UNIVERSITÄT
LEIPZIG

Leipzig University
Institute of Computer Science
Faculty of Mathematics and Computer Science
Database Department

Train an Agent to Drive a Vehicle in a Simulated Environment Using Reinforcement Learning

Master Thesis

Submitted by:
Maximilian Schaller

Matriculation number:
3750663

Supervisor:
Prof. Dr. Erhard Rahm
Dr. Thomas Burghardt

Abstract

This master thesis investigates the capability of reinforcement learning techniques for training an agent to drive a vehicle in a simulated environment using visual camera input. The goal is to contribute to the understanding and developing of autonomous driving systems in realistic simulated environments using machine learning.

Therefore, this thesis addresses the following research questions: (1) Can the problem of training an autonomous driving agent be modeled in the context of RL? (2) How can visual camera input be processed and utilized to achieve learning success? (3) How good is RL at solving the problem of training an agent to drive a vehicle and pass all goals? (4) How robust are the developed RL algorithms against varying external influences?

To answer these questions, a real existing robot and arena have been modeled in a simulation by using Unity. The proposed objective has been fit into the context of Reinforcement Learning by modeling the state, action, and reward. Therefore, four distinct Proximal Policy Optimization variants have been designed to challenge this task. Two of the approaches used simplified training, in which the objective during the training was minimized to always reach the next goal, whereas the other two approaches trained on complete parkours. In addition, two algorithms were distinguished by the inclusion of memory in one approach while excluding it in the other. All variations have been limited to the usage of the pre-processed camera picture attached to the robot as input.

In conclusion, this thesis successfully modeled the problem of training an autonomous driving agent using RL techniques that consume the pre-processed vehicle's camera picture and achieved satisfactory performance in various experiments throughout all designed algorithms. The research findings contribute to the advancement of training machine learning algorithms in simulated environments and AI-optimized driving systems in general. Moreover, it provides valuable insights for further research, including training the algorithms for a longer time, incorporating additional sensors or multiple cameras, enhancing image pre-processing techniques, improving memory utilization, and training the agent with varying external influences from the beginning. Additionally, this thesis provides a modular framework that can be used to adjust the simulation and examine alternative approaches related to the environment of this thesis.

Overall, this work sets the foundation for the development of more efficient, capable, and robust autonomous driving agents in the future.

Maximilian Schaller
3750663

Contents

Nomenclature

β	Entropy coefficient
δ	State transition function
Γ	Markov Decision Process
γ	Discount factor
λ	GAE parameter
$\mathcal{A}$	Set of Actions in Markov Decision Process
$\mathcal{R}$	Set of Reward functions in Markov Decision Process
$\mathcal{S}$	Set of States in Markov Decision Process
π	Policy function of RL agent
ρ	Ratio of the policy change after the network update
τ	Trajectory of state-action-reward triples
θ	Parameter/ Weights of a the deep neural network
A	Advantage function
a	Action in Markov Decision Process
E	Episode
K	Maximum number of steps in one training
R	Reward function
r	Reward in Markov Decision Process
s	State in Markov Decision Process
T	Final time step of an episode in the Markov Decision Process
T	Global Environment time in the training
t	Discrete time step in Markov Decision Process
V_ψ	Value network defined by parameter ψ

Acronyms

A2C Advantage Actor Critic 5, 13

A3C Asynchronous Advantage Actor Critic 13

ACKTR Actor Critic using Kronecker-Factored Trust Region 5

AI Artificial Intelligence 1

CNN Convolutional Neural Network 5, 6, 66

DDPG Deep Deterministic Policy Gradient 5

DDQN Double Deep Q-Network 6

DNN Deep Neural Network 4, 9, 19, 32

DQN Deep Q-Network 4, 5

DRL Deep Reinforcement Learning 7

FMT Full-Map-Training 35, 36, 40, 41

GAE Generalized Advantage Estimation 14, 32

GO Game Object 17, 18

LSTM Long Short-Term Memory 5–7, 66

MDP Markov Decision Process 4, 8

ML Machine Learning 1, 14, 19, 36

MSE Mean Squared Error 35

NCR Normalized Cumulative Reward 49

PPO Proximal Policy Optimization 2, 5–7, 13–16, 19, 31, 33, 38, 39

PPO-BASIC Proximal Policy Optimization Without Memory 39

PPO-BASIC-FMT Proximal Policy Optimization Without Memory Using Full-Map-Training 42, 46, 52, 53, 57, 58, 60, VII

PPO-BASIC-SGT Proximal Policy Optimization Without Memory Using Single-Goal-Training 42, 46, 52–54, 58, 60, VII

PPO-CLIP PPO — Variant with clipped objective 15, 16, 35, 38

PPO-KL PPO with Adaptive KL Penalty 15

PPO-MEM Proximal Policy Optimization Using Memory 39

PPO-MEM-FMT Proximal Policy Optimization Using Memory and Full-Map-Training 42, 46, 52, 53, 58–60, VII

PPO-MEM-SGT Proximal Policy Optimization Using Memory and Single-Goal-Training 42, 46, 52–54, 57, 58, 60, 65, VII

RL Reinforcement Learning 1–16, 19, 27–29, 31, 33, 38, 43, 47, 52, 64, III, VI

RNN Recurrent Neural Network 66

SAC Soft Actor-Critic 7

ScaDS.AI Center for Scalable Data Analytics and Artificial Intelligence 1

SGD Stochastic Gradient Descent 5

SGT Single-Goal-Training 35–37, 40, 41

TD3 Twin Delayed Deep Deterministic Policy Gradient 7

TRPO Trust Region Policy Optimization 13, 15

VPG Vanilla Policy Gradient 13, 14, 16

List of Figures

List of Tables

1. Introduction

The rapid development of Artificial Intelligence (AI) and Machine Learning (ML) has led to significant achievements in various domains such as autonomous driving or, most currently, AI-based language models such as ChatGPT[1]. Furthermore, the most popular breakthroughs succeeded by using Reinforcement Learning (RL) as the fundamental learning algorithm. RL is a subfield of machine learning and has gained considerable attention in recent years due to its ability to train agents to solve complex tasks through trial and error. RL has shown significant success, for instance, in game playing, robotic control, and training autonomous vehicles to navigate real-world environments.

