gap size [m]	5	$\pm$ 0.1	0.75	5	$\pm$ 0.1	0.5	5	$\pm$ 0.1	5
access road width [m]	5	$\pm$ 0.1	1	5	$\pm$ 0.1	1.5	5	$\pm$ 0.1	1
barrier distance [m]	2	$\pm$ 0.1	0.2	2	$\pm$ 0.1	0.2	2	$\pm$ 0.1	0.2

The difficulty of parking scenarios is controlled by parameters such as the gap size, barrier distance, the width of the access alley, and the positional and angular tolerance of goals (see Figure 4.10). These parameters are changed adequately (within predefined ranges) to increase or decrease the difficulty by a constant defined for each control parameter separately (e.g. reducing gap size increases the difficulty).

A signal to increase difficulty is sent when the average performance of the agent, based on its return, has increased by a given margin with respect to the previous iteration and is stable (returned across episodes have a low standard deviation). Additionally, the minimum mean reward requirement was added to increase difficulty. To eliminate constant increasing and decreasing difficulty, a hysteresis was added in which difficulty is kept at the same level. If the performance drops by a given margin or becomes unstable, the difficulty decreases. This methodology is executed separately, with different parameters, for each parking scenario (parallel, at angle, and perpendicular). Details of the update procedures are presented in Table 4.4.

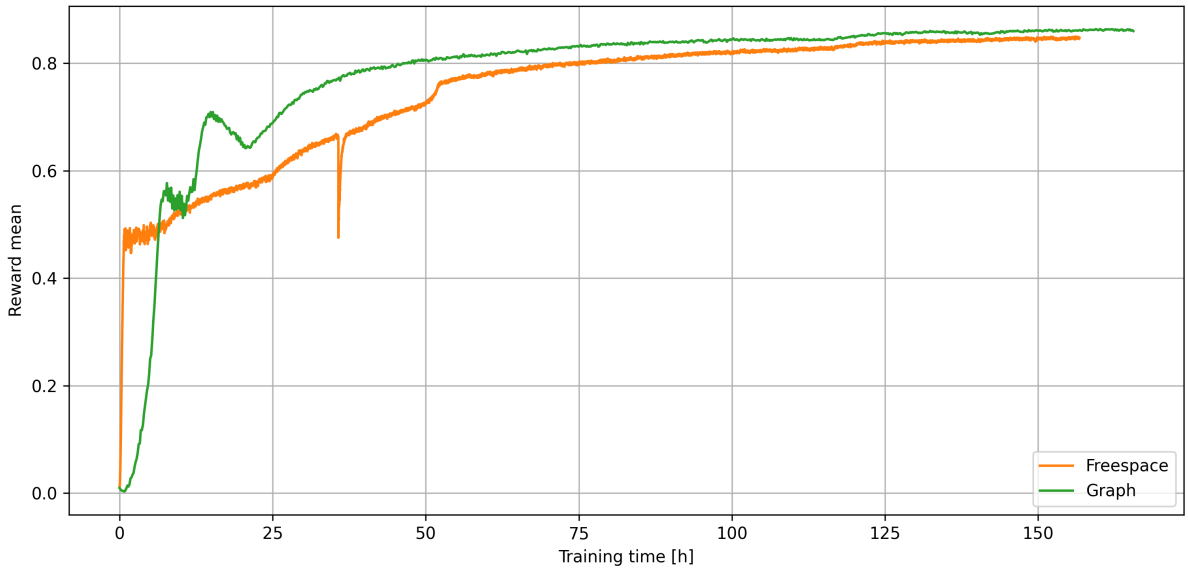


Figure 4.11: Reward average across trainings in a function of training time.

The use of curriculum learning dealt well with the initial training phases and was able to conduct training that in the end resulted in the well-performing agents in a target set of parking scenarios. The results of its application will be shown in 4.8.

4.8 Experiments

4.8.1 Observation Space and its Processing

As the reward in the case of both policy definitions was defined in the same way, the performance of the agents might be compared at high level by looking at the reward graph resulting from trainings. First of all, both models acquired in the end comparable reward scores. Looking at the reward graphs as a function of training time (see Figure 4.11), after good start of freespace system graph one gains advantage in the phase when curriculum process started to increase the difficulty of scenarios, and kept that advantage until the very end. Drawing the reward scores as a function of the simulation steps collected (Figure 4.12), a huge difference can be observed in the sense of the sample efficiency in favour of the graph model. It seems then that graph-based agent might build better discrimination of the scenarios from the start of the training, while freespace has to relay on detailed cues in observation. On the basis of that argument, an argument might be made that the training freespace model is much harder in the sense of sample efficiency with respect to the graph model, although comparable in terms of the wall clock.

In Figure 4.13 example parking scenarios are presented. Dots specing indicates consecutive vehicle positions and enables to judge when agent was using big movement of 1 meter and when a smaller one of 0.25 meters.

To follow up on the topic of models inference and training performance, time execution measurements of both models have been performed, and summarised them in Table 4.5. Based on the acquired measurement, the inference time of Graph NN is twice the time slower than Freespace NN. The faster execution of

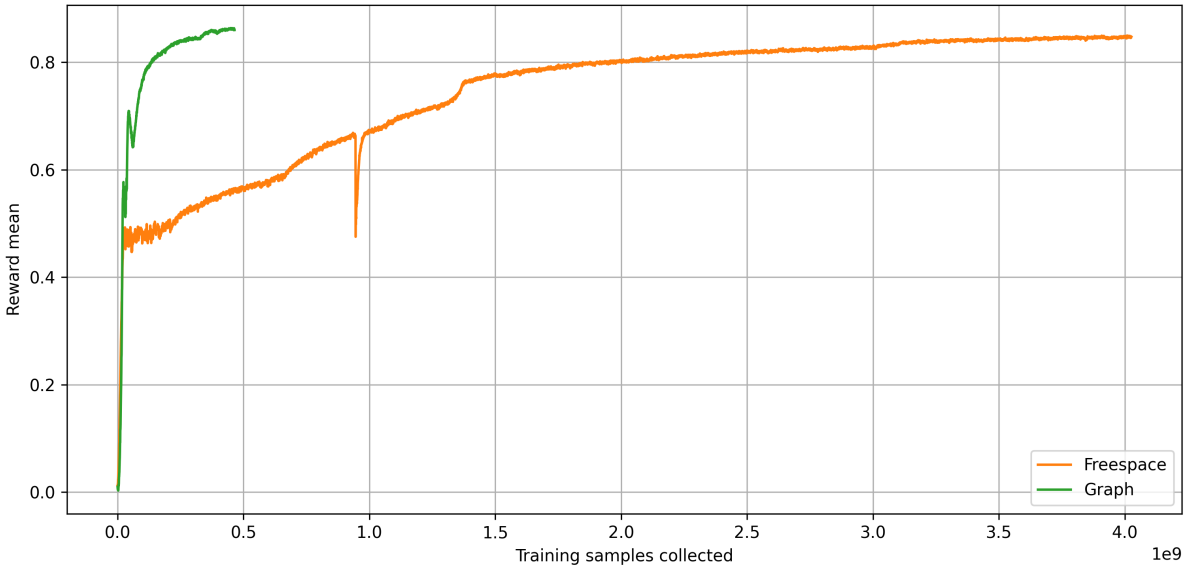


Figure 4.12: Reward average across trainings in a function of collected samples.

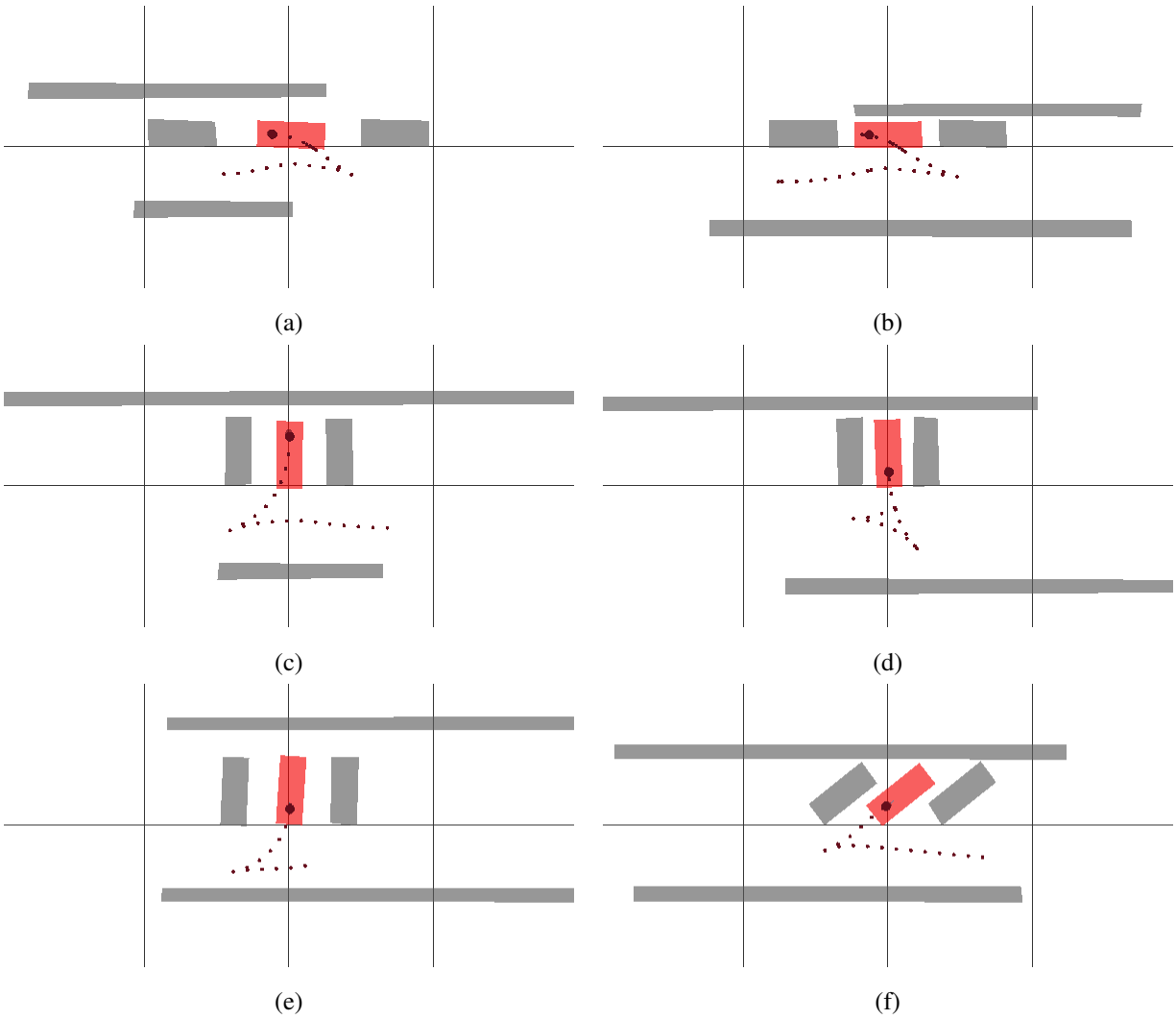


Figure 4.13: Visualisation of parking manoeuvres performed by a Freespace agent.

Table 4.5 Inference and training iteration times for models used in parking experiments. Training iteration involved gathering 100'000 samples across 16 CPU cores and running 10 PPO optimization rounds on GPU. Machine details: CPU: 12th Gen Intel® Core™ i7-12800H × 20, GPU: NVIDIA RTX A1000, Python: 3.8, Pytorch: 1.11, Ray: 2.3.1.

	Graph Net	**Freespace Net**
Inference time for single sample - average	2.22 ms	1.021 ms
Inference time for single sample - standard deviation	0.652 ms	0.446 ms
Training iteration - average	115.3 s	12.7 s
Training iteration - standard deviation	2.19 s	0.41 s

Freespace NN is even more visible when comparing training iteration times, where Graph NN is approximately 10 times slower, indicating a slow training process of graph-based neural networks. All experiments were performed with Python-based Pytorch models, which have not been optimised by any means. The inference times for single samples have been measured while inferring model on a CPU, while training iteration times involved rollout collection on a CPUs and optimisation on a GPU. Experiments setup, in addition to models, differ only in the way in which observation is created.

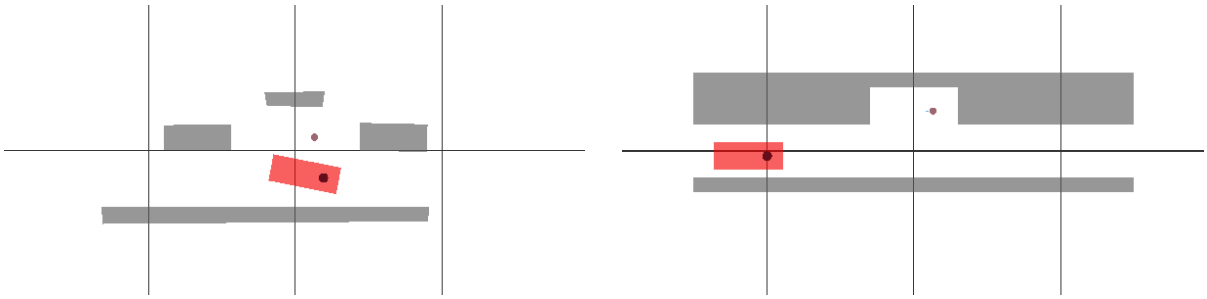
To better understand performance differences, an analysis of the most important qualities of the planned path for each system was performed. The summary of this analysis can be found in Table 4.6. The outcome of the performance analysis was in line with the mean scores of the acquired reward, proving that the graph neural network-based system performed slightly better than the freespace one. Overall results show as well that graph-based system has generated smaller amount of collisions, which might suggest that this way of processing information about obstacles provides better information in close vicinity to the agent. One advantage of the graph formulation, which originates from the definition of observation creation mechanism, is providing more context in scenarios where obstruction plays a role. In terms of the number of steps and the generated path length, the graph-based system performed better again. This metric suggests that graph-based agent might have better understanding of the scene through the whole episode, while its freespace version knowledge depends more on current perspective, and thus it has to spend more time to drive around the scene. One metric in which the free-space system gained advantage was the average direction change count, which has been part of the reward signal formulation.

Looking at the analysis per scenario, the biggest difference has been visible in the parallel use case, in which the graph version acquired superior performance in terms of all measures. Looking at the mean performance, parallel parking was the most difficult parking scenario. The freespace model worked better for perpendicular parking. Surprisingly, the lowest performance for both models has been acquired in parking at angle scenarios, which in theory should be the easiest ones in the set.

Robustness analysis by running the policy in evaluation scenarios was also performed. A data set with scenarios that were aligned with the training use cases from a functional perspective has been prepared, but where obstacles were differently shaped (in general, the obstacles were larger and less randomised; see Figure 4.14 for comparison). Then an analogue performance analysis as previously was performed, which

Table 4.6 KPIs measures for parking agents trained with two kinds of neural network - Graph neural network and Freespace neural network, calculated for three kind of scenarios.

	All scenarios		Parallel		Perpendicular		At Angle	
	Graph	Freespace	Graph	Freespace	Graph	Freespace	Graph	Freespace
samples number	5000	5000	1648	1680	1717	1683	1635	1637
goal reached [%]	**98.46**	98.26	**99.27**	98.3	98.19	**98.57**	**97.92**	97.86
in collision [%]	**1.44**	2.1	**0.73**	1.67	1.63	**1.42**	**1.96**	3.23
average path length [m]	**14.24**	14.53	**14.88**	15.45	14.43	**14.26**	**13.4**	13.71
average episode length	**16.6**	17.33	**17.2**	18.93	**16.85**	16.89	**15.7**	16.13
average direction changes	1.5	**1.42**	**1.11**	1.17	1.66	**1.55**	1.73	**1.54**



(a) (b)

Figure 4.14: Differentiation between the example training scenario (Figure a) and the validation scenario (Figure b). Both scenarios are constructed by four obstacles, but the validation ones are more enclosed and less randomised.

might be found in Table 4.7. Although the performance has drooped significantly in the case of both models, the free space version has been much more robust than the graph model, where the performance of reaching the goal dropped to around 30%. Those results suggest that the shape of the obstacles, even if it do not play any role in the functional aspect of the problem, might have an effect on agent performance. As the the observation difference in freespace net is smaller than in the case of graph model, the primer is more robust to such disturbances in scenario definition. These results underscore the necessity of heavy randomisation of training scenarios to acquire good performance.

4.8.2 Real-World Experiments

Having well performing module of parking slot planner, the decision has been made to integrate it inside one of Aptiv's test cars (Figure 4.15). For this purpose, the development 2019 BMW 7-series with integrated sensor suite, including front and corner radars, front vision camera, and roof LIDAR sensor was used. The test car was also equipped with an industrial-class computer, allowing custom algorithms to be run and monitors to present the algorithm output. The aim was to run an RL-based parking slot planner, which consisted of both the environment part and policy itself in an open loop, and visualise live the results.