The development of AI-optimized autonomous driving yields a tremendous opportunity to revolutionize transportation by improving road safety, reducing traffic, and enhancing the overall efficiency of transportation systems, thus minimizing environmental pollution. However, training autonomous vehicles to operate in real-world environments presents numerous challenges, including the high cost and risk associated with testing on public roads. Therefore, developing realistic simulated environments offers a safe, cost-effective alternative for training and evaluating algorithms. Furthermore, it is central to divide the problem into small sub-tasks to achieve progress in such a complex challenge.

Prior to this research, two preceding bachelor theses contributed groundbreaking work to the current investigation. The first thesis [1] explored the possibility of navigating a simulated vehicle in an arena using evolution-based neural networks. Subsequently, a second thesis [2] attempted to bridge the gap between simulation and reality by implementing the findings of the first thesis in a real-world context. However, several challenges arose during this process. At this point, this master thesis originates. On the one hand, it tries to improve the work of [1] by utilizing Reinforcement Learning as a promising approach in solving complex control tasks, and, on the other hand, it intends to avoid the problems that appeared in the work of [2], by considering the findings and modeling the real-world context as accurately as possible. As real-world context serves a constructed arena and a robot, both located at the research facility Center for Scalable Data Analytics and Artificial Intelligence (ScaDS.AI) [2].

To sum up, this master thesis aims to investigate the feasibility of using reinforcement learning techniques for training an agent to drive a vehicle in a simulated environment by consuming visual camera input. Specifically, the focus will be on developing an RL-based approach that is able to control a virtual vehicle to navigate through a simulated environment in which red and blue goals are placed. The vehicle should be able to correctly interpret the camera input and guide the vehicle through the map by driving through all spawned goals. During the simulation development, strict attention is paid to modeling all aspects as realistically as possible.

[1] OpenAI. (2023). ChatGPT (May 24 Version). `https://chat.openai.com/chat`

[2] ScaDS.AI (Center for Scalable Data Analytics and Artificial Intelligence) Dresden/Leipzig. `https://scads.ai`

Throughout the research, this thesis seeks to answer the following questions:

1. Is it possible to model the problem of training an autonomous driving agent in the context of reinforcement learning? This thesis will explore formulating the problem as a Reinforcement Learning problem and design suitable reward functions, state, and action spaces.

2. How can the camera input, which provides visual information about the environment, be processed and utilized to feed into the RL algorithm and achieve learning success?

3. What is the performance of the developed RL algorithms in solving the problem of training an agent to drive a vehicle in the simulated environment and pass all goals? The thesis will evaluate the performance of the RL-based approaches to assess the suitability and effectiveness of this specific task.

4. How robust are the developed algorithms against varying external influences, including light settings and motor engine power?

By addressing these research questions, this thesis aims to contribute to the understanding of how Reinforcement Learning can be used to train autonomous driving agents navigating vehicles through goals by only using the camera as the input source. Furthermore, the vehicle used in this thesis simulates a real existing robot. Therefore, the insights gained from this study help to refine existing techniques and provide a new approach that can be transferred and used on this robot in reality. Additionally, this thesis delivers a framework that enables further research with different machine-learning attempts in the constructed simulation.

This thesis is structured as follows: Chapter 2 provides an overview of preceding work, recent studies, and related work. Chapter 3 provides the background information and theoretical foundations on reinforcement learning and the simulation required to understand this work. Chapter 4 describes the implementation aspects, including the simulated environment, image pre-processing techniques, and the training setup. Chapter 5 provides insights into the experimental setup and the design of the experiments to evaluate the performance of the trained agents and to answer the stated research questions. The results and evaluation of the experiments are presented in Chapter 6. Finally, Chapter 7 concludes the thesis, summarizing the results, outlining future work, and highlighting the enhancements of the preceding works and the contributions to autonomous driving, robotics, and artificial intelligence.

2. Related Work

The underlying idea of the research of this work derives from two preceding theses. Thus, Section 2.1 briefly introduces these works and gives detailed insights that are important to understanding the critical components of this thesis. Following, the focus is shifted towards related studies considering the theory behind this research. Firstly, as Reinforcement Learning builds the fundamental algorithm to solve the navigating problem in this work, Section 2.2 describes the literature that explains the underlying concepts. Finally, Section 2.3 provides a collection of studies that research on comparable experiments and, thus, provides valuable insights into results that can be reused in the experiments of this work.

2.1. Build Upon Previous Work

This thesis continues the work of two previous bachelor theses, which provide foundational research. This Section supplies detailed information on these works.