Table 4.7 Performance of both models in case of validation scenarios.

	Graph	Freespace
samples number	810	810
goal reached [%]	30.12	**74.45**
in collision [%]	68.89	**25.30**
average path length [m]	20.6	**15.99**
average episode length	23.24	**19.23**
average direction changes	5.41	**1.74**

Python-code prototypes of the function and, for the sake of time optimisation, the freespace version of the policy.

The setup of the car and its abilities differ slightly from how the problem definition in the simulation, therefore the setup and add some functionalities required to be adopted, which are described in a high-level below:

- The car's surroundings has been represented in the form of a 200 by 200 grid, representing the area of 20 metres by 20 metres, with each cell being occupied or free. Because of that, the freespace model was used, as it was easier to encode the observation for that network.
- To create free space observation for the policy network, the ray casting algorithm has been employed, allowing one to derive the distance to the first obstacle in the ray's path.
- As an additional precaution, a mechanism was added that prevented the agent from pulling out of the grid. This was achieved by constraining both free-space observation detection by a grid border and detection collisions when the car wanted to drive out of the defined area.
- Collision detection has been performed by checking if any occupied cell lies within the rectangular shape of the ego. If that was the case, a collision has been reported.
- To define the scenario, potential parking spots in each of the grid measurements needed to be found. To do so, a simple algorithm has been implemented that analyses the free space on the left and right of the longitudinal axes of the car. The algorithm was designed to detect parallel and perpendicular parking spots, the former having precedence.

The initial evaluation of the simulation-trained policy showed weak performance in the in finding paths to detected parking spots. This problem was most likely analogous to one observed in the validation set of scenarios, where the nature of the observation has been changed. These issues could be addressed by further randomisation of scenarios. In this case, however, to address this, the policy has been retrained (policy weights have been initiated by those trained in pure simulation) on data collected during test drives. During recording session, car collected series of sensors readings and, resulting from them, grid representation of the car's surroundings. In each of the grid frames, a parking spot finder algorithm was run. The grid frame with parking spots was then treated as a scenario definition.

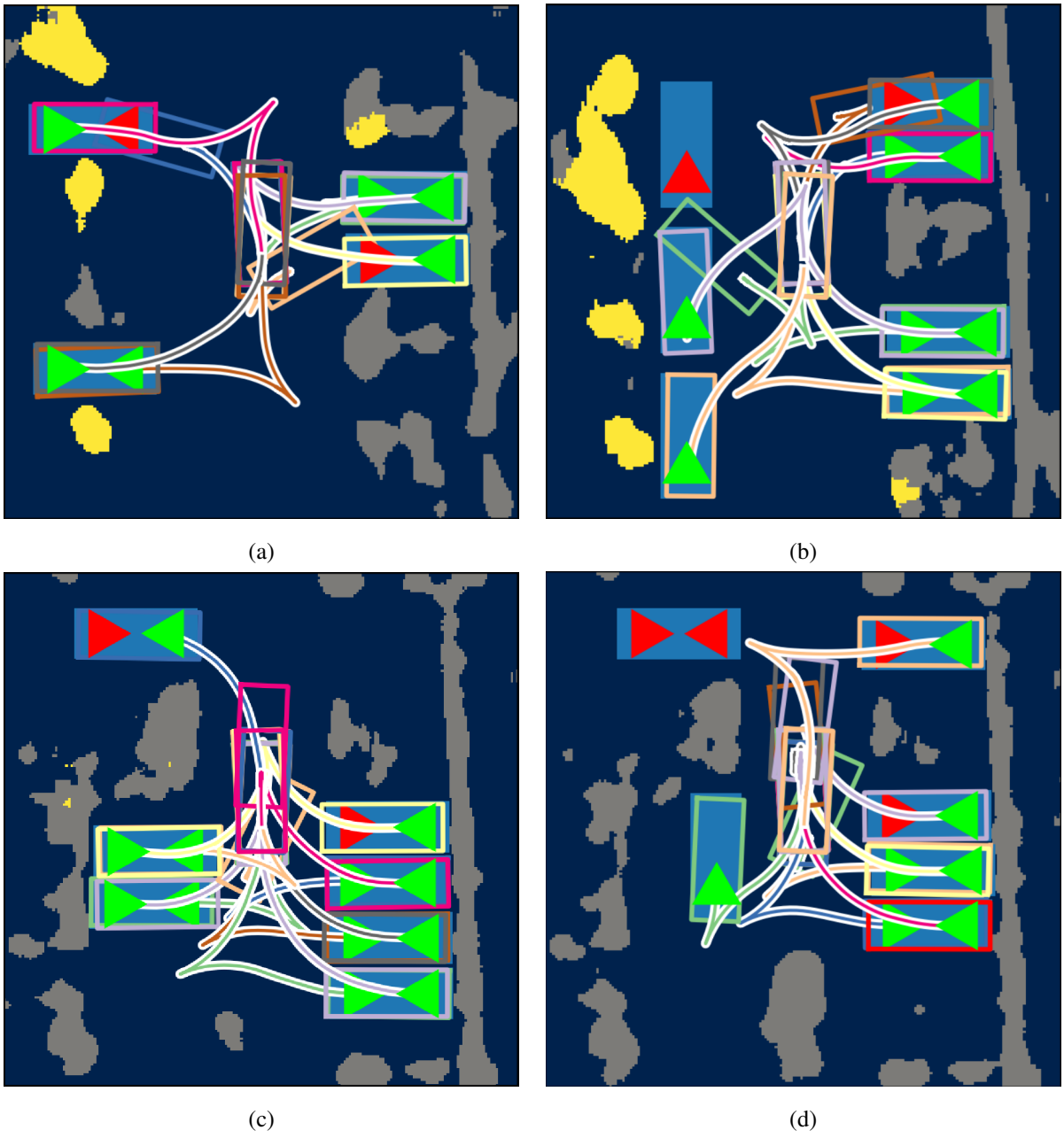
Figure 4.15: Aptiv test vehicle where RL-based parking spot planner has been integrated.

The PPO algorithm in multi-agent setup was used, spawning as many simulated vehicles as detected spots, each with its own goal. Each simulated agent has been invisible to the others. Subsequently, a standard interaction between agents and the environment was performed, which resulted in multiple rollouts from each simulated vehicle.

The same methodology has been used during evaluation of the module in the vehicle, therefore algorithm was able to plan the path to multiple parking spots in each iteration. To save time, inference of the policy has been done in batch manner, allowing one to define the action in a single pass through network for all active agents. The output of the RL-based parking spot planner can be seen in Figure 4.16. In most cases, the agent is able to successfully park in the designated position. The proposed training mechanism allowed one to adapt the policy to real-world scenarios. Additionally, with relatively small computation time requirement, the proposed solution allows planning paths to different parking spots in real time, potentially providing great value to the end customer. This includes planning manoeuvres in real-time as the car moves or checking the possibility to park in detected parking spots just by planning a path.

4.9 Discussion and Further Work

Conducted research and experiments suggest big potential in application of reinforcement learning methodology to the use case of parking spaces. Both in terms of functional and computation performance, the presented method provides a window of applicability to a real-time system. The experiments performed present the strengths and weaknesses of both observation mechanisms with the resulting neural network architectures. In case of optimising for execution time, the freespace version of the policy should be treated as candidate to choose. In case of more complex scenarios, especially if one would like to encode more complicated shapes and consider potentially occluded objects, a graph-based network is a viable option.



(a) (b)

(c) (d)

Figure 4.16: Examples of paths found by RL-based policy in real-world data. Both yellow and grey areas indicate obstacles, while dark blue colour represents the free space. Rectangles of different colours, along with corresponding paths, represent multiple agents with their corresponding parking spots in light blue. Arrows represent the target position, with green indicating successful parking of agent in a given spot, and red indicating failure in doing so.

Further research allowing to combine both options seems to be as well a promising way of evolving the parking system.

The test performed on real data in a test car further supports the claim of a high applicability potential of the presented solution to the parking problem. In particular, the low computation cost requirements can enrich autonomous parking applications allowing parking manoeuvres to be planned in real time during the drive.

Presented research can also be followed up in more ways. First, more attention could be paid to simulating more realistic and complex parking scenarios, which would provide more variability in the static context definition. This process would allow the agent to be more robust to real-world scenarios and to use free space in an even more creative way. Another direction is investigating ways how the resulting RL-based policy could be integrated in more classical system and become a part of hybrid solution, which surely could bring benefits in the manner of action selection mechanism and reproducibility of paths. On the path to productionalization another challenge arises, including taking into consideration changeable car dimensions or potential sensor noise. To support the hybrid approach, the usage of other reinforcement learning algorithms like DQN or Monte Carlo tree search algorithms [122] might be helpful in approximating the true value or action distribution for the parking problem. Redesigning the reward function will help in generating desirable shapes paths, where its specific aspects like length or curvature could be addressed directly. Expanding the scope of planning from the slot parking manoeuvre itself to the parking lot is also an attractive research direction. Regarding test car deployment, introducing a closed-loop control mechanism, allowing to realise the proposed path, would further enhance the research. With this capability in place, the RL agent could be trained to generate more robust paths when faced with a potentially changeable perception of the driving environment, as the car would be in motion.

In Chapter 5 the training and testing environment was also extended to a multi-agent setup, in which the interaction between road users and how the reward function impacts the resulting policy was studied.

Chapter 5

Multi-agent Maneuvering

5.1 Introduction

Driving a car is inextricably linked to cooperation with other road users. In a lot of cases, those interactions are codified by traffic rules providing clear right-of-way definitions or are more one-way and might be seen as responding to other road users' behaviour (imagine a typical ACC scenario, when the following car responds to lead car velocity profile and has little or none impact on lead car behaviour). Still, there are situations where uncodified cooperation between road users is a necessity to effectively solve given road scenarios. Such situations may involve a lack of clearly defined right-of-way, scenarios in which following the rules is not enough, or where blindly following them yield suboptimal solutions. Such examples, in which all agents have to come up with their own strategy consistent with strategies of other road users, are often easily solved by human drivers but hard to codify by handwritten rules. When such alignment is not found, either some subset of road users will have trouble with achieving their own goals (busy road with priority where nobody is letting merging cars in), or all agents' performance will be jeopardised (cars driving in separate directions blocked in a bottleneck when either none or both of the agents aims at passing this section first).

At the same time, the reinforcement learning methodology provides the tools to tackle multi-agent setups, with the record of solving complex problems [7, 60, 91, 122].

In this part, multi-agent reinforcement learning methods was in focus and have been applied to challenging on-road scenarios that require extensive cooperation from all the agents. To train and later evaluate the resulting control policies, a multi-agent environment was implemented that has been based on a parking environment (introduced in Chapter 4). In the extended version, multiple agents are simulated at once, each with its own distinctive goal. Based on that, scenarios similar to road situations were formed, in which coordination of agents is a necessity.

Part of the simulated scenarios is quite novel for reinforcement learning setup, which is quite specific in its kind, therefore comparison to other methods is difficult to consider. In some cases, such a comparison would not be informative due to different assumptions in problem formulation (like in the case of global planning methods), and some would require extensive implementation efforts (e.g., the system for prediction and planning). Existing methods should be then considered as important solutions to the stated problem, but

cannot be easily and directly compared with the approach which has been taken in this research. Therefore, the focus has been on comparing different customisations within the proposed method and has referred to the baseline acquired by training in the simplest version of the proposed approach.

The method proposed in this research relies on a combination of a relatively straightforward training approach, policy design, and environmental dynamics with a carefully designed scenario generation process. Policy inference is based only on the local perception of the scene and, therefore, agents do not rely on any communication with other road users or infrastructure and resembles the human driver decision-making process. By achieving satisfactory performance, it might be argued that individual agents' policies lead to a globally efficient strategy in some of the most challenging cases from a decision-making perspective in autonomous driving. In the following part of the research, the introduction of a shared reward mechanism was examined, which both improves training efficiency and allows better coordination of actions in congested traffic scenarios. It allowed as well to train the policy to cover other agents' objectives, without the requirement of having access to those during execution time. On the basis of the results, it might be argued that the proposed approach is well suited to motion planning applications, especially when dealing with multiple road users, and its main principles might be used in autonomous driving decision-making systems. The outcome of multi-agent research has also been summarised in [93].

5.2 Problem Formulation and Assumptions

5.2.1 Problem Formulation

Experiments summarised in this chapter aim to solve the multi-agent aspect of driving. Using the same base environment as in parking experiments (see Chapter 4), three scenarios have been modelled that require agents to cooperate, including zipper, bottleneck, and crossroad. All agents in the scene are controlled with the same trained policy and share its parameters.

Due to the existence of multiple agents in the scene, the problem is modelled as Partially Observable Markov Game, but in line with previous experiments partial observability has not been addressed in any specific manner during training process. At the beginning of each episode, the environment is generating a scenario consisting of obstacles which reflect the road infrastructure (like crossroads) and a given number of agents in their initial positions, each with its individual goal position. The optimised policy, shared across the agents, is responsible for deriving an action which consists of combination of acceleration and steering angle for each of those agents individually. Policy operates only on local information, which includes goal position, ego vehicle speed, freespace measuremenet around the car (which reflects both static context and other agents) and relative position and speed of other agents. In response to all agent actions, the environment simulates their movement according to kinematic motion model. Later, the check for potential collision between agents and infrastructure and between agents is made, where any collision terminates and excludes from simulation all agents involved in the collision. Similarly, each agent is checked for achievement of the goal. At the end, a new set of observations is provided to all active agents in the scene. The episode ends when all the agents terminates individually by reaching the goal or being involved in a collision.

A multi-agent version of the PPO algorithm was used, where each observation is collected in the same buffer and used to optimise singular policy. Different reward mechanisms are examined and compared, allowing to estimate the effect of multi-agent rewards application.

5.2.2 Assumptions and Limitations

Our multi-agent experiments with the following assumptions and limitations:

- All vehicles movements are modeled with bicycle kinematic model in plannar way, not assuming any dynamic effects. As in some cases this assumption might introduce errors, it might be argued that the ADAS and AD systems operate in limited dynamic scope, so that the dynamic effects are negligible and can be addressed at the control level.
- No traffic laws or regulations were directly modelled, as agents were asked to manoeuvre around each other without prior knowledge or limitations. Some of the behaviours acquired during training that resemble human driver behaviour are the effect of crafting scenarios in a specific manner (like placing agents on the correct side of the road). Some of the behaviours are just the effect of randomness and might not reflect the desired behaviour (see bottleneck scenario manoeuvres, where agents pass themselves on the left, while at crossroads they follow the right-side movement pattern).
- The occlusion happening in the scene for agents' observation were not modelled, therefore each agent has perfect knowledge of all the agents in the scene regardless of their position.
- All agents in the scene are controlled with the same policy, therefore, resulting policy is surely not robust to out-of-distribution behaviours. This is a strong assumption, and fitting to other behaviour profiles needs to be addressed before moving to the application stage.
- Perfect knowledge of the surroundings without any measured noise modelled was assumed, which is a not realistic assumption. The use of randomised data, ideally based on real-world measurements, would be a preferable next step.

5.3 Prior Art

The problem of multi-agent coordination in complex and interactive scenarios for on-road maneuvering is an active field of research. In [152], authors summarised the most common approaches and provided the taxonomy for the methods used in intersection scenarios, however, this review could be extended to non-intersection cases as well. Cooperative driving strategies rely on communication between vehicles and infrastructure (V2I–vehicle–to–infrastructure communication) or between vehicles directly (V2V–vehicle–to–vehicle communication). In the first case, planning is made in the central coordination unit which was assigned to a given intersection. By having access to all vehicle information and controlling them, these methods might define globally optimal strategies, which is their biggest advantage. This strategising can be done at the level of individual agents [157, 159] or by grouping the agents into platoons and establishing drive-through priorities of those [49, 62, 169]. In the case of direct V2V communication, the planning

process is distributed among agents who communicate with each other to establish a suboptimal coordination solution. The strategic interaction of agents based on cooperative game theory has been successfully applied in [24, 46, 64]. Methods based on model predictive control applied to multi-vehicle traffic optimisation have been presented in [56, 82].

As cooperative methods undoubtedly come with big advantages, including the ability to define the globally optimal strategy and directly communicate the intention to others, they also have serious limitations. In the case of centralised cooperative methods, additional hardware has to be mounted in place of interest. Therefore, methods that are limited to specific locations, while negotiation might be required in random places as well. Furthermore, all cooperative methods heavily rely on communication and, by definition, require all traffic participants take part in that process. As human-driven cars, bicycles, and pedestrians are an inherent element of traffic (at least in the near future), this requirement cannot be fulfilled. This is the reason why most implementation-orientated research is focused on the second family of methods, which is concerned with individual driving strategies.