In the work of [1], a comprehensive simulation, including the arena, obstacles, and vehicle, has been developed using Unity and serves as the basis for the current simulation. This work already intended to model the arena and the obstacles existing in reality at ScaDS.AI but produced a few inaccuracies in the modeling. The vehicle has yet to be constructed at the time of the work of [1]. Thus, the modeled vehicle in [1] entirely differs from the vehicle in this thesis and the reality. The differences include the sizes of the vehicle and the driving physics. Thus, the simulation in this research is an improved version with several significant changes to more accurately correspond to the reality at ScaDS.AI. However, the author explored the feasibility of evolutionary algorithms to maneuver the vehicle within the map. To solve the task, it is necessary to process the camera input. Note that the vehicle has a front camera that serves the camera input. Hence, the author introduced the idea of using OpenCV to detect the boundaries of the different colored obstacles. Despite this working very well and the author's ability to accurately detect substantial obstacles efficiently, the author encountered a problem. In different views of the vehicle, there are a different number of obstacles that are recognized and processed by OpenCV. As the algorithms in the research require a constant input, the author had to map the changing output of OpenCV to a list with a constant shape. The author achieved this by encoding the output using a neural network. This thesis has precisely the same input as in [1] and resues the achieved knowledge. Thus, this thesis uses the OpenCV pipeline but renounces using the neural network and manually maps the output. The exact concept will be explained in more detail in Section 4.2. Moreover, in the research of [1], the author implemented the evolutionary algorithms by hand without using high-level APIs. Thus, it was required to implement a communication interface between the simulation and the custom algorithms. This appeared to be a very complex task that caused plenty of

problems. Therefore, this work avoided implementing this manually and investigated for appropriate APIs. Subsequently, the ML-Agents Framework will be used, as discussed in Section 3.2.2. Unfortunately, the task requirements have changed in this thesis, and the author in [1] has not evaluated the experiments to an extent that allows a comparison.

In the other work [2], the author tried to transfer the simulation of [1] to reality and investigates the simulation-to-reality gap. Therefore, the author intended to bring the research of [1] into a real-world setting. Challenging the upcoming problems provided fundamental ideas to construct the simulation in this thesis to avoid these problems from the beginning. This includes modeling all objects, such as the arena with the obstacles and the vehicle with the correct sizes as in reality, placing the vehicles camera at the same position, at the same angle and using the exact resolution, and modeling the physics as accurately as possible. Furthermore, the author constructed the physical robot as this thesis's final real-world model. Another crucial part of the author's work was examining the significant issues of the computer vision approach. The results of this research led to the final technique that is used in this work.

Given that a long-term goal of this thesis is to transfer the simulation to reality in the future, the work of [2] helped to target likely upcoming issues concerning this objective. This led to crucial design decisions in the simulation and algorithms. In addition, besides evaluating only the algorithm's performance, it inspired the idea to investigate its stability considering changes in external influences such as varying motor engines.

2.2. Reinforcement Learning

Reinforcement Learning serves as the key component during the experimentation. Therefore, this Section introduces the most relevant literature that inspired the main ideas in this thesis.

Sutton and Bartos's book [3] serves as a comprehensive introduction to RL, providing concepts and algorithms. It serves as a valuable resource for understanding the fundamentals of RL and gives an idea of how to set up RL in different contexts to solve complex problems. It introduces the underlying mathematical framework Markov Decision Process (MDP), including all core components of RL problems, such as *State*, *Action*, *Reward*, *Policy*, that are required to model problems in the context of RL .

One influential work in RL is the Atari DQN [4], which combined RL with Deep Q-Network (DQN) for playing Atari 2600 games. This work provided initial insights into the potential of RL combined with Deep Neural Networks. However, it is considered as one of the fundamental papers in the domain of RL and, hence, it highlights some of the core challenges and gives a good intuition about using RL together with complex inputs such as raw image pixels. With their approach, they achieved tremendous results in six out of seven games and demonstrated the potential of RL in complex controlling tasks.

Nevertheless, they also showed that the DQN algorithm still lacks stability, which leads to the decision to use PPO instead of DQN in this research.

In the work of [5], they examine the performance of different RL algorithms. This study undermines the effectiveness of Proximal Policy Optimization, as it showed a stable and fast learning process compared to other approaches, such as Deep Deterministic Policy Gradient (DDPG) [6] or Actor Critic using Kronecker-Factored Trust Region (ACKTR) [7]. Furthermore, they discovered the primary influence of the hyperparameter on the performance of the different algorithms.

Another work by [8] compares gradient-based RL methods, such as Advantage Actor Critic and PPO. They mainly focus on the influence of the learning rate and different gradient descent-based optimization methods, including Stochastic Gradient Descent (SGD) [9] and specific variants like SGD-Adam [10], which is used in the experimentation in the PPO algorithm. They determined that PPO shows promising performances concerning a brighter variety of learning rates. Additionally, they mentioned that it is not sufficient to use the default values of the optimizer and that they should also be fine-tuned, besides other hyperparameters, to achieve satisfying results.

[11] presents a benchmarking study of four different RL algorithms, including Proximal Policy Optimization , on commercially available real physical robots. The results underline the excellent performances of PPO and indicate that Proximal Policy Optimization, generally, is a good choice when it comes to solving complex tasks in the real world. Despite this, the study highlights the algorithm's sensitivity to the hyperparameter setting. It draws attention to the fact that it is doubtful that the hyperparameter setting of one experiment can be transferred without any deviation from another experiment.

In [12], the authors used Proximal Policy Optimization to train a robot in order to solve precise control tasks, maneuvering a cube with a robotic hand. The study successfully employs Proximal Policy Optimization to solve the task using RGB images as input. They mentioned that it is necessary to preprocess the camera input. Otherwise, the state space for the Reinforcement Learning would be too huge to solve the problem in a reasonable time. In their approach, they used Convolutional Neural Network (CNN) to pre-process the image before feeding it to the algorithm and trained this network besides the RL algorithm. This proposes another exciting and promising approach, besides the method in this thesis, on how to deal with the camera input, as they also achieved great success. Nevertheless, as they demonstrated the effort that they required to design the architecture for the CNN, including a hyperparameter tuning and training, completely separated from the RL algorithm, it is apparent that this attempt exceeds the scope of this thesis. Particularly considering that the work in [1] provided a significantly more straightforward and promising approach. Additionally, they examined the effectiveness of using memory in RL tasks. Therefore, they added memory by applying different Long Short-Term Memory (LSTM) to the algorithms and showed that memory has a positive effect on the performance of the algorithm. Despite this, it is evident from the methods used that these are

beyond the scope of this work since these techniques are very complex and require particular research. Furthermore, this work particularly examines how to fit RL to this problem, modeling the state, action, and reward function and providing a brief explanation of the parameter setting, giving valuable insights that could be transferred to the experiments in this thesis. Moreover, similar to this thesis, they trained the algorithms in a simulation and successfully transferred them to reality. For their experiments, they had access to computers using up to 16 GPUs and 12228 CPU cores and spent up to 50 hours training one algorithm. This indicates how complex and time-consuming the training of RL is and helps to estimate the necessary time to train the algorithms in this thesis. Thus, the complexity has to be reduced as much as possible to be able to achieve success in a reasonable amount of time on the hardware available in this project.

2.3. Self-Driving Vehicles in Simulations based on Reinforcement Learning

This section explores recent research studies that share a similar basis of training self-driving vehicles in simulated virtual environments using Reinforcement Learning .

[13] conducts the first study. They constructed a simulated environment in Unity, where they used a real existing robot called Donkey Car as the model for the vehicle. They seek to train an RL algorithm to control the donkey car and navigate it along a street by consuming the visual input of a camera. For preprocessing the camera picture, they used a CNN, similar to the work in [12] and implemented the DDQN [4] as the learning algorithm. They combined the last three recorded camera frames with the actual frame as a simple memory. This provides a simple and effective alternative to the LSTM in the research of [12]. The objective of the simulated vehicle was to navigate along a curvy street. In the simulation, the provided approach showed a strong performance. Subsequently, they transferred the model trained in unity to the real-world car and observed a satisfying result even in the real-world scenario. The results show that the trained car demonstrates effective self-driving capabilities and that the learned policy is successfully transferred to reality.

In another work by [14], a simulation environment is created using Unity, and the MLA-gents library is employed to set up a training environment for RL agents. These agents are trained to navigate carts through a track, especially using PPO as the training algorithm. They achieved great success in navigating cars through complex environments and overcoming challenging obstacles.

Furthermore, in the recent work of [15], they trained an agent to control a vehicle in a Unity 3D simulated environment, where the agent had to successfully switch lanes and navigate through traffic by steering and accelerating while avoiding collisions. This is another excellent example that shows the significant success of RL in such complex controlling tasks.

Additionally, in [16], they trained an RL agent to control a vehicle in a simulated environment, specifically for parking the car. The implementation of Proximal Policy Optimization proved successful in training the agent, and the learned policy was effectively transferred from the simulation to the real world.

Lastly, the authors in [17] developed a Deep Reinforcement Learning (DRL) framework for training an agent in a simulated environment to navigate a vehicle to follow a lane and safely overtake slower vehicles and, subsequently, transfer this model to reality. Therefore, they implemented a perception module that extracts relevant information from the camera input by transferring the picture into HSV color space that uses color thresholds and transformation tasks and feeds the information as input state to the RL algorithm, similar to the pre-processing in this thesis. Following, they implemented a control module that utilized the input and applied different RL methods, such as Soft Actor-Critic (SAC) [18] Twin Delayed Deep Deterministic Policy Gradient (TD3) [19] with and without memory in the form of a LSTM [20]. In their experiments, they achieved promising results and provided valuable insights in an environment that is very similar to the one in this thesis, especially concerning the vehicle, camera input, and controlling module.

Although these research studies may differ in certain aspects, they all align closely with the experiments in the thesis. They helped to develop the ideas on how to apply RL to the context of this research and showed valuable examples of modeling the state, action, and reward function. Furthermore, these studies emphasize the importance of image pre-processing techniques and parameter fine-tuning. They further provide valuable insights for the parameter configuration chosen in this thesis. Additionally, they show that the utilization of Unity as an engine and ML-Agents as a training framework is very beneficial.

3. Background

This Chapter introduces the required theoretical background that explains the core concepts utilized in this thesis. Therefore, it is divided into two Sections: Section 3.1 discusses the principles of Reinforcement Learning, and Section 3.2 provides a brief overview of the functionality of the simulation.

3.1. Reinforcement Learning

Reinforcement Learning [3] is one of the three main domains in machine learning, alongside supervised learning and unsupervised learning. Unlike supervised learning, where a supervisor provides labeled data for training, or unsupervised learning, where the algorithm discovers patterns and structures in unlabeled data, RL trains the algorithm by interacting with the environment and, thus, generating its proper data set through observing the surroundings. The agent learns from the environment's responses in a process of trial-and-error. This makes RL an approach that comes close to human learning and is considered a promising approach to achieving natural intelligence. To challenge the objective of this thesis, all examined algorithms derive from Reinforcement Learning as a fundamental principle. Thus, this Section proposes detailed insights into the core concepts. Section 3.1.1 introduces the Markov Decision Process (MDP) [3, 21, 22, 23, 24, 25, 26] as the underlying mathematical framework that constitutes the foundational theory behind RL. Following, Section 3.1.2 offers an overview of the taxonomy of the different variations of RL, highlighting the decision path to the utilized variation in this work. Finally, Section 3.1.3 gives valuable information about the theoretical background of the selected algorithm, Proximal Policy Optimization.