Quite similar to how human drivers make decisions on the road, individual driving strategies rely purely on on-board perception and aim at taking decisions consistent with other road users' strategies. By losing access to perfect information about other road users' intentions, methods struggle to derive the globally optimal strategy, however, they support operations with non-automated agents and do not require any communication with them. Often, these methods require extended scene perception, including complex understanding of road structures, object-to-lane assignment, traffic lights and signs detection and understanding, as well as the intention and trajectory prediction of other road users. The last part is often executed simultaneously with the planning algorithm to check how agent's decisions impact the potential behaviour of others.

The baseline methods from this family are based on the concept of Finite State Machines (FSMs), which divide the vehicle state into a set of modes, which are later traversed with FSM supervision based on a list of heuristics, resulting in specific control strategies. Extension of FSMs, handling hierarchical states presentations, has been successfully applied in one of the first autonomous driving experiments in urban scenarios- the DARPA Urban challenge [40, 81]. However, those methods are suitable only for simple cases and struggle to cover the vastness of possible on-road scenarios [67].

The obvious choice is to address the multiplicity of scenarios and cases in data-driven methods. Agent control by imitation learning with the addition of expert data has been presented in [8]. As predicting the future is an essential part of planning motion in dynamic and reactive environments, numerous methods used neural network models to predict other road users' behaviours and trajectories [19, 100, 110]. At the same time, since trajectory planning is a typical closed-loop control problem, reinforcement learning methodologies have also been utilised in the autonomous driving domain [23, 57, 63, 147].

In Section 2.2.10 the main concepts in Multi-Agent Reinforcement Learning (MARL) have been introduced, along with a subset of the most influential publications. Here, this review was enriched by referring to MARL methods applied to different autonomous driving problems.

The authors of [97] introduced a platform that allowed the definition of customisable learning approaches for multi-agent autonomous driving systems, which enabled training agents in a partially observable 3-way intersection environment, where the right-of-way has been controlled by stop signs while agents

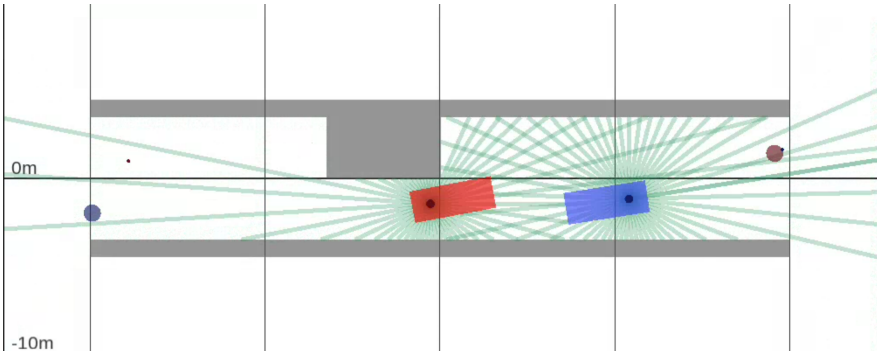


Figure 5.1: The bottleneck scenario simulated in a maneuvering environment, including two agents trying to negotiate to drive through it. The goals of individual agents are in the same color as the corresponding agents, while green lines represent freespace simulation.

operated based on raw camera observation. In [22], the supervisory system based on communication channels between vehicles and infrastructure (V2I) has been developed to control the merging of cars on the highway. Cooperation between automated vehicles and human drivers has been studied in [133, 134], which introduced the notion of altruism in the definition of reward functions that improve safety and traffic flow. The application of reinforcement learning to behaviour planning in both highway and urban scenarios, while employing a semantic action space, was implemented and tested in [118, 119]. In an effort to find potential failure modes of a given autonomous driving system, the authors of [144] introduced adversarial agents based on MARL, with the objective of causing collisions with other road users. The use of the MARL methodology for adaptive traffic signal control (ATSC) was shown in [26], solving the issue of a massive action space of a large ATSC system by distributing control to individual local agents. The authors of [34] presented a survey that enriches the above review of the literature.

5.4 Multi-Agent Manoeuvring Environment

Using the parking environment introduced in Section 4.4 as a base, the simulation was extended with the modelling of more than one agent and their corresponding goals. Information about the location of a specific goal is available only to the agent that owns it. The same mechanism for collision detection as in the Parking Environment was used, adding agent-to-agent collision checks. Such events result in the termination of all involved agents and their removal from the simulation. All agents observe and act locally from their individual perspectives and do not require any communication channels between them.

5.4.1 Motion Modelling and Action Space

Simulating interactions between road users in low-speed scenarios requires tracking the dynamic state of each vehicle, as their current speed and steering play an important role in predicting their intentions and negotiation. To do so, the movement of cars was simulated in the form of the bicycle model [70].

The position of the given agent has been represented as x and y values in the global coordinate system, with the orientation angle (yaw) ψ defined along the longitudinal axis of a car. Additionally, the current

velocity value along the orientation angle was represented as v, with the yaw rate ω. No modelling of the slip angle was performed. The input to the motion model, understood as a control, is defined as the acceleration value a and the angle of the front wheels δ. The model is parameterized with the wheelbase L, representing the spacing between the axes. The amount of time to be simulated is represented by Δt. The equations that define the motion model are defined as follows.

$$x_{k+1} = x_k + r\left(\sin(\psi_k + \omega_{k+1}\Delta t) - \sin\psi_k\right), \tag{5.1}$$

$$y_{k+1} = y_k + r\left(\cos\psi_k - \cos(\psi_k + \omega_{k+1}\Delta t)\right), \tag{5.2}$$

$$\psi_{k+1} = \psi_k + \omega_{k+1}\Delta t, \tag{5.3}$$

$$v_{k+1} = v_k + a_k\Delta t, \tag{5.4}$$

$$\omega_{k+1} = \frac{v_k + \frac{a_k\Delta t}{2}}{r}, \tag{5.5}$$

where the turn radius, r, is derived as:

$$r = \frac{L}{\tan\delta_k}. \tag{5.6}$$

In the straight motion use case (when $\delta = 0$), which would result in a badly defined turn radius r, the equations are simplified. Only forward movement was allowed and the maximum velocity has been introduced, by saturating the vehicle speed between $0\,\mathrm{m/s}$ and $7\,\mathrm{m/s}$. Saturation applied to velocity has direct implications on the distances travelled in each step, therefore, of most of the model.

With this motion model in mind, we defined the action space, which was a combination of a discrete sets of accelerations (in $\mathrm{m/s^2}$) and wheel angles (in degrees):

$$\mathcal{A} = \mathcal{A}^a \times \mathcal{A}^\delta, \tag{5.7}$$

where

$$\mathcal{A}^a = \{-1.0, -0.5, 0.0, 0.5, 1.0\}, \tag{5.8}$$

$$\mathcal{A}^\delta = \{-28^\circ, -24^\circ, ..., 24^\circ, 28^\circ\}. \tag{5.9}$$

5.4.2 Environment Observation

During the design of the observation of the environment, the inspiration was taken from [7] which dealt with a similar multi-agent problem in a simulated game of hide-and-seek. The description of the observation space used in the maneuvering environment can be found in Table 5.1. The same as in the Parking Environment, the space around individual agents has been represented by a set of distance measurements to the closest obstacle at uniformly distributed azimuths, resulting in 50 values (see Figure 4.8). As the motion model used was aimed at representing a car driving with a given speed, information about current velocity and yaw rate has been added to the relative position of the goal in the ego coordinate system, representing ego-car information. To provide information about other agents in the scene, parameters of individual agents are represented as relative position and speed in the ego coordinate system, along with doubled information

Table 5.1 Observation space used in maneuvering environments. All values has been normalized according to their natural ranges.

Values	Space	Range	Description
context	[4]		contextual information about the goal and the ego vehicle
v		$< -1, 1 >$	current ego longitudinal velocity
ω		$< -1, 1 >$	current ego yaw rate
x_{g}		$< -1, 1 >$	relative x position of the goal in VCS
y_{g}		$< -1, 1 >$	relative x position of the goal in VCS
freespace	[50]		freespace described as a set of distance measurements around the car
d_i		$< 0, 1 >$	freespace at the i-th azimuth to the closest obstacle or other agent; saturated at 50 metres
agents	[10 x 8]		list of agents in the scene with concatenated context
agent	[8]		encoding of the single agent
x_a		$< -1, 1 >$	position in the x-axis of the a-th agent
y_a		$< -1, 1 >$	position in the y axis of the a-th agent
v_a^{x}		$< -1, 1 >$	velocity of the a-th agent in the x-axis
v_a^{y}		$< -1, 1 >$	velocity of the a-th agent in the y-axis
v		$< -1, 1 >$	current ego longitudinal velocity
ω		$< -1, 1 >$	current ego yaw rate
x_{g}		$< -1, 1 >$	relative x position of the goal in VCS
y_{g}		$< -1, 1 >$	relative x position of goal in VCS
valid agents	[10]	$\{0, 1\}$	indication of agents' validity defined above

about the ego context. Those embeddings are later concatenated to form a list, supported by a mask allowing to identify which embeddings are valid and represent a real car (in case the number of agents is lower than one assumed by the neural network).

5.4.3 Scenarios

The motion model and collision detection, along with the interface defined by observation and action spaces, make up the base of the multi-agent manoeuvre environment. Concrete simulated use cases are realised by defining scenarios, which are parameterized as combinations of agents' initial positions, corresponding goals, and obstacle configuration.

With an aim of simulating scenarios that would require agent cooperation, three classes have been defined. These are Bottleneck, Zipper, and Crossroads, which are explained in detail below.

Bottleneck

The bottleneck case is often encountered on smaller roads or residential areas (Figure 5.2). It includes a narrowing that prevents agents driving from opposite directions from freely passing each other and forces one of them to wait before the bottleneck and give way to the other vehicles. In the simulated case, two agents have been spawned at opposite ends of the 40-meter long road, which is 7 metres in width. The

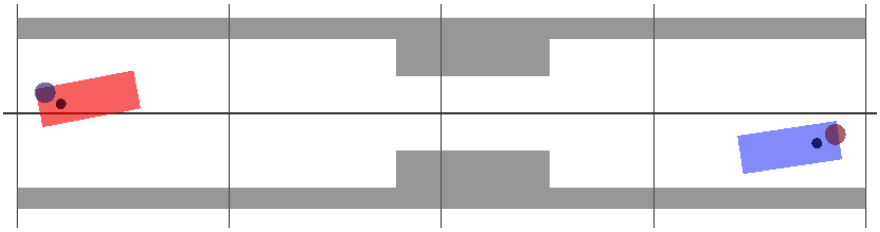


Figure 5.2: The bottleneck scenario with a centrally placed bottleneck.

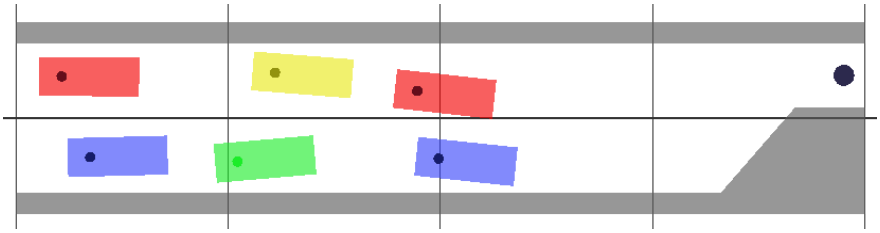


Figure 5.3: The zipper scenario with the narrowing located on the left side of the road.

bottleneck part narrows the road to 3.5 metres. The agents' goals have been placed at the other end of the road, respectively.

The bottleneck itself has been randomly selected for its precise location, size, and specific structure. In these experiments, the test scenarios included those with no bottleneck at all, singular or two consecutive narrowings, as well as placement of the narrowing on one side of the road or centrally.

Zipper

The zipper scenario can often be found in cities or during road work, when one of the lanes is closing, causing two lanes to become one (Figure 5.3). In case of high traffic, the agents have to negotiate their right of way. The simulated scenario included six agents simulated in two lanes with a singular goal location placed at the end of the narrowing. The narrowing itself might have been spawned on the left, centre, and right side, or might not have been spawned at all.

Crossroad

The four-way crossroad scenario is encountered most frequently in residential areas (Figure 5.4). At the cruciform intersection, up to 10 agents on different connecting roads were spawned, with the goals randomly selected on other connecting roads ending. Arbitration of road priority was not introduced.

5.5 Policy Optimisation

All performed experiments were done with agents trained with a self-play mechanism, which improved data efficiency and allowed synchronisation of agents' strategies. Training followed the principle of decentralised experience collection and centralised training. During execution, every agent acts only having access to local observation. All the experiences collected by all the agents across all the episodes are gathered in a centralised replay buffer and are used later to improve the policy.

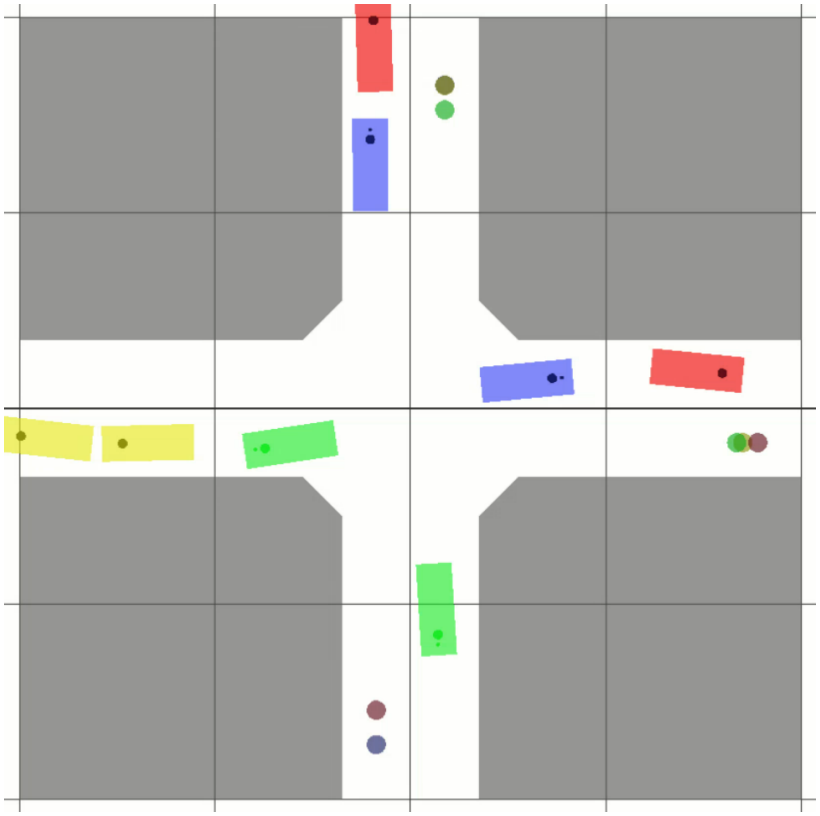


Figure 5.4: The crossroad scenario, with multiple agents each aiming at a different end goal, which is color-coded.

The policy is trained with PPO [115] with General Advantage Estimation as a value estimation, using the implementation included in the RLLib library [65]. To support multi-agent use cases and improve training efficiency, the guidelines introduced in [161] were partially followed, including reward normalisation, increase in train batch size, and reduction of the policy optimisation step number per train batch. No special adaptation has been made to the PPO algorithm to address multi-agent coordination. A list of the parameters used in training is provided in Table 5.2. All experiments have been run on the local cluster (introduced in Section 3.6.2), and for most of the training runs 50 rollout workers were used for experience collection and a single GPU for policy optimisation.

All the agents in the simulation share the same policy parameters, but observe and act based only on local information. No communication channels are present. Based on the neural network architecture introduced in [7], similar architecture responsible for agent control (Figure 5.5) has been created. The closest surrounding of the agents is presented in the form of a free-space observation, and it is processed with 1D Circular Convolution, and concatenated with ego-specific data. Each of the other agent's data is first concatenated with ego-data (for context) and later embedded with the use of FC layers, which are shared for all the agents. Those embeddings, along with the embedding of the ego itself, are then processed with three masked multi-head attention layers. Masking is used to eliminate non-existent object embeddings from processing. The embedded corresponding to the ego is selected and processed with FC layers to result in a discrete action distribution and an estimate of the value function.

Table 5.2 Parameters of PPO algorithm used in all experiments.