3.1.1. Theoretical foundation

This Section provides an introduction to the mathematical background of Reinforcement Learning. Subsequently, it supplies the most crucial definitions required to model a problem in the context of RL.

Definition 1. A *Markov Decision Process* [3, 21, 26] is a stochastic decision-making process that can be used as a mathematical framework to describe the context of Reinforcement Learning (RL). It consists of a Tuple $\Gamma = (\mathcal{S}, \mathcal{A}, \pi, \delta, R)$, where $\mathcal{S}$ is a set of *Markov States*, $\mathcal{A}$ a set of actions, π the policy function that proposes a probability function that determines which actions to take in certain situations. γ represents the state-transition function that defines the transition from one state to another when an action is performed. Finally, R is the Reward function.

Definition 2. A *markov state* $s \in \mathcal{S}$ [3, 21] describes the current surroundings of the agent's environment. Hence, the set of states $\mathcal{S} = \{s_0, s_1, s_2, ...\}$ comprises all possible states that the environment can enter. In a time-dependent system, a state $s_t \in \mathcal{S}$ represents the environment at timestamp t.

Definition 3. The *environment* [3, 21] is defined by the set of states $\mathcal{S}$. In the context of Reinforcement Learning, it describes the current surroundings s_t of the agent at a given, discrete point of time t.

Definition 4. The *agent* [3, 21] is the acting unit that can execute specific actions and, thereby, interacts with the surroundings. In every timestamp t, the agent decides to take an action a_t in the state s_t following the trained policy π.

Definition 5. An *action* a_t [3, 21] is an interaction with the environment at a certain point in time t that results in a transition from the current state s_t to a new state s_{t+1} according to the state-transition function δ [3] from Equation 3.1. Therefore, in every time step t, the agent has to choose an action a_t from the set $\mathcal{A} = \{a_0.a_1, a_2, ...\}$. The set of actions $\mathcal{A}$ comprises all actions the agent can take.

$$\delta : \mathcal{S} \times \mathcal{A} \to \mathcal{S} \tag{3.1}$$

Definition 6. The *reward* r_t [3, 21] is the environment's numerical response to the agent's performed action. It indicates how beneficial the action that the agent has accomplished. The set of rewards $\mathcal{R} \subseteq \mathbb{R}$ constitutes all possible rewards that could either be positive and, thus, encourage a favorable action or negative, which penalizes unfavorable actions. The reward is calculated in the manner of the reward function [21]:

$$R : \mathcal{S} \times \mathcal{A} \to \mathcal{R} \tag{3.2}$$

Definition 7. The *policy* π [3, 21, 26] is a deterministic or stochastic function that should decide which action to take in which state. In the scope of this thesis, policy π will only be considered stochastic. This means that for a given state s_t, the policy assigns a probability distribution over the possible actions a_t. Thus, the agent selects actions based on these probabilities, mainly choosing the most promising action. Hence, the policy serves as a strategy that guides agents to maneuver through an environment. It is important to note that in the context of the implemented algorithms in this thesis, the policy function is represented by a Deep Neural Network (DNN). The parameters of the DNN, denoted as θ, determine the behavior of the policy. These parameters can be adjusted to increase the agent's performance.

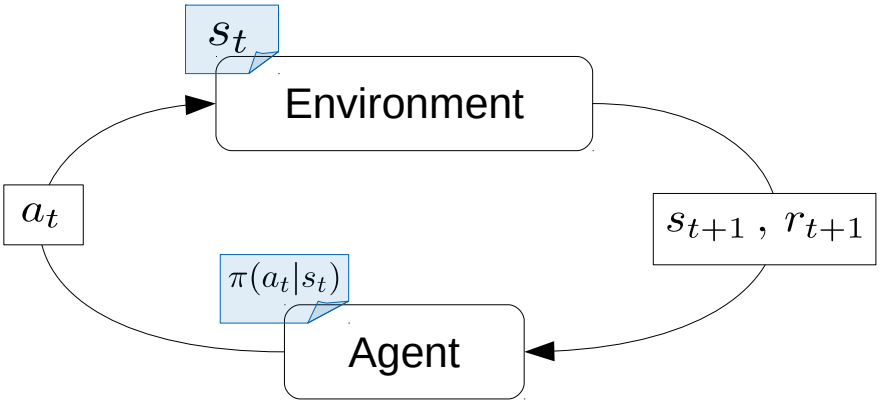


Figure 3.1.: Reinforcement Learning Cycle

Definition 8. A finite sequence $\tau = s_0, a_0, r_0, s_1, a_1, r_1, ...s_T, a_T, r_T$ [3, 26] of state-action-reward triples (s_t, a_t, r_t) that result from the interaction of the agent with the environment, is called *trajectory*. Each trajectory has a probability of $P(\tau)$ and yields a cumulative reward of $R(\tau)$. A trajectory is an arbitrary sequence of iterations in the RL cycle as shown in Figure 3.1.

Figure 3.1 shows the RL process, which is represented by a circuit consisting of two main parts: the *Agent* and the *Environment*. At each time step t, the agent executes an action a_t from the set of action $\mathcal{A}$ based on the policy $\pi(a_t|s_t)$. Subsequently, the environment responds to the action by transitioning from state $s_t \rightarrow s_{t+1}$ according to the transition function γ and provides a reward r_{t+1} to the agent. Thus, in every iteration t, a State-Action-Reward Triple (s_t, a_t, r_{t+1}) is collected, and the agent is situated in the new state s_{t+1}. In RL, this process is repeated over a bunch of steps t until the final step T and can be summarized into one episode. Note that in this representation, r_0 represents the immediate reward received by the agent in the initial state and does not correspond to a state transition.