Parameter	**Value**
train batch size	2,000,000
number of sgd iterations	6
discount parameter γ	0.995
GAE parameter λ	0.95
kl coefficient β	0.0
clipping parameter ϵ	0.1
gradient clipping	2.0
learning rate	5×10^{-5}

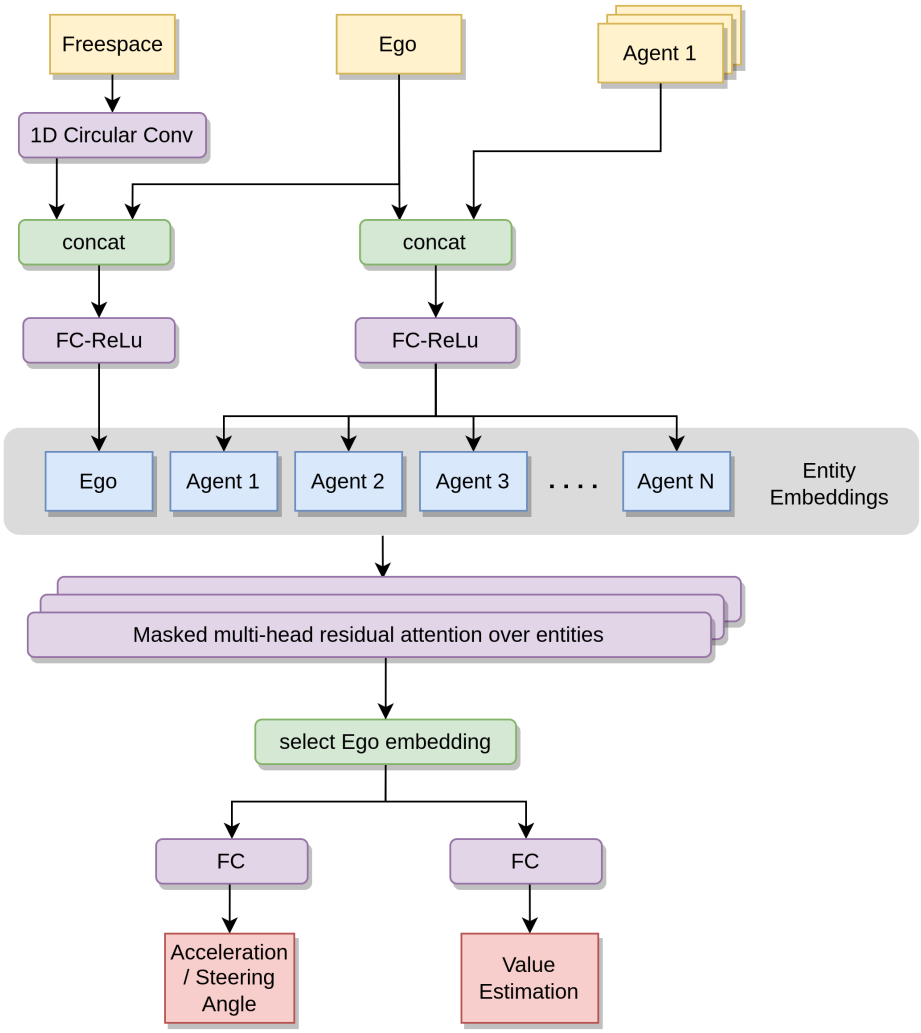


Figure 5.5: Graph representing neural network architecture. Color scheme follow the one introduced in Figure 3.12.

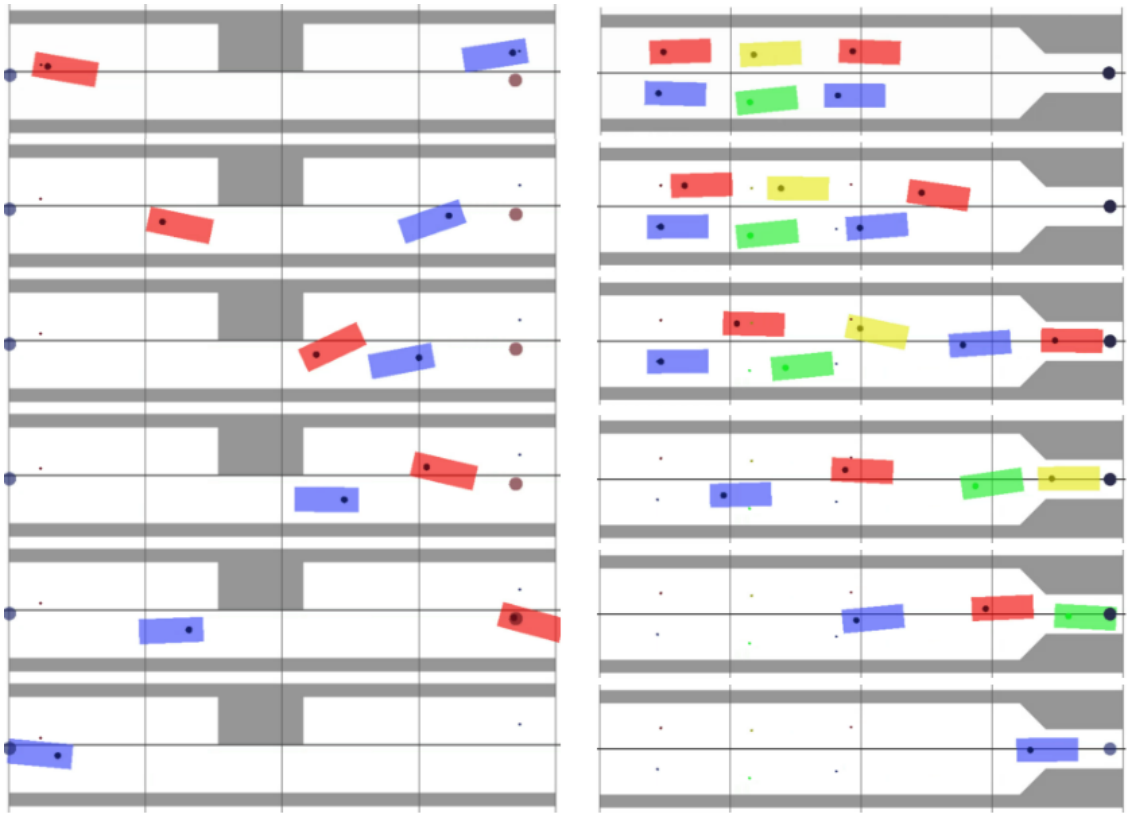


Figure 5.6: Evolution of episodes for bottleneck and zipper scenarios.

5.6 Results

5.6.1 Egoistic Rewards Training Evaluation

The simulated road scenarios can be characterised as episodic tasks from the perspective of individual agents, with clear end criteria of arriving at a goal or causing a collision. Due to this, in the initial experiments, a straightforward, sparse reward mechanism that was purely egoistic was implemented. In this setup, each agent receives a +1 reward when arriving at the goal (resulting in the end of episode for this agent), and is not rewarded in all other cases (Equation (5.10)). This definition does not directly encourage agents to maximise their speed except for the natural effect of the discount parameter (γ), which is less than 1.0. Agents have been trained in all scenarios separately.

$$r(s, a, s') = \begin{cases} 1, & \text{if goal is reached} \\ 0, & \text{otherwise} \end{cases} \tag{5.10}$$

Agents have been trained separately in all the mentioned scenarios (zipper, bottleneck, and crossroads), using a trained policy for all agents on the scene.

The evolution of example episodes with trained agents can be checked in Figure 5.6 and 5.7. The most interesting and immediately conspicuous conclusion is that all training resulted in cooperative and well-performing agents (see Baseline data Table 5.3 for the results in the Crossroads scenarios), despite the fact that the defined reward was purely egoistic. After careful analysis, the potential reason for this behaviour

was related to the fact that simulated scenarios are inherently cooperative and their competitive traits are negligible. With such problem formulation, situations in which one agent takes advantage of the other without cooperation and at the same time not jeopardising its own performance are hard to achieve. Numerous factors contribute to such a situation, which suggests potential differences between this setup and real-world use cases. First, accidents and potential blockages have an equally detrimental effect on the performance of all agents involved. Going further, the lack of traffic rules that require prioritisation of the drive-through or assigning guilt for an accident does not allow agents to be penalised for specific actions. Additionally, sharing the same policy parameters by all agents allows one to derive a consistent policy. Last but not least, agents are not directly rewarded for arriving at the destination as fast as possible, so they do not mind waiting for their turn. To conclude, with the main incentive not to cause collisions, agents have to acquire strong cooperative skills to maximise performance, and the individual goals of each agent are aligned with the goodness of all the others.

By comparing the training times required to achieve stable performance, one might conclude that the hardest scenario to train was the bottleneck (see Figure 5.8). The reason for that is most probably the fact that a deadlock situation is easily achievable, and such a case has to be carefully negotiated and demands long prediction and planning horizons.

Taking into consideration all the setup limitations, such as lack of traffic rules, shared policy among all agents, no adversary actors in the scene, and a narrow set of scenarios, it is important to highlight the good performance of the policy. Simulated scenarios are not easily solvable, while the objective (the reward) was defined in a quite straightforward manner. Behaviour acquired by agents, based on visual inspection, resembles a human one and it is quite realistic.

5.6.2 Introduction of Time Incentive and Reward Sharing

After realising that the initial setup is highly cooperative, more competitiveness was introduced by making the rewards dependent on the time needed to reach the goal. Same as in the previous experiments, a sparse reward mechanism that was non-zero only at the end of the episode was utilised, when the agent arrived at the destination. In case of success, the agent's driving time (t_d) and the length of the reference route, d_{ref}, were used to calculate the effective average velocity, V_e. In the case of crossroads scenarios, the reference route has been defined as driving in a straight line from the original position to the crossroad centre and from there driving to the destination. Normalisation of the reward has been achieved by dividing the average speed by the assumed high bound for it V_{ref}, which in the test scenarios was set to 5 m/s. Such a definition allowed one to keep a reward within a normalised range and promote arriving at the destination in the shortest time.

$$r = \frac{V_\mathrm{e}}{V_\mathrm{ref}}, \text{ where } V_\mathrm{e} = d_\mathrm{ref}/t_\mathrm{d} \tag{5.11}$$

Furthermore, to steer the level of cooperation between agents, the reward sharing mechanism was introduced, similar to the one introduced in [91]. Reward sharing allows weighting individual agent rewards, r_i, with the average reward of the team $\bar{r}$ with the team spirit parameter τ, where $\tau \in <0;1>$ (see Equation 5.12).

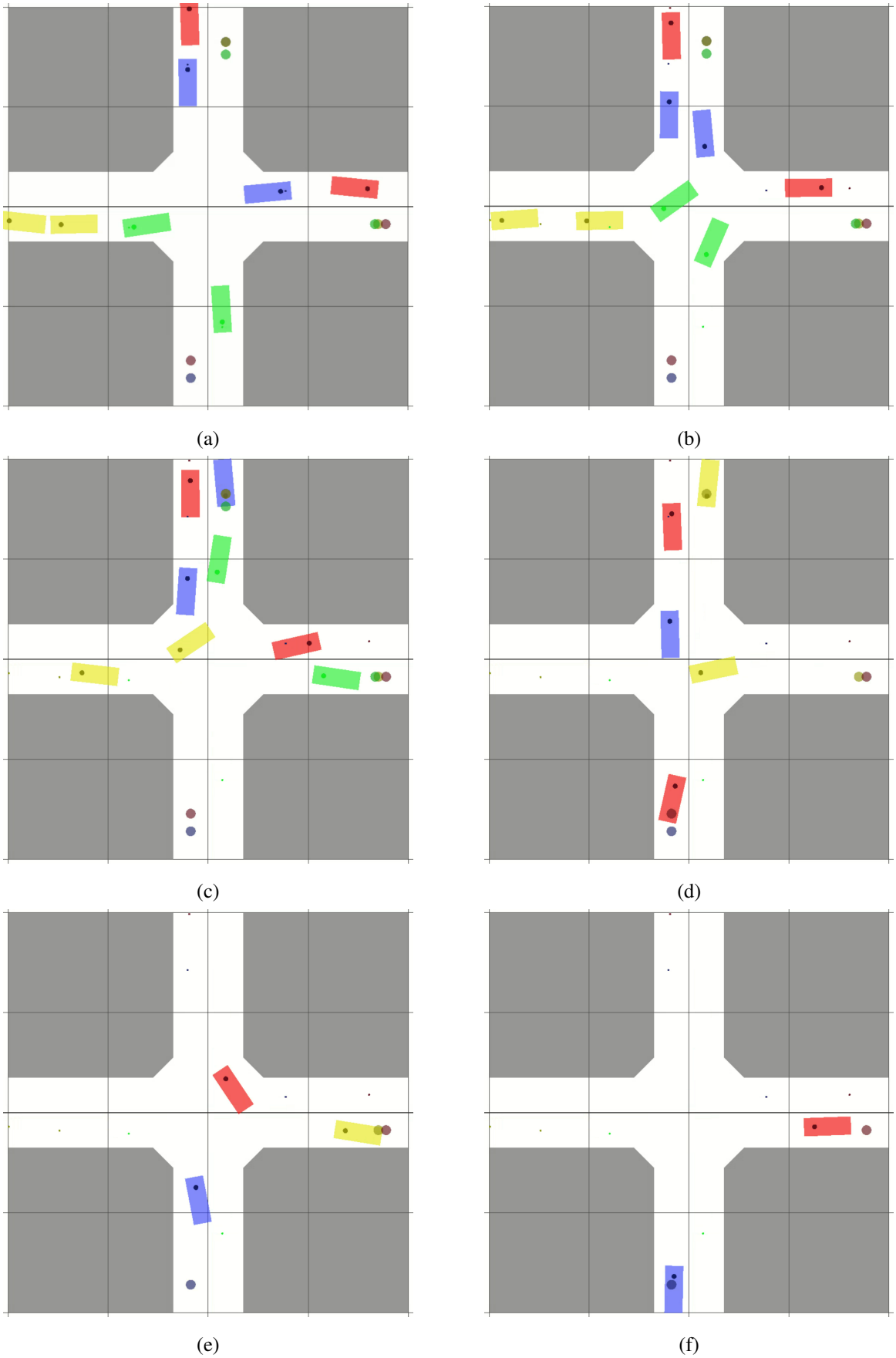


Figure 5.7: Evolution of episode for one of the crossroad scenarios.

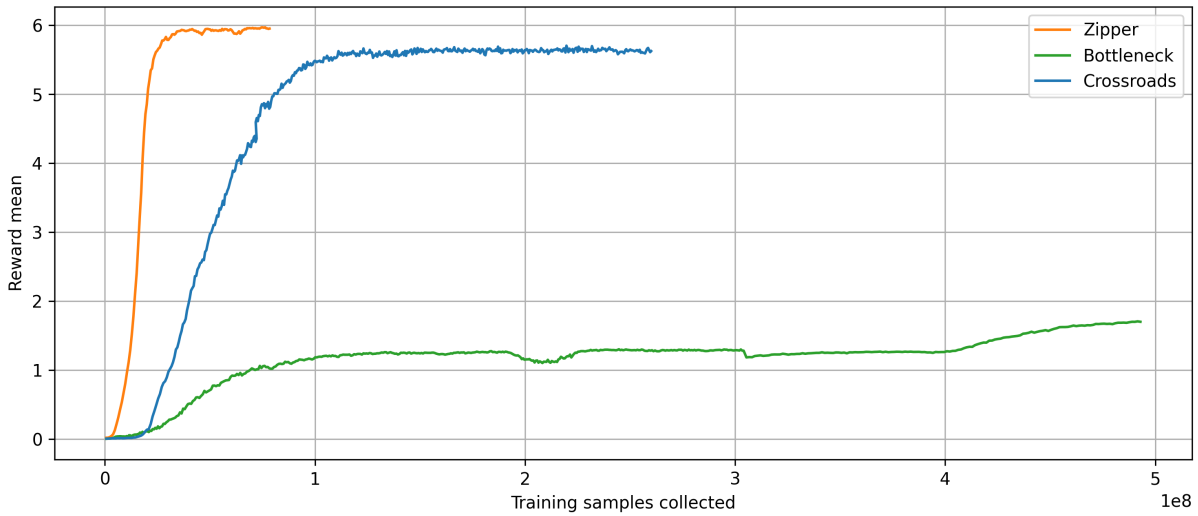


Figure 5.8: Reward mean progression for tree trained scenarios. Note: the target value of mean reward for each of the scenario is different as its depends on the mean number of agents simulated in the scene.

$$r_i^f = (1-\tau)r_i + \tau\bar{r}\,, \tag{5.12}$$

In the experiments, the assumption has been made that all the agents belong to the same team and share the rewards with all the others. Additionally, since agents are terminated at different moments of the episode and use a sparse reward mechanism, to effectively share the reward, the publication of their terminal state along with the reward was delayed until the termination of all agents. With that, terminated agents are removed from the scene as previously, but that fact is not reported right at this moment. When all agents do terminate, all rewards are recalculated with the reward sharing mechanism, and the final tuples are published to the trainer process.

To compare the effects of the introduced reward and the reward sharing mechanism, experiments were performed in crossroad scenarios. The performance of the policy trained with and without reward sharing, as well as the baseline training without time-dependent reward (see Section 5.6.1) was compared. Regarding the reward sharing mechanism, the constant value of team spirit $\tau = 0.5$ was used. Initially, a value of 1.0 was tested (agent rewarded only for team performance), however, the policy did not converge at all, which is expected behaviour with such a weak relationship between own behaviour and reward. The rewards graph has been presented in Figure 5.9, suggesting that reward sharing improves policy performance.

To analyse the policies characteristics in detail, an evaluation on 10,000 episodes resulting in around 563,000 agent trajectories has been performed. These results are presented in Table 5.3. Predictably, the Baseline training presented the highest performance in achieving the goal and caused fewer collisions, while Timed and Timed with reward sharing policies acquired higher average velocities and smoother behaviours, with a slight advantage of the second (see Figure 5.10 for a histogram of agents' average velocities). The behaviour of the agents, depending on the number of agents that have been simulated in the episode, with respect to their average velocities and their performance in achieving goals, has also been made (Figure 5.11). The results suggest that the reward-sharing mechanism has been especially helpful in achieving higher

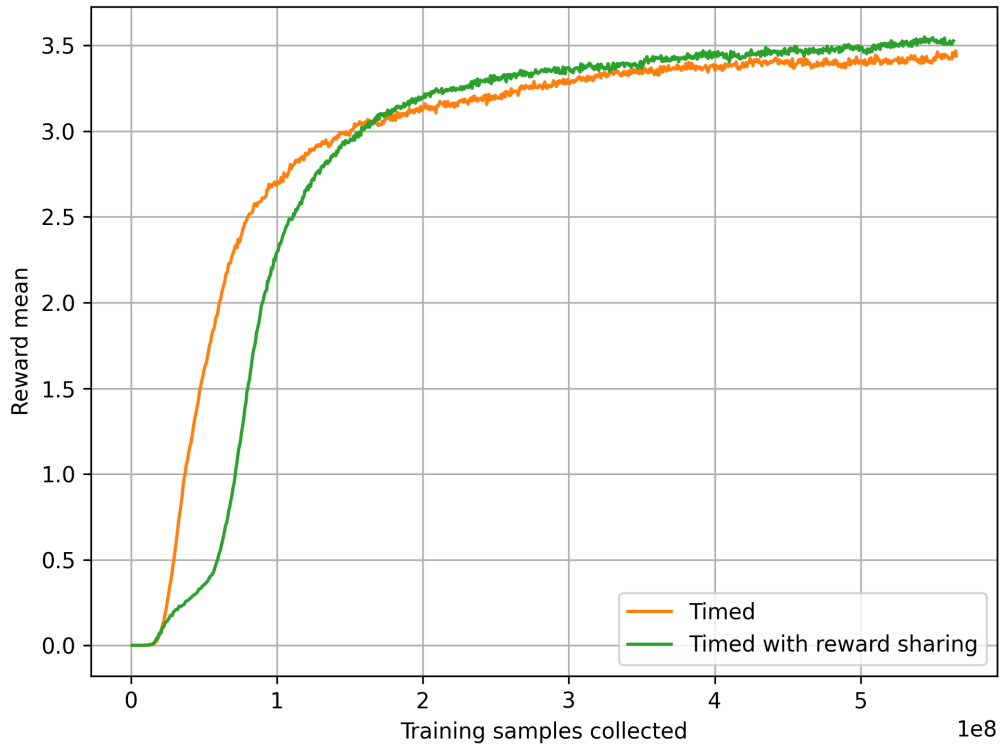


Figure 5.9: Average reward graph showing the progress of training. The introduced reward-sharing mechanism slows down progress in the beginning but is able to achieve better final performance.

Table 5.3 Table presents performance evaluation for three crossroads setups: Baseline, with reward not taking into consideration time, Timed with such incentive and Timed with shared reward, where additionally performance of all agents has been shared. The values which relate to episode duration, speed, and acceleration have been only calculated for agents successfully arriving at the destination.

	Baseline	**Timed**	**Timed with Reward Sharing**
Goal reached [%]	99.5	96.9	97.65
Obstacle collision [%]	0.12	0.43	0.24
Agent collision [%]	0.32	2.73	2.16
Avg episode length	31.08	23.33	22.86
Avg speed [$\mathrm{m/s}$]	1.9	2.566	2.584
Max speed [$\mathrm{m/s}$]	3.41	5.21	5.761
Min speed [$\mathrm{m/s}$]	0.52	1.01	1.032
Static in episode [%]	13.23	6.14	6.34
Avg sum acc [$\mathrm{m/s^2}$]	20.64	18.58	18.27
Std sum acc [$\mathrm{m/s^2}$]	0.76	0.856	0.855

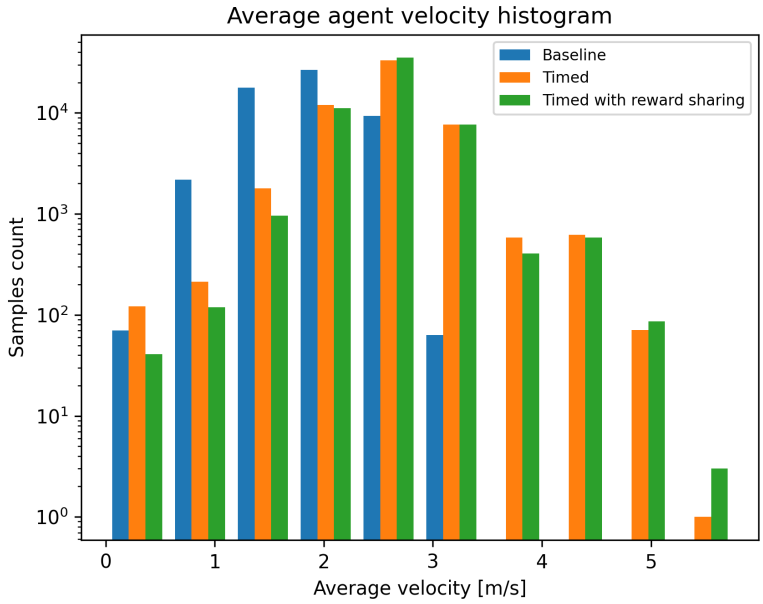


Figure 5.10: Histogram of average velocities acquired in episodes.

average speeds in crowdy scenarios, while in ones with fewer traffic the results have not provided any benefit or worsen the performance. Goal-reaching performance metrics suggest that reward sharing is helpful in all cases, most probably due to the multiplied effect of caused collisions.

5.7 Discussion and Further Work

The above experiments prove that with straightforward problem formulation, it is possible to acquire policies that perform well in road scenarios that require a lot of cooperation between road users. Additionally, since the reward design was quite straightforward, one may argue that cooperation is a natural strategy in challenging on-road scenarios. With the detrimental and equal effect of collision for all participants of such an event, both the victim and the culprit, priority number one is collision avoidance. At the same time, acquired behaviours seemed to be quite human-like, even though any direct mechanism of making them similar to the human benchmarks has been applied. Lack of traffic rules and enforcing them reward mechanisms, effected in highly cooperative and efficient, but a bit unstructured, movement patterns.

With time-dependent reward experiments, it has been shown that the reward-sharing mechanism improves cooperation between agents and yields better individual results. This effect was amplified in scenarios with many agents, where coordination played an even more important role. At the same time, the balance between average speed and safety must be carefully addressed.

Numerous possible extensions of this research are possible. From a policy optimisation perspective, a grid search for parameters such as team spirit, gamma, and reward parameters should be executed to find the best set that meets the target KPIs. Although training scenarios have been randomised and other agents' behaviour played an important role in the diversification of experiences, a more broad set of road use cases should be added to acquire more robust policies. As the training has been carried out in a self-play manner

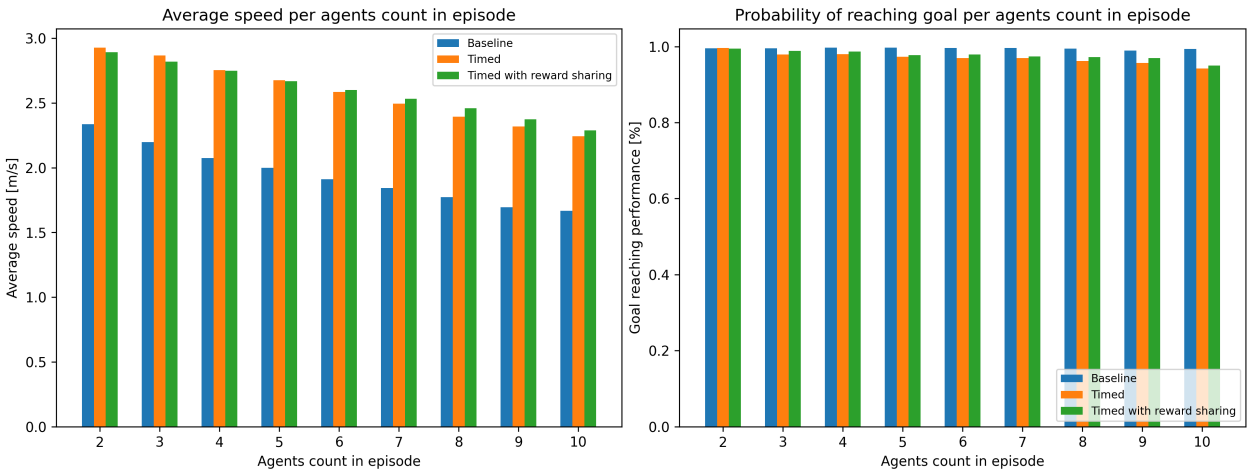


Figure 5.11: Presentation of the average speed of the agents (left Figure) and goal-reaching performance right Figure depending on the number of vehicles present in a given scenario. First, as the number of agents grows, the average velocity decreases. What is especially interesting is that the agent trained with a reward-sharing mechanism (green) improves average speed in scenarios with higher traffic (when there are more than 5 agents in the scene) and lowers it when the number of agents is smaller. Goal-achieving performance is improved by reward sharing in all scenarios, although the baseline policy (blue) is superior in all cases. Those results bring up the conclusion that the reward-sharing mechanism plays an important role in multi-agent scenarios especially when the number of agents is greater.

in a highly cooperative driving environment, the strategy of all agents assumes most likely narrow behaviour patterns of other road users and is not robust to any out-of-distribution cases. With that in mind, a robustness study followed by adaptations of the policies to other potential behavioural patterns of other road users would be an important addition to the research. Finally, safety considerations with the integration of rule-based constraints or handwritten rules into the planning mechanism (encapsulating traffic rules) would be one of the most important and challenging elements moving toward commercialisation applicability. Despite the above, the multi-agent reinforcement learning with carefully designed extensions can be successfully used as a solid base for the motion planning of highly automated vehicles.

Chapter 6

Conclusions

6.1 Key Contributions and Conclusions

Based on the experiments performed in different domains, the conclusion can be drawn that the reinforcement learning methodology is an attractive and reasonable alternative to the standard control methods used so far in the autonomous driving domain.

In the following part, key contributions of this work have been listed.

- It has been proven that a simulated autonomous car can be controlled by a high-level interface, such as a behaviour planning one, with the use of reinforcement learning methodology, which supports claim (i).

- Research proved that the introduction of a proposed deterministic mechanism during training, including a finite state machine manoeuvre, available action predefinition mechanism, and trajectory generation, results in better end performance and faster convergence compared to doing so after training. This supports claim (ii).

- It has been confirmed that introduction of deterministic rules decreases the transparency of the system from reinforcement learning agent perspective, therefore such integration needs to be done with care.

- As part of a collaborative effort, a traffic simulator applicable for the autonomous driving reinforcement learning application has been created.

- It has been proven that controlling a vehicle in parking scenarios with a low-level control interface by reinforcement learning policies is possible, supporting claim (iii).

- Comparison of two observation models and associated neural network architectures with them has been performed, showing benefits and drawbacks of both solutions.

- The integration of the RL-based parking application within the car test system has been carried out, proving the real-time potential of the proposed solution and the applicability to real-world data.

- The reinforcement learning methodology has been successfully used to coordinate the movement of multiple agents in city-like scenarios, which required careful coordination of all agents' behaviour, supporting claim (iv).
- The proposed reward sharing mechanism with a straightforward definition of the reward function improved the effectiveness of training and the resulting functional performance for all agents, which proved claim (v).
- The research conducted resulted in the implementation of a set of reinforcement learning environments which can be further customised and reused to perform new experiments.

To summarise, author argues that reinforcement learning methodologies are applicable to autonomous driving problem, however, due to high complexity and safety requirements, introduction of them to real-world system will require further development, careful analysis and validation accordingly to all automotive standards. Based on the examination, parking setup was the most successful trial to introduce the reinforcement learning methodology to a given autonomous driving problem and has the greatest industrialisation potential. All resulting models and solutions created within work described in this work are considered part of the Aptiv product portfolio and will be considered in future business pursuits.

6.2 Looking Behind and Ahead

Looking at the research performed from the perspective of time, it has been concluded that some of the research elements could be done better. Early expansion of computation capabilities was crucial to achieve a stable optimisation process, and having a local cluster significantly reduced optimisation costs. At the same time, utilising cloud environments would accelerate activities much faster and reduce the time spent managing on-premise resources. The prioritisation of a clear and precise validation and evaluation mechanism should be prioritised almost from the beginning of training, as reinforcement learning problems in autonomous driving are complex and cannot be clearly evaluated just by looking at the behaviour of the vehicles.

Looking ahead, many potential ways of continuing research were identified both in terms of increasing its capability and adapting it to a production environment. Multiple ways of progressing the research in the corresponding conclusions chapter were already provided, therefore, most general ones will be reiterated. First and foremost, the utilisation of real-world data to close the gap between simulation and real-world data will be crucial to make the solution resilient for a potential production environment. Within that, simulating imperfect perception data and, therefore, training resilient reinforcement learning policies is an attractive research direction. To successfully work on real-world data, production-grade application will require extensive testing and evaluation and strong statistical reasoning to prove potential effectiveness in real-world examples. The problem of reward definition in direct control of reinforcement learning agent behaviour should be studied in depth as well. Potential connection with methods allowing for behaviour cloning, which could allow to mimic desired behaviours, would be crucial to introduce more control over agent behaviour. Although clear benefits of doing pure behaviour cloning can be seen, optimising behaviour by reinforcement

learning means is still a crucial part of artificial intelligence-driven autonomous driving system. In summary, research on architecture that would combine reinforcement learning, supervised learning, and model-based methods is highly appreciated. Exploring the use of other reinforcement learning algorithms, including ones such as DQN [76] or AlphaZero [122] would be equally beneficial, as it will provide less unimodal policies.

Bibliography

[1] *(R) Taxonomy and Definitions for Terms Related to Driving Automation Systems for On-Road Motor Vehicles*. Tech. rep. Geneva, CH: SAE International, Apr. 2021. DOI: https://doi.org/10.4271/J3016{_}202104.

[2] Andrew Jazwinski. *Stochastic Processes and Filtering Theory, Volume 64 1st Edition*. San Diego: Academic Press, 1970, p. 370. ISBN: 9780080960906.

[3] Marcin Andrychowicz et al. "Hindsight Experience Replay". In: *Advances in Neural Information Processing Systems* 2017-December (July 2017), pp. 5049–5059. ISSN: 10495258. DOI: 10.48550/arxiv.1707.01495. URL: https://arxiv.org/abs/1707.01495v3.

[4] M. Sanjeev Arulampalam, Simon Maskell, Neil Gordon, and Tim Clapp. "A tutorial on particle filters for online nonlinear/non-Gaussian Bayesian tracking". In: *IEEE Transactions on Signal Processing* 50.2 (Feb. 2002), pp. 174–188. ISSN: 1053587X. DOI: 10.1109/78.978374.

[5] *ASAM OpenDRIVE® v1.7*. Tech. rep. ASAM, 2021. URL: https://www.asam.net/index.php?eID=dumpFile&t=f&f=4422&token=e590561f3c39aa2260e5442e29e93f6693d1cccd.

[6] Peter Auer, Nicolò Cesa-Bianchi, and Paul Fischer. "Finite-time Analysis of the Multiarmed Bandit Problem". In: *Machine Learning 2002 47:2* 47.2 (May 2002), pp. 235–256. ISSN: 1573-0565. DOI: 10.1023/A:1013689704352. URL: https://link.springer.com/article/10.1023/A:1013689704352.

[7] Bowen Baker et al. "Emergent Tool Use From Multi-Agent Autocurricula". In: (Sept. 2019). DOI: 10.48550/arxiv.1909.07528. URL: https://arxiv.org/abs/1909.07528v2.

[8] Mayank Bansal, Alex Krizhevsky, and Abhijit Ogale. "ChauffeurNet: Learning to Drive by Imitating the Best and Synthesizing the Worst". In: *Robotics: Science and Systems* (Dec. 2019). ISSN: 2330765X. DOI: 10.15607/RSS.2019.XV.031. URL: https://arxiv.org/abs/1812.03079v1.

[9] Gabriel Barth-Maron et al. "Distributed Distributional Deterministic Policy Gradients". In: *6th International Conference on Learning Representations, ICLR 2018 - Conference Track Proceedings* (Apr. 2018). DOI: 10.48550/arxiv.1804.08617. URL: https://arxiv.org/abs/1804.08617v1.