Definition 9. An *episode* E [3, 21, 26] refers to a complete interaction or run of an agent within an environment that is terminated after T time steps. An episode typically consists of multiple trajectories comprising the history of state-action-reward triples, e.g., $E = [(s_0, a_0, r_1), (s_1, s_2, r_2), ...]$. An episode starts with placing the agent in the initial state s_0. The agent then acts in the environment, by following the policy according to the RL cycle. This process is done until a terminal state at time step T is reached. The terminal state can be defined by a time limit or an aborting signal and indicates the end of an episode. In order to tune the policy and train the agent, multiple episodes are completed within one RL problem. Episodes inside an RL problem are independent of each other. In the context of computer games, this could be seen as attempting the same level multiple times. One trial of the level would refer to one episode.

The purpose of Reinforcement Learning is to solve a complex problem. To achieve this, the problem has to be modeled as a RL problem containing all the crucial parts such as

Environment and Agent. Solving the problem in the context of RL would mean maximizing the cumulative reward r_{max} of one episode [3]:

$$r_{max} = r_0 + r_1 + r_2 + ... + r_T \tag{3.3}$$

Equation 3.3 indicates that the reward sums up over every time step. From Equation 3.2, it is apparent that the size of each part of the reward is directly influenced by the action the agent takes in each state. Consequently, since the policy determines which action to take, the overall reward that the agent acquires is determined by the quality of the policy. Therefore, the ultimate objective in Reinforcement Learning is to find the optimal policy that maximizes the cumulative reward over time.

Definition 10. The *optimal policy* [3, 21] is the best strategy that the agent can follow to achieve the highest possible reward during a complex task. It represents the ideal set of actions for the agent to take in each state to maximize the cumulative reward.

3.1.2. Taxonomy of Reinforcement Learning

The research on Reinforcement Learning has significantly increased over recent years, and thus, a very diverse set of different approaches to solving RL problems appeared. Understanding the taxonomy and classification of RL algorithms enables a better understanding of the algorithm selection and experimentation conducted in this thesis. Therefore, this Chapter briefly overviews and explains the most significant attempts.

To illustrate the landscape of RL approaches, Figure 3.2 shows a hierarchical breakdown of the different domains in RL, following the fundamental work of [27, 28]. It is apparent, that the breakdown could possibly be more fine-granular, but in order to stick to the scope of this thesis, it is limited to the most important components in order to be able to classify the algorithms that are used during the experiments.

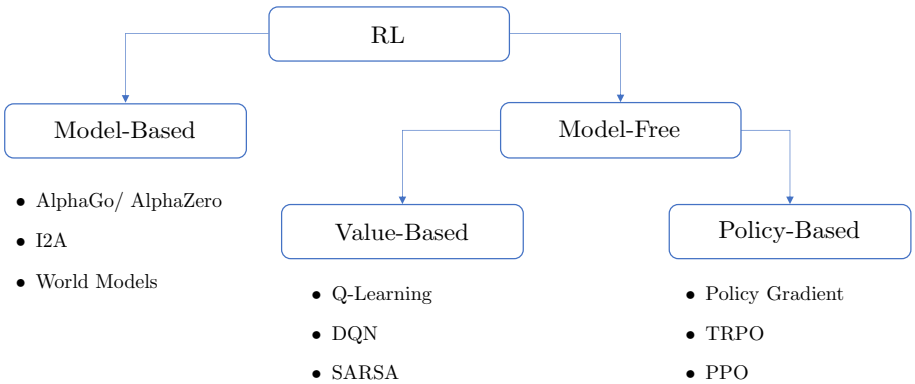


Figure 3.2.: Taxonomy of Reinforcement Learning, including the most recent implementations of each domain

Model-Based vs. Model-Free Approaches

Firstly, the RL domain can be divided into Model-Based [29, 30], and Model-Free attempts at the top level. In RL, a model explicitly describes the agent's environment and acts as an oracle that can be requested to receive future information. This could be the transition to the upcoming states and the according future rewards. Evidently, in Model-Based RL, the algorithms have access to such predictors. This access could further be constrained. For instance, AlphaZero [31], an RL approach that gained attention for its superhuman performance in games like Chess and Go, demonstrates the tremendous success of model-based methods. Nevertheless, the spotlight of research recently has shifted more towards model-free techniques since, in most real-world scenarios, the computation and access to such models are excessively expensive or even impossible. This, for instance, applies to the circumstances in the experimentation in this thesis, as there does not exist a perfect model for the given task. Therefore, model-based approaches are not considered in this work. Consequently, this Section continues by introducing two major model-free approaches in RL: Value-Based methods and Policy-Based methods.

Value-Based Approaches

To achieve the goal of maximizing the cumulative reward, Value-Based methods focus on learning a value function that intends to assign a value to each state. This can be achieved by utilizing two types of value functions: the *State-Value-Function* [3, 21], estimates the expected future reward in every state, assuming that the agent follows the optimal policy. On the opposite, the *Action-Value-Function* [21, 32, 33] estimates the expected future reward for each action in the current state, assuming that the agent follows the best strategy after performing the selected action. Note that both strategies are very similar and can be transformed into each other. Both methods assume that, from the following state onward, the agent follows the best strategy. Subsequently, the policy can be derived from the agent always seeking the most beneficial states. These methods work very intuitively and, thus, provide great insights into RL, and, moreover, they are comparably easy to implement.

Another benefit of Value-Based approaches is that they work *off-policy* [3]. This means that gathering data through interaction with the environment can completely be separated from training the policy. Hence, training on samples at any time and using them multiple times during one training is possible. Popular algorithms are, i.e. DQN [4], Q-Learning [3, 32], SARSA [3, 33] and C51 [34]. Nevertheless, Value-Based methods have rather shown poor performance solving more complex tasks as they, for instance, tend to be less stable [35]. The instability is mainly caused by training on an abstraction of the policy instead of directly concentrating on learning the optimal policy. This is exactly the objective of Policy-Based, also called *Policy Optimization*, techniques.

Policy-Based Approaches

In contrast to Value-Based methods, Policy-Based techniques, also known as *Policy Optimization*, focus on directly optimizing the policy π through *On-Policy Learning* [3]. This method is considered more stable and reliable, and, subsequently, the research in this area has led to tremendous success [36, 37]. This results in the emergence of many promising algorithms, such as Advantage Actor Critic (A2C) [38], Asynchronous Advantage Actor Critic (A3C) [38], Trust Region Policy Optimization (TRPO) [39] and Proximal Policy Optimization (PPO) [40], which, by using Policy Optimization, achieved outstanding results in complex problems. Especially, PPO has recently attained enormous success and is considered a state-of-the-art RL approach and outperforms almost every other approach in the most complex situations.

3.1.3. Proximal Policy Optimization Foundations

As already mentioned, PPO is considered state-of-the-art in the area of Reinforcement Learning. As shown in Section 2, there are several experiments in which PPO has been successfully used in order to achieve similar objectives compared to this thesis. This Section explains the algorithm in more detail by introducing Vanilla Policy Gradient, the initial approach that constitutes the fundamental ideas that are utilized by the Proximal Policy Optimization algorithm. Following, the most recent implementation of Proximal Policy Optimization is examined.

Vanilla Policy Gradient

To understand PPO, it is essential to comprehend the methodology behind Policy-Based approaches since Proximal Policy Optimization can be considered as an enhancement of these methods. Therefore, initially, the Vanilla Policy Gradient (VPG) [26, 36] algorithm is explained, as this is the fundamental algorithm in this domain. As determined in Definition 7, the policy π is a function, which in policy optimization problems is described by a set of mutable parameters θ. The assignment of the probabilities over the possible actions in each state is changed by adjusting these parameters. Consequently, this leads to a change in the agent's behavior and, in addition to that, to different rewards. Conceptually, in Policy-Based methods, the goal is to modify the parameter θ to maximize the overall reward. To successfully achieve this, the first step is to formulate the objective as a function:

$$L(\theta)^{VPG} = \mathbb{E}_t[log\pi_\theta(a_t|s_t) * A_t] \tag{3.4}$$

Equation 3.4 represents the VPG objective function or the VPG loss. Therefore, it multiplies an *advantage* A_t to the probabilities of the actions computed by the policy π. Policy-based algorithms introduce the advantage function A_t to assess the value of taking

a specific action. A_t quantifies the advantage of taking an action a_t in state s_t over other actions. Typically, as in the scope of this thesis, the advantage is calculated following the theory of Generalized Advantage Estimation (GAE) [26, 41]. Therefore, A_t is computed by subtracting the approximated future reward predicted by the function from the actual discounted sum of rewards. Note that the discounted sum of rewards is the sum of all formerly collected rewards multiplied by a constant < 1. As a result, rewards that are farther in the past are less regarded than the most recent rewards. However, the approximation of the future reward is computed by a value approximation function V_ψ, which is typically expressed by a neural network described by the parameter ψ. This network is additionally trained throughout the training process, mainly by using regression fitting. To summarize, A_t captures whether the real reward and, thus, also the performance of the executed action, was higher or lower than expected.

Subsequently, by multiplying the probability of the action with its advantage, the objective function encourages the policy to increase the probability of actions that have a positive advantage and decrease the probability of actions with a negative advantage.

In terms of RL, optimizing the objective function means finding those parameters θ that maximize L. This will consequently lead to the greatest advantage and, thus, to the highest reward. Finally, this conducts to the optimal policy introduced in Definition 10.

In traditional Policy-Based methods, the parameters θ are updated by following the direction of the gradient in order to reach the maximum of the objective function:

$$\theta \leftarrow \theta + \alpha \nabla_\theta L(\theta) \tag{3.5}$$

The influence of the learning rate α on the performance of policy gradient methods, as indicated by Equation 3.5, is significant. The learning rate determines the magnitude of parameter updates and plays a crucial role. It is important to note that an excessively high learning rate, similar to other tasks in Machine Learning, misses the maximum of the objective function and leads to a strong change of the policy in a negative direction. This often results in the policy being too far away from the optimal policy. This concludes with the policy finding a local maxima in the unfavorable area. In reality, this often results in a strange behavior of the agent, in which it is then stuck forever. Despite the superior stability of on-policy gradient-based methods compared to other approaches, certain limitations still need to be addressed.

Proximal Policy Optimization

In the previous Section, Vanilla Policy Gradient was introduced as the fundamental algorithm within the area of policy gradient methods. However, VPG suffers from certain limitations. One particular problem is its tendency to make too large policy updates, which often results in unstable training and sub-optimal convergence. Therefore, Proximal Policy

Optimization is a subsequent enhancement that addresses this issue. To achieve this, PPO intends to better control how the policy evolves by observing the policy change rather than having a simple update rule based on the gradient. The change of the policy is quantified with ρ:

$$\rho_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)} \tag{3.6}$$

Thus, deriving from Equation 3.6, ρ presents the ratio between the policy with the previous parameterization θ_{old} and the updated policy with the new parameters θ. Therewith, it constitutes a measure of the change of the policy.