[10] Andrea Bassich, Francesco Foglino, Matteo Leonetti, and Daniel Kudenko. "Curriculum Learning with a Progression Function". In: (Aug. 2020). DOI: 10.48550/arxiv.2008.00511. URL: https://arxiv.org/abs/2008.00511v2.

[11] Eduardo Bejar and Antonio Moran. "Reverse Parking a Car-Like Mobile Robot with Deep Reinforcement Learning and Preview Control". In: *2019 IEEE 9th Annual Computing and Communication Workshop and Conference (CCWC)* (Mar. 2019), pp. 377–383. DOI: 10.1109/CCWC.2019.8666613.

[12] Marc G. Bellemare, Will Dabney, and Rémi Munos. "A Distributional Perspective on Reinforcement Learning". In: *34th International Conference on Machine Learning, ICML 2017* 1 (July 2017), pp. 693–711. DOI: 10.48550/arxiv.1707.06887. URL: https://arxiv.org/abs/1707.06887v1.

[13] Richard Bellman. *Dynamic programing*. Princeton: Princeton University Press, 1957. ISBN: 069107951X.

[14] Yoshua Bengio, Jérome Louradour, Ronan Collobert, and Jason Weston. "Curriculum learning". In: *International Conference on Machine Learning* 382 (2009). DOI: 10.1145/1553374.1553380.

[15] Dimitri P. Bertsekas. *Dynamic programming and optimal control*. Vol. I. 2017. ISBN: 1-886529-43-4.

[16] Dimitri P. Bertsekas. *Dynamic programming and optimal control: Approximate Dynamic Programming*. Vol. II. 2012. ISBN: 978-1-886529-44-1.

[17] Greg Brockman et al. "OpenAI Gym". In: *Arxiv* (June 2016). DOI: 10.48550/arxiv.1606.01540. URL: https://arxiv.org/abs/1606.01540v1.

[18] Rodney A. Brooks. "A Robust Layered Control System For A Mobile Robot". In: *IEEE Journal on Robotics and Automation* 2.1 (1986), pp. 14–23. ISSN: 08824967. DOI: 10.1109/JRA.1986.1087032.

[19] Holger Caesar et al. "Nuscenes: A multimodal dataset for autonomous driving". In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition* (Mar. 2020), pp. 11618–11628. ISSN: 10636919. DOI: 10.1109/CVPR42600.2020.01164. URL: https://arxiv.org/abs/1903.11027v5.

[20] Micah Carroll et al. "On the Utility of Learning about Humans for Human-AI Coordination". In: *Advances in Neural Information Processing Systems* 32 (Oct. 2019). ISSN: 10495258. DOI: 10.48550/arxiv.1910.05789. URL: https://arxiv.org/abs/1910.05789v2.

[21] Chao Chen, Markus Rickert, and Alois Knoll. "Path planning with orientation-aware space exploration guided heuristic search for autonomous parking and maneuvering". In: *2015 IEEE Intelligent Vehicles Symposium (IV)* 2015-August (Aug. 2015), pp. 1148–1153. DOI: 10.1109/IVS.2015.7225838.

[22] Dong Chen et al. “Deep Multi-Agent Reinforcement Learning for Highway On-Ramp Merging in Mixed Traffic”. In: *IEEE Transactions on Intelligent Transportation Systems* (2023). ISSN: 15580016. DOI: 10.1109/TITS.2023.3285442.

[23] Jianyu Chen, Bodi Yuan, and Masayoshi Tomizuka. “Model-free Deep Reinforcement Learning for Urban Autonomous Driving”. In: *2019 IEEE Intelligent Transportation Systems Conference (ITSC)* (Oct. 2019), pp. 2765–2771. DOI: 10.1109/ITSC.2019.8917306.

[24] Xuemei Chen et al. “A conflict decision model based on game theory for intelligent vehicles at urban unsignalized intersections”. In: *IEEE Access* 8 (2020), pp. 189546–189555. ISSN: 21693536. DOI: 10.1109/ACCESS.2020.3031674.

[25] Filippos Christianos, Lukas Schäfer, and Stefano V. Albrecht. “Shared Experience Actor-Critic for Multi-Agent Reinforcement Learning”. In: *Advances in Neural Information Processing Systems* 2020-December (June 2020). ISSN: 10495258. DOI: 10.48550/arxiv.2006.07169. URL: https://arxiv.org/abs/2006.07169v4.

[26] Tianshu Chu, Jie Wang, Lara Codeca, and Zhaojian Li. “Multi-Agent Deep Reinforcement Learning for Large-Scale Traffic Signal Control”. In: *IEEE Transactions on Intelligent Transportation Systems* 21.3 (Mar. 2019), pp. 1086–1095. ISSN: 15580016. DOI: 10.1109/TITS.2019.2901791.

[27] Paweł Cichosz. “Truncating Temporal Diierences: On the EEcient Implementation of TD for Reinforcement Learning”. In: *Journal of Artiicial Intelligence Research* 2 (1995), pp. 287–318.

[28] Laurene Claussmann, Marc Revilloud, Dominique Gruyer, and Sebastien Glaser. “A Review of Motion Planning for Highway Autonomous Driving”. In: *IEEE Transactions on Intelligent Transportation Systems* 21.5 (May 2020), pp. 1826–1848. ISSN: 15580016. DOI: 10.1109/TITS.2019.2913998.

[29] *Cruise’s Safety Record Over 1 Million Driverless Miles | Cruise*. URL: https://getcruise.com/news/blog/2023/cruises-safety-record-over-one-million-driverless-miles/.

[30] Jonas Degrave et al. “Magnetic control of tokamak plasmas through deep reinforcement learning”. In: *Nature 2022 602:7897* 602.7897 (Feb. 2022), pp. 414–419. ISSN: 1476-4687. DOI: 10.1038/s41586-021-04301-9. URL: https://www.nature.com/articles/s41586-021-04301-9.

[31] *Delphi Successfully Completes First Coast-to-Coast Automated Drive*. URL: https://www.aptiv.com/en/newsroom/article/delphi-successfully-completes-first-coast-to-coast-automated-drive.

[32] Nachiket Deo, Eric Wolff, and Oscar Beijbom. “Multimodal Trajectory Prediction Conditioned on Lane-Graph Traversals”. In: *Proceedings of the 5th Conference on Robot Learning*. Ed. by Aleksandra Faust, David Hsu, and Gerhard Neumann. Vol. 164. Proceedings of Machine Learning Research. PMLR, July 2022, pp. 203–212. URL: https://proceedings.mlr.press/v164/deo22a.html.

[33] *Detecting Obstacles and Drivable Free Space with RadarNet | NVIDIA Technical Blog*. URL: https://developer.nvidia.com/blog/detecting-obstacles-and-drivable-free-space-with-radarnet/.

[34] Joris Dinneweth, Abderrahmane Boubezoul, René Mandiau, and Stéphane Espié. "Multi-agent reinforcement learning for autonomous vehicles: a survey". In: *Autonomous Intelligent Systems* 2.1 (Nov. 2022), p. 27. ISSN: 2730-616X. DOI: 10.1007/S43684-022-00045-Z. URL: https://link.springer.com/10.1007/s43684-022-00045-z.

[35] Dmitri Dolgov, Sebastian Thrun, Michael Montemerlo, and James Diebel. "Path Planning for Autonomous Vehicles in Unknown Semi-structured Environments". In: *The International Journal of Robotics Research* 29.5 (Apr. 2010), pp. 485–501. ISSN: 02783649. DOI: 10.1177/0278364909359210.

[36] Ziyue Feng, Shitao Chen, Yu Chen, and Nanning Zheng. "Model-Based Decision Making with Imagination for Autonomous Parking". In: *IEEE Intelligent Vehicles Symposium, Proceedings* 2018-June (Oct. 2018), pp. 2216–2223. DOI: 10.1109/IVS.2018.8500700. URL: https://arxiv.org/abs/2108.11420v1.

[37] *First internationally valid system approval: Conditionally automated driving | Mercedes-Benz Group > Innovation > Product innovation > Autonomous driving*. URL: https://group.mercedes-benz.com/innovation/product-innovation/autonomous-driving/system-approval-for-conditionally-automated-driving.html.

[38] Meire Fortunato et al. "Noisy Networks for Exploration". In: *6th International Conference on Learning Representations, ICLR 2018 - Conference Track Proceedings* (June 2017). DOI: 10.48550/arxiv.1706.10295. URL: https://arxiv.org/abs/1706.10295v3.

[39] Gene F Franklin, J David Powell, and Abbas Emami-Naeini. *Feedback Control of Dynamic Systems (8th Edition) (What's New in Engineering)*. Pearson, 2018. ISBN: 0134685717. URL: https://www.amazon.com/Feedback-Control-Dynamic-Systems-Engineering/dp/0134685717?SubscriptionId=AKIAIOBINVZYXZQZ2U3A&tag=chimbori05-20&linkCode=xm2&camp=2025&creative=165953&creativeASIN=0134685717.

[40] Lina Fu, Ahmet Yazici, and Ümit Ozgüner. "Route planning for OSU-ACT autonomous vehicle in DARPA Urban Challenge". In: *IEEE Intelligent Vehicles Symposium, Proceedings* (2008), pp. 781–786. DOI: 10.1109/IVS.2008.4621279.

[41] Scott Fujimoto, Herke Van Hoof, and David Meger. "Addressing Function Approximation Error in Actor-Critic Methods". In: *35th International Conference on Machine Learning, ICML 2018* 4 (Feb. 2018), pp. 2587–2601. DOI: 10.48550/arxiv.1802.09477. URL: https://arxiv.org/abs/1802.09477v3.

[42] David González, Joshué Pérez, Vicente Milanés, and Fawzi Nashashibi. "A Review of Motion Planning Techniques for Automated Vehicles". In: *IEEE Transactions on Intelligent Transportation Systems* 17.4 (Apr. 2016), pp. 1135–1145. ISSN: 15249050. DOI: 10.1109/TITS.2015.2498841.

[43] Ralph Grewe, Andree Hohm, Stefan Lüke, and Hermann Winner. "Environment Models as Standardised Interface for Future ADAS". In: *ATZelektronik worldwide 2012 7:5* 7.5 (Oct. 2012), pp. 10–15. ISSN: 2192-9092. DOI: 10.1365/S38314-012-0110-5. URL: https://link.springer.com/article/10.1365/s38314-012-0110-5.

[44] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. "Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor". In: *35th International Conference on Machine Learning, ICML 2018* 5 (Jan. 2018), pp. 2976–2989. DOI: 10.48550/arxiv.1801.01290. URL: https://arxiv.org/abs/1801.01290v2.

[45] Tuomas Haarnoja et al. "Soft Actor-Critic Algorithms and Applications". In: (Dec. 2018). DOI: 10.48550/arxiv.1812.05905. URL: https://arxiv.org/abs/1812.05905v2.

[46] Peng Hang, Chen Lv, Chao Huang, Yang Xing, and Zhongxu Hu. "Cooperative Decision Making of Connected Automated Vehicles at Multi-Lane Merging Zone: A Coalitional Game Approach". In: *IEEE Transactions on Intelligent Transportation Systems* 23.4 (Apr. 2022), pp. 3829–3841. ISSN: 15580016. DOI: 10.1109/TITS.2021.3069463. URL: https://arxiv.org/abs/2103.07887v1.

[47] Matteo Hessel et al. "Rainbow: Combining Improvements in Deep Reinforcement Learning". In: *32nd AAAI Conference on Artificial Intelligence, AAAI 2018* (Oct. 2017), pp. 3215–3222. DOI: 10.48550/arxiv.1710.02298. URL: https://arxiv.org/abs/1710.02298v1.

[48] *Honda Global | November 11 , 2020 "Honda Receives Type Designation for Level 3 Automated Driving in Japan"*. URL: https://global.honda/newsroom/news/2020/4201111eng.html.

[49] Hsu Chieh Hu, Stephen F. Smith, and Rick Goldstein. "Cooperative Schedule-Driven Intersection Control with Connected and Autonomous Vehicles". In: *IEEE International Conference on Intelligent Robots and Systems* (Nov. 2019), pp. 1668–1673. ISSN: 21530866. DOI: 10.1109/IROS40897.2019.8967975. URL: https://arxiv.org/abs/1907.01984v1.

[50] Tommi Jaakkola, Satinder P. Singh, and Michael I. Jordan. "Reinforcement learning algorithm for partially observable Markov decision problems". In: *NIPS'94: Proceedings of the 7th International Conference on Neural Information Processing Systems*. 1994, pp. 345–352. URL: https://dl.acm.org/doi/10.5555/2998687.2998730.

[51] Jyun Hao Jhang and Feng Li Lian. "An Autonomous Parking System of Optimally Integrating Bidirectional Rapidly-Exploring Random Trees* and Parking-Oriented Model Predictive Control". In: *IEEE Access* 8 (2020), pp. 163502–163523. ISSN: 21693536. DOI: 10.1109/ACCESS.2020.3020859.

[52] Simon J. Julier and Jeffrey K. Uhlmann. "Unscented filtering and nonlinear estimation". In: *Proceedings of the IEEE* 92.3 (Mar. 2004), pp. 401–422. ISSN: 00189219. DOI: 10.1109/JPROC.2003.823141.

[53] Rudolf Kalman. “A new approach to linear filtering and prediction problems”. In: *Journal of Fluids Engineering, Transactions of the ASME* 82.1 (1960), pp. 35–45. ISSN: 1528901X. DOI: 10.1115/1.3662552.

[54] Malvin H. Kalos and Paula A. Whitlock. *Monte Carlo Methods: Second Edition*. Wiley-VCH, July 2009, pp. 1–203. ISBN: 9783527407606. DOI: 10.1002/9783527626212. URL: https://onlinelibrary.wiley.com/doi/book/10.1002/9783527626212.

[55] Takeo Kanade, Chuck Thorpe, and William (Red) L Whittaker. “Autonomous Land Vehicle Project at CMU”. In: *Proceedings of ACM 14th Annual Conference on Computer Science (CSC ’86)*. Feb. 1986, pp. 71–80.

[56] Alexander Katriniok, Peter Kleibaum, and Martina Joševski. “Distributed Model Predictive Control for Intersection Automation Using a Parallelized Optimization Approach”. In: *IFAC-PapersOnLine* 50.1 (July 2017), pp. 5940–5946. ISSN: 2405-8963. DOI: 10.1016/J.IFACOL.2017.08.1492.

[57] B. Ravi Kiran et al. “Deep Reinforcement Learning for Autonomous Driving: A Survey”. In: *IEEE Transactions on Intelligent Transportation Systems* 23.6 (June 2020), pp. 4909–4926. ISSN: 15580016. DOI: 10.1109/TITS.2021.3054625.

[58] Sebastian Klemm et al. “Autonomous multi-story navigation for valet parking”. In: *IEEE Conference on Intelligent Transportation Systems, Proceedings, ITSC* (Dec. 2016), pp. 1126–1133. DOI: 10.1109/ITSC.2016.7795698.

[59] János Kramár et al. “Negotiation and honesty in artificial intelligence methods for the board game of Diplomacy”. In: *Nature Communications 2022 13:1* 13.1 (Dec. 2022), pp. 1–15. ISSN: 2041-1723. DOI: 10.1038/s41467-022-34473-5. URL: https://www.nature.com/articles/s41467-022-34473-5.

[60] Karol Kurach et al. “Google Research Football: A Novel Reinforcement Learning Environment”. In: *AAAI 2020 - 34th AAAI Conference on Artificial Intelligence* (July 2019), pp. 4501–4510. ISSN: 2159-5399. DOI: 10.48550/arxiv.1907.11180. URL: https://arxiv.org/abs/1907.11180v2.

[61] Stéphanie Lefèvre, Dizan Vasquez, and Christian Laugier. “A survey on motion prediction and risk assessment for intelligent vehicles”. In: *ROBOMECH Journal* 1.1 (Dec. 2014), pp. 1–14. ISSN: 21974225. DOI: 10.1186/S40648-014-0001-Z/FIGURES/8. URL: https://robomechjournal.springeropen.com/articles/10.1186/s40648-014-0001-z.

[62] Bai Li, Youmin Zhang, Yue Zhang, Ning Jia, and Yuming Ge. “Near-Optimal Online Motion Planning of Connected and Automated Vehicles at a Signal-Free and Lane-Free Intersection”. In: *IEEE Intelligent Vehicles Symposium, Proceedings* 2018-June (Oct. 2018), pp. 1432–1437. DOI: 10.1109/IVS.2018.8500528.

[63] Changjian Li and Krzysztof Czarnecki. "Urban Driving with Multi-Objective Deep Reinforcement Learning". In: *Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems, AAMAS* 1 (Nov. 2018), pp. 359–367. ISSN: 15582914. URL: https://arxiv.org/abs/1811.08586v2.

[64] Nan Li, Yu Yao, Ilya Kolmanovsky, Ella Atkins, and Anouck R. Girard. "Game-Theoretic Modeling of Multi-Vehicle Interactions at Uncontrolled Intersections". In: *IEEE Transactions on Intelligent Transportation Systems* 23.2 (Feb. 2022), pp. 1428–1442. ISSN: 15580016. DOI: 10.1109/TITS.2020.3026160. URL: https://arxiv.org/abs/1904.05423v1.

[65] Eric Liang et al. "RLlib: Abstractions for Distributed Reinforcement Learning". In: *35th International Conference on Machine Learning, ICML 2018* 7 (Dec. 2017), pp. 4768–4780. DOI: 10.48550/arxiv.1712.09381. URL: https://arxiv.org/abs/1712.09381v4.

[66] Timothy P. Lillicrap et al. "Continuous control with deep reinforcement learning". In: *4th International Conference on Learning Representations, ICLR 2016 - Conference Track Proceedings* (Sept. 2015). DOI: 10.48550/arxiv.1509.02971. URL: https://arxiv.org/abs/1509.02971v6.

[67] Xiao Lin et al. "Decision Making through Occluded Intersections for Autonomous Driving". In: *2019 IEEE Intelligent Transportation Systems Conference, ITSC 2019* (Oct. 2019), pp. 2449–2455. DOI: 10.1109/ITSC.2019.8917348.

[68] Chang Liu, Seungho Lee, Scott Varnhagen, and H. Eric Tseng. "Path planning for autonomous vehicles using model predictive control". In: *2017 IEEE Intelligent Vehicles Symposium (IV)* (July 2017), pp. 174–179. DOI: 10.1109/IVS.2017.7995716.

[69] Ryan Lowe et al. "Multi-Agent Actor-Critic for Mixed Cooperative-Competitive Environments". In: *Advances in Neural Information Processing Systems* 2017-December (June 2017), pp. 6380–6391. ISSN: 10495258. DOI: 10.48550/arxiv.1706.02275. URL: https://arxiv.org/abs/1706.02275v4.

[70] Kevin M Lynch and Frank C Park. *MODERN ROBOTICS MECHANICS, PLANNING, AND CONTROL*. Cambridge: Cambridge University Press, 2017. ISBN: 9781107156302. URL: http://modernrobotics.org.Commentsarewelcome!.

[71] Pattie Maes. "Do the right thing." In: *Nursing standard (Royal College of Nursing (Great Britain) : 1987)* 1.3 (Jan. 1989), pp. 291–323. ISSN: 13600494. DOI: 10.1080/09540098908915643.

[72] Daniel J. Mankowitz et al. "Faster sorting algorithms discovered using deep reinforcement learning". In: *Nature 2023 618:7964* 618.7964 (June 2023), pp. 257–263. ISSN: 1476-4687. DOI: 10.1038/s41586-023-06004-9. URL: https://www.nature.com/articles/s41586-023-06004-9.

[73] Martín Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. 2015. URL: https://www.tensorflow.org/.

[74] Branka Mirchevska, Christian Pek, Moritz Werling, Matthias Althoff, and Joschka Boedecker. "High-level Decision Making for Safe and Reasonable Autonomous Lane Changing using Reinforcement Learning". In: *IEEE Conference on Intelligent Transportation Systems, Proceedings, ITSC* 2018-November (Dec. 2018), pp. 2156–2162. DOI: 10.1109/ITSC.2018.8569448.

[75] Volodymyr Mnih et al. "Asynchronous Methods for Deep Reinforcement Learning". In: *33rd International Conference on Machine Learning, ICML 2016* 4 (Feb. 2016), pp. 2850–2869. DOI: 10.48550/arxiv.1602.01783. URL: https://arxiv.org/abs/1602.01783v2.

[76] Volodymyr Mnih et al. "Human-level control through deep reinforcement learning". In: *Nature 2015 518:7540* 518.7540 (Feb. 2015), pp. 529–533. ISSN: 1476-4687. DOI: 10.1038/nature14236. URL: https://www.nature.com/articles/nature14236.

[77] Volodymyr Mnih et al. "Playing Atari with Deep Reinforcement Learning". In: *Arxiv* (Dec. 2013). DOI: 10.48550/arxiv.1312.5602. URL: https://arxiv.org/abs/1312.5602v1.

[78] *Mobileye REM™ - Road Experience Management*. URL: https://www.mobileye.com/technology/rem/.

[79] Sharada Mohanty et al. "Flatland-RL : Multi-Agent Reinforcement Learning on Trains". In: (Dec. 2020). DOI: 10.48550/arxiv.2012.05893. URL: https://arxiv.org/abs/2012.05893v2.

[80] George E. Monahan. "State of the Art—A Survey of Partially Observable Markov Decision Processes: Theory, Models, and Algorithms". In: *http://dx.doi.org/10.1287/mnsc.28.1.1* 28.1 (Jan. 1982), pp. 1–16. ISSN: 00251909. DOI: 10.1287/MNSC.28.1.1. URL: https://pubsonline.informs.org/doi/abs/10.1287/mnsc.28.1.1.

[81] Michael Montemerlo et al. "Junior: The stanford entry in the urban challenge". In: *Springer Tracts in Advanced Robotics* 56 (2009), pp. 91–123. ISSN: 16107438. DOI: 10.1007/978-3-642-03991-1{_}3/COVER. URL: https://link.springer.com/chapter/10.1007/978-3-642-03991-1_3.

[82] Alejandro Ivan Morales Medina, Nathan Van De Wouw, and Henk Nijmeijer. "Cooperative Intersection Control Based on Virtual Platooning". In: *IEEE Transactions on Intelligent Transportation Systems* 19.6 (June 2018), pp. 1727–1740. ISSN: 15249050. DOI: 10.1109/TITS.2017.2735628.

[83] Igor Mordatch and Pieter Abbeel. "Emergence of Grounded Compositional Language in Multi-Agent Populations". In: *32nd AAAI Conference on Artificial Intelligence, AAAI 2018* (Mar. 2017), pp. 1495–1502. DOI: 10.48550/arxiv.1703.04908. URL: https://arxiv.org/abs/1703.04908v2.

[84] *Motional Launches First Robotaxi Service on the Uber Network | Motional*. URL: https://motional.com/news/motional-launches-first-robotaxi-service-uber-network.

[85] Rémi Munos, Thomas Stepleton, Anna Harutyunyan, and Marc G. Bellemare. "Safe and Efficient Off-Policy Reinforcement Learning". In: *Advances in Neural Information Processing Systems* (June 2016), pp. 1054–1062. ISSN: 10495258. DOI: 10.48550/arxiv.1606.02647. URL: https://arxiv.org/abs/1606.02647v2.

[86] Raul Mur-Artal and Juan D. Tardos. "ORB-SLAM2: an Open-Source SLAM System for Monocular, Stereo and RGB-D Cameras". In: *IEEE Transactions on Robotics* 33.5 (Oct. 2016), pp. 1255–1262. DOI: 10.1109/TRO.2017.2705103. URL: http://arxiv.org/abs/1610.06475%20http://dx.doi.org/10.1109/TRO.2017.2705103.

[87] Kevin P. Murphy. *A Survey of POMDP Solution Techniques | Semantic Scholar*. Tech. rep. 2007. URL: https://www.semanticscholar.org/paper/A-Survey-of-POMDP-Solution-Techniques-Murphy/43ac7c95bce9c9d1417c45b7c5b76b6c88fcb633.

[88] Subramanya Nageshrao, H. Eric Tseng, and DImitar Filev. "Autonomous highway driving using deep reinforcement learning". In: *Conference Proceedings - IEEE International Conference on Systems, Man and Cybernetics* 2019-October (Oct. 2019), pp. 2326–2331. ISSN: 1062922X. DOI: 10.1109/SMC.2019.8914621. URL: https://arxiv.org/abs/1904.00035v1.

[89] Sanmit Narvekar et al. "Curriculum Learning for Reinforcement Learning Domains: A Framework and Survey". In: *Journal of Machine Learning Research* 21 (Mar. 2020), pp. 1–50. ISSN: 15337928. DOI: 10.48550/arxiv.2003.04960. URL: https://arxiv.org/abs/2003.04960v2.

[90] OpenAI. *GPT-4 Technical Report*. Tech. rep. OpenAI, Mar. 2023. URL: https://arxiv.org/abs/2303.08774v3.

[91] OpenAI et al. "Dota 2 with Large Scale Deep Reinforcement Learning". In: (Dec. 2019). DOI: 10.48550/arxiv.1912.06680. URL: https://arxiv.org/abs/1912.06680v1.

[92] Mateusz Orłowski and Paweł Skruch. "A Reinforcement Learning Framework for Motion Planning of Autonomous Vehicles". In: *Progress in Polish Artificial Intelligence Research 4*. Łódź: Lodz University of Technology Press, 2023, pp. 395–400. ISBN: 978-83-66741-92-8. DOI: 10.34658/9788366741928.62.

[93] Mateusz Orłowski and Paweł Skruch. "Multiagent Manuvering with the Use of Reinforcement Learning". In: *Electronics* 12.8 (Apr. 2023), p. 1894. ISSN: 2079-9292. DOI: 10.3390/ELECTRONICS12081894. URL: https://www.mdpi.com/2079-9292/12/8/1894/htm%20https://www.mdpi.com/2079-9292/12/8/1894.

[94] Mateusz Orłowski, Tomasz Wrona, Nikodem Pankiewicz, and Wojciech Turlej. "Safe and Goal-Based Highway Maneuver Planning with Reinforcement Learning". In: *Advances in Intelligent Systems and Computing* 1196 AISC (2020), pp. 1261–1274. ISSN: 21945365. DOI: 10.1007/978-3-030-50936-1{_}105/COVER. URL: https://link.springer.com/chapter/10.1007/978-3-030-50936-1_105.

[95] Mateusz Orłowski, Tomasz Wrona, Wojciech Turlej, and Nikodem Pankiewicz. *Methods and systems for determining a maneuver to be executed by an autonomous vehicle*. 2020. URL: https://patents.google.com/patent/EP4001041A1/de?oq=EP4001041.

[96] Piotr Franciszek Orzechowski, Christoph Burger, and Martin Lauer. "Decision-Making for Automated Vehicles Using a Hierarchical Behavior-Based Arbitration Scheme". In: *IEEE Intelligent Vehicles Symposium, Proceedings* (Mar. 2020), pp. 767–774. DOI: 10.1109/IV47402.2020.9304723. URL: http://arxiv.org/abs/2003.01149%20http://dx.doi.org/10.1109/IV47402.2020.9304723.

[97] Praveen Palanisamy. "Multi-Agent Connected Autonomous Driving using Deep Reinforcement Learning". In: *Proceedings of the International Joint Conference on Neural Networks* (Nov. 2019). DOI: 10.48550/arxiv.1911.04175. URL: https://arxiv.org/abs/1911.04175v1.

[98] Nikodem Pankiewicz, Tomasz Wrona, Wojciech Turlej, and Mateusz Orłowski. "Promises and Challenges of Reinforcement Learning Applications in Motion Planning of Automated Vehicles". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 12855 LNAI (June 2021), pp. 318–329. ISSN: 16113349. DOI: 10.1007/978-3-030-87897-9{_}29. URL: https://link.springer.com/chapter/10.1007/978-3-030-87897-9_29.

[99] Adam Paszke et al. "PyTorch: An Imperative Style, High-Performance Deep Learning Library". In: *Advances in Neural Information Processing Systems 32*. Ed. by H Wallach et al. Curran Associates, Inc., 2019, pp. 8024–8035. URL: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[100] Tung Phan-Minh, Elena Corina Grigore, Freddy A. Boulton, Oscar Beijbom, and Eric M. Wolff. "CoverNet: Multimodal Behavior Prediction Using Trajectory Sets". In: *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (2019), pp. 14062–14071. ISSN: 10636919. DOI: 10.1109/CVPR42600.2020.01408.

[101] Harold Phelippeau, Mohamed Akil, Breno Dias Rodrigues, Hugues Talbot, and S. Bara. "Bayer bilateral denoising on TriMedia3270". In: *Real-Time Image and Video Processing 2009* (Feb. 2009). ISSN: 0277786X. DOI: 10.1117/12.812330. URL: https://www.researchgate.net/publication/237537605_Bayer_Bilateral_denoising_on_TriMedia_3270.

[102] Philip Polack, Florent Altche, Brigitte DAndrea-Novel, and Arnaud De La Fortelle. "The kinematic bicycle model: A consistent model for planning feasible trajectories for autonomous vehicles?" In: *IEEE Intelligent Vehicles Symposium, Proceedings* (July 2017), pp. 812–818. DOI: 10.1109/IVS.2017.7995816.

[103] *Radars | Advanced Safety | Aptiv*. URL: https://www.aptiv.com/en/solutions/advanced-safety/adas/radars.

[104] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. "You Only Look Once: Unified, Real-Time Object Detection". In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition* 2016-December (June 2015), pp. 779–788. ISSN: 10636919. DOI: 10.1109/CVPR.2016.91. URL: https://arxiv.org/abs/1506.02640v5.

[105] Julio K. Rosenblatt. "DAMN: a distributed architecture for mobile navigation". In: *J. Exp. Theor. Artif. Intell.* 9.2-3 (Apr. 1997), pp. 339–360. ISSN: 13623079. DOI: 10.1080/095281397147167.

[106] Reuven Y. Rubinstein and Dirk P. Kroese. *Simulation and the Monte Carlo Method.* 3rd ed. Wiley, 2016, p. 435. ISBN: 978-1-118-63216-1.

[107] Tim Salzmann, Boris Ivanovic, Punarjay Chakravarty, and Marco Pavone. "Trajectron++: Dynamically-Feasible Trajectory Forecasting with Heterogeneous Data". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 12363 LNCS (2020), pp. 683–700. ISSN: 16113349. DOI: 10.1007/978-3-030-58523-5{_}40.

[108] Mikayel Samvelyan et al. "The StarCraft Multi-Agent Challenge". In: *Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems, AAMAS* 4 (Feb. 2019), pp. 2186–2188. ISSN: 15582914. DOI: 10.48550/arxiv.1902.04043. URL: https://arxiv.org/abs/1902.04043v5.

[109] *Saved by the Sensor: Vehicle Awareness in the Self-Driving Age | Machine Design*. URL: https://www.machinedesign.com/mechanical-motion-systems/article/21836344/saved-by-the-sensor-vehicle-awareness-in-the-selfdriving-age.

[110] Maximilian Schafer, Kun Zhao, Markus Buhren, and Anton Kummert. "Context-Aware Scene Prediction Network (CASPNet)". In: *IEEE Conference on Intelligent Transportation Systems, Proceedings, ITSC* 2022-October (Jan. 2022), pp. 3970–3977. DOI: 10.1109/ITSC55140.2022.9921850. URL: https://arxiv.org/abs/2201.06933v1.

[111] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. "Prioritized Experience Replay". In: *4th International Conference on Learning Representations, ICLR 2016 - Conference Track Proceedings* (Nov. 2015). DOI: 10.48550/arxiv.1511.05952. URL: https://arxiv.org/abs/1511.05952v4.

[112] Julian Schrittwieser et al. "Mastering Atari, Go, chess and shogi by planning with a learned model". In: *Nature 2020 588:7839* 588.7839 (Dec. 2020), pp. 604–609. ISSN: 1476-4687. DOI: 10.1038/s41586-020-03051-4. URL: https://www.nature.com/articles/s41586-020-03051-4.

[113] John Schulman. "Optimizing Expectations: From Deep Reinforcement Learning to Stochastic Computation Graphs". PhD thesis. 2016. URL: http://joschu.net/docs/thesis.pdf.

[114] John Schulman, Sergey Levine, Philipp Moritz, Michael Jordan, and Pieter Abbeel. "Trust Region Policy Optimization". In: *32nd International Conference on Machine Learning, ICML 2015* 3 (Feb. 2015), pp. 1889–1897. DOI: 10.48550/arxiv.1502.05477. URL: https://arxiv.org/abs/1502.05477v5.

[115] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov Openai. "Proximal Policy Optimization Algorithms". In: (July 2017). DOI: 10.48550/arxiv.1707.06347. URL: https://arxiv.org/abs/1707.06347v2.

[116] Harm van Seijen and Richard S Sutton. "True Online TD(λ)". In: *Proceedings of the 31st International Conference on Machine Learning* 32.1 (2014), pp. 692–700.

[117] Konstantin M. Seiler, Hanna Kurniawati, and Surya P.N. Singh. "An online and approximate solver for POMDPs with continuous action space". In: *Proceedings - IEEE International Conference on Robotics and Automation* 2015-June.June (June 2015), pp. 2290–2297. ISSN: 10504729. DOI: 10.1109/ICRA.2015.7139503.

[118] Shai Shalev-Shwartz, Shaked Shammah, and Amnon Shashua. "On a Formal Model of Safe and Scalable Self-driving Cars". In: (Aug. 2017). DOI: 10.48550/arxiv.1708.06374. URL: https://arxiv.org/abs/1708.06374v6.

[119] Shai Shalev-Shwartz, Shaked Shammah, and Amnon Shashua. "Safe, Multi-Agent, Reinforcement Learning for Autonomous Driving". In: *arXiv preprint arXiv:1610.03295v1* (Oct. 2016), p. 13. DOI: 10.48550/arxiv.1610.03295. URL: https://arxiv.org/abs/1610.03295v1.

[120] Xu Shen, Xiaojing Zhang, and Francesco Borrelli. "Autonomous Parking of Vehicle Fleet in Tight Environments". In: *Proceedings of the American Control Conference* 2020-July (July 2020), pp. 3035–3040. ISSN: 07431619. DOI: 10.23919/ACC45564.2020.9147671. URL: https://arxiv.org/abs/1910.02349v3.

[121] Xu Shen et al. "ParkPredict: Motion and Intent Prediction of Vehicles in Parking Lots". In: *2020 IEEE Intelligent Vehicles Symposium (IV)* (2020), pp. 1170–1175. DOI: 10.1109/IV47402.2020.9304795.

[122] David Silver et al. "Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm". In: (Dec. 2017). DOI: 10.48550/arxiv.1712.01815. URL: https://arxiv.org/abs/1712.01815v1.

[123] Satinder P. Singh and Richard S. Sutton. "Reinforcement learning with replacing eligibility traces". In: *Machine Learning 1996 22:1* 22.1 (1996), pp. 123–158. ISSN: 1573-0565. DOI: 10.1007/BF00114726. URL: https://link.springer.com/article/10.1007/BF00114726.

[124] *Slurm Workload Manager - Documentation*. URL: https://slurm.schedmd.com/documentation.html.

[125] Bruno Sousa, Tiago Ribeiro, Joana Coelho, Gil Lopes, and A. Fernando Ribeiro. "Parallel, Angular and Perpendicular Parking for Self-Driving Cars using Deep Reinforcement Learning". In: *2022 IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC)* (2022), pp. 40–46. DOI: 10.1109/ICARSC55462.2022.9784800.

[126] Petru Soviany, Radu Tudor Ionescu, Paolo Rota, and Nicu Sebe. "Curriculum Learning: A Survey". In: *International Journal of Computer Vision* 130.6 (June 2022), pp. 1526–1565. ISSN: 15731405. DOI: 10.1007/S11263-022-01611-X. URL: https://arxiv.org/abs/2101.10382v3.

[127] Sainbayar Sukhbaatar, Arthur Szlam, and Rob Fergus. "Learning Multiagent Communication with Backpropagation". In: *Advances in Neural Information Processing Systems* (May 2016), pp. 2252–2260. ISSN: 10495258. DOI: 10.48550/arxiv.1605.07736. URL: https://arxiv.org/abs/1605.07736v2.

[128] Richard S Sutton. "Generalization in Reinforcement Learning: Successful Examples Using Sparse Coarse Coding". In: *Advances in Neural Information Processing Systems*. Ed. by D. Touretzky, M.C. Mozer, and M. Hasselmo. MIT Press, 1995. URL: https://papers.nips.cc/paper/1995/hash/8f1d43620bc6bb580df6e80b0dc05c48-Abstract.html.

[129] Richard S Sutton and Andrew G Barto. *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: A Bradford Book, 2018. ISBN: 0262039249.

[130] Richard S Sutton, David McAllester, Satinder Singh, and Yishay Mansour. "Policy Gradient Methods for Reinforcement Learning with Function Approximation". In: *Advances in Neural Information Processing Systems*. Ed. by S. Solla, T. Leen, and K. Muller. Vol. 12. 1999. URL: https://proceedings.neurips.cc/paper/1999/file/464d828b85b0bed98e80ade0a5c43b0f-Paper.pdf.

[131] Richard S. Sutton. "Learning to predict by the methods of temporal differences". In: *Machine Learning 1988 3:1* 3.1 (Aug. 1988), pp. 9–44. ISSN: 1573-0565. DOI: 10.1007/BF00115009. URL: https://link.springer.com/article/10.1007/BF00115009.

[132] Sebastian Thrun et al. "Stanley: The robot that won the DARPA Grand Challenge". In: *Springer Tracts in Advanced Robotics* 36 (2007), pp. 1–43. ISSN: 1610742X. DOI: 10.1007/978-3-540-73429-1{_}1/COVER. URL: https://link.springer.com/chapter/10.1007/978-3-540-73429-1_1.

[133] Behrad Toghi, Rodolfo Valiente, Dorsa Sadigh, Ramtin Pedarsani, and Yaser P. Fallah. "Cooperative Autonomous Vehicles that Sympathize with Human Drivers". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (2021), pp. 4517–4524. ISSN: 21530866. DOI: 10.1109/IROS51168.2021.9636151.

[134] Behrad Toghi, Rodolfo Valiente, Dorsa Sadigh, Ramtin Pedarsani, and Yaser P. Fallah. "Social Coordination and Altruism in Autonomous Driving". In: *IEEE Transactions on Intelligent Transportation Systems* 23 (2022), pp. 24791–24804. ISSN: 15580016. DOI: 10.1109/TITS.2022.3207872.

[135] *Traffic AI™ – Simteract*. URL: https://simteract.com/projects/traffic-ai/.

[136] Paul G. Trepagnier, Jorge Nagel, Powell M. Kinney, Cris Koutsougeras, and Matthew Dooner. "KAT-5: Robust systems for autonomous vehicle navigation in challenging and unknown terrain". In: *Springer Tracts in Advanced Robotics* 36 (2007), pp. 103–128. ISSN: 1610742X. DOI: 10.1007/978-3-540-73429-1{_}3/COVER. URL: https://link.springer.com/chapter/10.1007/978-3-540-73429-1_3.

[137] Nivedita Tripathi and Senthil Yogamani. "Trained Trajectory based Automated Parking System using Visual SLAM on Surround View Cameras". In: (Jan. 2020). DOI: 10.48550/arxiv.2001.02161. URL: https://arxiv.org/abs/2001.02161v3.

[138] Wojciech Turlej, Mateusz Orłowski, Tomasz Wrona, and Nikodem Pankiewicz. *Method and system for planning the motion of a vehicle*. 2021. URL: https://patents.google.com/patent/US20210300413A1/en.

[139] G. E. Uhlenbeck and L. S. Ornstein. "On the Theory of the Brownian Motion". In: *Physical Review* 36.5 (Sept. 1930), p. 823. ISSN: 0031899X. DOI: 10.1103/PhysRev.36.823. URL: https://journals.aps.org/pr/abstract/10.1103/PhysRev.36.823.

[140] Chris Urmson et al. "A robust approach to high-speed navigation for unrehearsed desert terrain". In: *Springer Tracts in Advanced Robotics* 36 (2007), pp. 45–102. ISSN: 1610742X. DOI: 10.1007/978-3-540-73429-1{_}2/COVER. URL: https://link.springer.com/chapter/10.1007/978-3-540-73429-1_2.

[141] Christopher Urmson et al. *Tartan Racing: A Multi-Modal Approach to the DARPA Urban Challenge*. Tech. rep. Pittsburgh, PA: Carnegie Mellon University, Apr. 2007.

[142] Hado Van Hasselt, Arthur Guez, and David Silver. "Deep Reinforcement Learning with Double Q-learning". In: *30th AAAI Conference on Artificial Intelligence, AAAI 2016* (Sept. 2015), pp. 2094–2100. DOI: 10.48550/arxiv.1509.06461. URL: https://arxiv.org/abs/1509.06461v3.

[143] Harm Van Seijen, A. Rupam Mahmood, Patrick M. Pilarski, Marlos C. Machado, and Richard S. Sutton. "True Online Temporal-Difference Learning". In: *Journal of Machine Learning Research* 17 (Dec. 2015), pp. 1–40. ISSN: 15337928. DOI: 10.48550/arxiv.1512.04087. URL: https://arxiv.org/abs/1512.04087v2.

[144] Akifumi Wachi. "Failure-scenario maker for rule-based agent using multi-agent adversarial reinforcement learning and its application to autonomous driving". In: *IJCAI International Joint Conference on Artificial Intelligence* 2019-August (2019), pp. 6006–6012. ISSN: 10450823. DOI: 10.24963/IJCAI.2019/832.

[145] J. van der. Wal. *Stochastic dynamic programming : successive approximations and nearly optimal strategies for Markov decision processes and Markov games*. 2nd ed. Mathematisch Centrum, 1981. ISBN: 9061962188. URL: https://books.google.com/books/about/Stochastic_Dynamic_Programming.html?id=GMoGPwAACAAJ.

[146] Jiacun Wang and William M. Tepfenhart. *Formal methods in computer science*. Chapman & Hall, July 2019. ISBN: 9781498775328. URL: https://www.routledge.com/Formal-Methods-in-Computer-Science/Wang-Tepfenhart/p/book/9781498775328.

[147] Sen Wang, Daoyuan Jia, and Xinshuo Weng. "Deep Reinforcement Learning for Autonomous Driving". In: *[RecSys2018]Proceedings of the 12th ACM conference on Recommender systems* (Nov. 2018), pp. 95–103. URL: https://arxiv.org/abs/1811.11329v3.

[148] Weixun Wang et al. "From few to more: Large-scale dynamic multiagent curriculum learning". In: *AAAI 2020 - 34th AAAI Conference on Artificial Intelligence* (Sept. 2020), pp. 7293–7300. ISSN: 2159-5399. DOI: 10.1609/AAAI.V34I05.6221. URL: https://arxiv.org/abs/1909.02790v2.

[149] Ziyu Wang et al. "Dueling Network Architectures for Deep Reinforcement Learning". In: *33rd International Conference on Machine Learning, ICML 2016* 4 (Nov. 2015), pp. 2939–2947. DOI: 10.48550/arxiv.1511.06581. URL: https://arxiv.org/abs/1511.06581v3.

[150] Ziyu Wang et al. "Sample Efficient Actor-Critic with Experience Replay". In: *5th International Conference on Learning Representations, ICLR 2017 - Conference Track Proceedings* (Nov. 2016). DOI: 10.48550/arxiv.1611.01224. URL: https://arxiv.org/abs/1611.01224v2.

[151] Christopher J. C. H. Watkins and Peter Dayan. "Q-learning". In: *Machine Learning* 8.3-4 (May 1992), pp. 279–292. ISSN: 0885-6125. DOI: 10.1007/bf00992698. URL: https://link.springer.com/article/10.1007/BF00992698.

[152] Lianzhen Wei, Zirui Li, Jianwei Gong, Cheng Gong, and Jiachen Li. "Autonomous Driving Strategies at Intersections: Scenarios, State-of-the-Art, and Future Outlooks". In: *IEEE Conference on Intelligent Transportation Systems, Proceedings, ITSC* 2021-September (Sept. 2021), pp. 44–51. DOI: 10.1109/ITSC48978.2021.9564518. URL: https://arxiv.org/abs/2106.13052v2.

[153] *Welcome to the Ray documentation — Ray 1.12.0*. URL: https://docs.ray.io/en/latest/index.html.

[154] Moritz Werling, Julius Ziegler, Sören Kammel, and Sebastian Thrun. "Optimal trajectory generation for dynamic street scenarios in a Frenét Frame". In: *2010 IEEE International Conference on Robotics and Automation*. 2010, pp. 987–993. DOI: 10.1109/ROBOT.2010.5509799.

[155] Ronald J. Williams. "Simple statistical gradient-following algorithms for connectionist reinforcement learning". In: *Machine Learning 1992 8:3* 8.3 (May 1992), pp. 229–256. ISSN: 1573-0565. DOI: 10.1007/BF00992696. URL: https://link.springer.com/article/10.1007/BF00992696.

[156] Nicolai Wojke, Alex Bewley, and Dietrich Paulus. "Simple online and realtime tracking with a deep association metric". In: *Proceedings - International Conference on Image Processing, ICIP* 2017-September (Feb. 2018), pp. 3645–3649. ISSN: 15224880. DOI: 10.1109/ICIP.2017.8296962. URL: https://arxiv.org/abs/1703.07402v1.

[157] Jia Wu, Florent Perronnet, and Abdeljalil Abbas-Turki. "Cooperative vehicle-actuator system: A sequencebased framework of cooperative intersections management". In: *IET Intelligent Transport Systems* 8.4 (2014), pp. 352–360. ISSN: 1751956X. DOI: 10.1049/IET-ITS.2013.0093. URL: https://www.researchgate.net/publication/256473294_Cooperative_vehicle-actuator_system_A_sequencebased_framework_of_cooperative_intersections_management.

[158] Yi Xiao, Felipe Codevilla, Akhil Gurram, Onay Urfalioglu, and Antonio M. López. "Multimodal End-to-End Autonomous Driving". In: *IEEE Transactions on Intelligent Transportation Systems* 23.1 (June 2019), pp. 537–547. DOI: 10.1109/TITS.2020.3013234. URL: http://arxiv.org/abs/1906.03199%20http://dx.doi.org/10.1109/TITS.2020.3013234.

[159] Huile Xu, Yi Zhang, Li Li, and Weixia Li. "Cooperative Driving at Unsignalized Intersections Using Tree Search". In: *IEEE Transactions on Intelligent Transportation Systems* 21.11 (Nov. 2020), pp. 4563–4571. ISSN: 15580016. DOI: 10.1109/TITS.2019.2940641. URL: https://arxiv.org/abs/1902.01024v1.

[160] Weirui Ye et al. "Mastering Atari Games with Limited Data". In: (Oct. 2021). DOI: 10.48550/arxiv.2111.00210. URL: https://arxiv.org/abs/2111.00210v2.

[161] Chao Yu et al. "The Surprising Effectiveness of PPO in Cooperative, Multi-Agent Games". In: (Mar. 2021). DOI: 10.48550/arxiv.2103.01955. URL: https://arxiv.org/abs/2103.01955v4.

[162] Shuyou Yu, Matthias Hirche, Yanjun Huang, Hong Chen, and Frank Allgöwer. "Model predictive control for autonomous ground vehicles: a review". In: *Autonomous Intelligent Systems 2021* 1.1 (Aug. 2021), pp. 1–17. ISSN: 2730-616X. DOI: 10.1007/S43684-021-00005-Z. URL: https://link.springer.com/article/10.1007/s43684-021-00005-z.

[163] Ekim Yurtsever, Jacob Lambert, Alexander Carballo, and Kazuya Takeda. "A Survey of Autonomous Driving: Common Practices and Emerging Technologies". In: *IEEE Access* 8 (2019), pp. 58443–58469. ISSN: 21693536. DOI: 10.1109/ACCESS.2020.2983149.

[164] Wenyuan Zeng et al. "End-To-End Interpretable Neural Motion Planner". In: *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* 2019-June (June 2019), pp. 8652–8661. ISSN: 10636919. DOI: 10.1109/CVPR.2019.00886.

[165] Ji Zhang and Sanjiv Singh. "LOAM : Lidar Odometry and Mapping in real-time". In: *Robotics: Science and Systems Conference (RSS)* (June 2014), pp. 109–111.

[166] Peizhi Zhang et al. "Reinforcement Learning-Based End-to-End Parking for Automatic Parking System". In: *Sensors (Basel, Switzerland)* 19.18 (Sept. 2019). ISSN: 14248220. DOI: 10.3390/S19183996.

[167] Xiaojing Zhang, Alexander Liniger, Atsushi Sakai, and Francesco Borrelli. "Autonomous Parking Using Optimization-Based Collision Avoidance". In: *2018 IEEE Conference on Decision and Control (CDC)* 2018-December (July 2018), pp. 4327–4332. ISSN: 25762370. DOI: 10.1109/CDC.2018.8619433.

[168] Xinyuan Zhang, Cong Zhao, Feixiong Liao, Xinghua Li, and Yuchuan Du. "Online parking assignment in an environment of partially connected vehicles: A multi-agent deep reinforcement learning approach". In: *Transportation Research Part C: Emerging Technologies* 138 (May 2022). ISSN: 0968090X. DOI: 10.1016/J.TRC.2022.103624.

[169] Yue J. Zhang, Andreas A. Malikopoulos, and Christos G. Cassandras. "Optimal Control and Coordination of Connected and Automated Vehicles at Urban Traffic Intersections". In: *Proceedings of the American Control Conference* 2016-July (Sept. 2015), pp. 6227–6232. DOI: 10.1109/ACC.2016.7526648. URL: http://arxiv.org/abs/1509.08689%20http://dx.doi.org/10.1109/ACC.2016.7526648.

[170] Julius Ziegler, Philipp Bender, Thao Dang, and Christoph Stiller. "Trajectory planning for Bertha — A local, continuous method". In: *2014 IEEE Intelligent Vehicles Symposium Proceedings* (2014), pp. 450–457. DOI: 10.1109/IVS.2014.6856581.

[171] Julius Ziegler et al. "Making bertha drive-an autonomous journey on a historic route". In: *IEEE Intelligent Transportation Systems Magazine* 6.2 (2014), pp. 8–20. ISSN: 19391390. DOI: 10.1109/MITS.2014.2306552.