Now, the objective function can be adjusted such that the policy change is considered as follows:

$$L(\theta) = \mathbb{E}_t[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)} A_t] = \mathbb{E}_t[\rho_t(\theta) A_t] \tag{3.7}$$

Following Equation 3.8, the aim of the objective function, on the one hand, is still to increase the probabilities of actions with positive advantages, which leads to an improved performance. Nevertheless, on the other hand, it considers the difference between the new and old policy. Since the advantage function is calculated using rewards gained while following the old policy, the policy ratio corrects for the change in probabilities. This enables a smoother transition between policies and prevents large and unstable updates. Thus, it improves stability and leads to convergence in the training process.

By constraining the policy change, PPO challenges the issue of large updates and provides more stable training. This enhancement is used across several modern variations in the domain of policy-based RL , such as Trust Region Policy Optimization and, as well, as in different sub-forms of PPO like PPO with Adaptive KL Penalty (PPO-KL) [40] and PPO — Variant with clipped objective (PPO-CLIP) [40]. Historically, TRPO was developed first, and subsequently, this led to the development of PPO-KL as an easier-to-implement version. Nowadays, PPO-CLIP is the most recent implementation and is considered state-of-the-art. Hence, to align with the scope of this work, the subsequent discussion introduces the PPO — Variant with clipped objective in detail.

In PPO-CLIP the objective function from Equation 3.8 is fine-tuned as follows:

$$L^{CLIP}(\theta) = \mathbb{E}_t[min(\rho_t(\theta)A_t, clip(\rho_t(\theta), 1-\epsilon, 1+\epsilon)A_t] \tag{3.8}$$

In Equation 3.8, the objective function accounts for the minimum between the product of the policy ratio and the advantage A_t, and a clipped version of the policy ratio within the range of $(1-\epsilon, 1+\epsilon)$ multiplied with the advantage. Empirical results from recent research have indicated that $\epsilon = 0.2$ is generally a statistically robust value. This will further be explained in Chapter 2. However, the maximal degree of change is further restricted by clipping the policy ratio to a specific range. This implies that the new policy can not

deviate too much from the old one. This is a major change, as now the policy can not be pushed too far away from the former behavior. This led to a tremendous enhancement compared to VPG. Subsequently, the policy update can be achieved by searching the parameter θ that maximizes the objective function L^{CLIP}:

$$\theta_{k+1} = \arg\max_{\theta} L^{CLIP}(s, a, \theta, \theta_k) \tag{3.9}$$

Finally, the following algorithm describes the practical implementation of PPO-CLIP:

Algorithm 1 PPO-CLIP

Require: initial arbitrary policy parameter θ_0, threshold ϵ

1: **for** k = 0, 1, 2 ... K **do**
2: Collect a set $\mathcal{D}_k$ of partial trajectories τ, on policy $\pi_k = \pi(\theta_k)$
3: Estimate Advantage A_t on current value function V_{ψ_k} ▷ Using any advantage estimation
4: Calculate the policy update by maximizing the objective:

$$\theta_{k+1} = \arg\max_{\theta} L^{CLIP}(s, a, \theta, \theta_k)$$

▷ SGD Adam [10] is used
5: Fit the value approximation function V_ψ by regression on the current collected reward R_t
6: **end for**

Algorithm 1 provides a description of the implementation of PPO-CLIP, aligned with explanations in this chapter. However, it is necessary to acknowledge that there may exist additional variations and enhancements of the PPO-CLIP that are not addressed within the scope of this thesis. During the experimentation, this specific implementation will be utilized together with the empirically derived recommended parameter values. Note that the parameter K defines the maximum number of steps during one training. This training could contain different numbers of episodes, as the step size for one episode can differ. Furthermore, multiple samples are used in the SGD update. This means that a small buffer is used to store a small number of the most recent state-action-reward-triples. Thus, the update can be computed considering multiple samples at one time, which further accelerates the training time.

A last crucial part in Reinforcement Learning is the trade-off between exploring the agent's environment and exploiting the actual policy. As PPO follows a stochastic on-policy, the executed actions are random at the beginning, depending on the initialization of the parameters. The confidence of the agent about deciding which action to take in which state increases over the time that the agent interacts within the environment. This results in a less random and, therewith, less exploring behavior. In practice, a term is added to the objective function and works as a regularizer. The influence of this term is controlled by the entropy coefficient β. Advancing the entropy leads to a more random-acting agent. However, as in all RL algorithms, the agent can still get stuck in local optima.

3.2. Simulation

Facing the objective of designing a sufficient algorithm that should solve complex problems by using real existing machines in reality often leads to the issue that these algorithms can not be tested in the real environment. There are various reasons for this issue, such as safety, costs, the existence of machines, or time limitations. Therefore, Simulations offer the possibility to reproduce these environments and make them easy and fast accessible. This tremendously accelerates the development of the algorithms and is used in plenty of tasks in the domain of Reinforcement Learning. This thesis uses Unity as a platform to build the simulation. Therefore, Section 3.2.1 introduces the fundamentals to understand its core concepts. Additionally, Section 3.2.2 explains the ML-Agents Framework, an extension for Unity, that simplifies the interconnection between the simulation and the machine learning algorithms.

3.2.1. Unity

Unity [42, 43] is a real-time 3D development platform that employs a modern physics engine and rendering system and, thereby, facilitates a variety of simulation tasks, including video games, filmmaking, computational physics research, and machine learning experiments that necessitate simulated environments. Furthermore, the Unity Editor provides an intuitive and interactive graphical user interface that allows the design of environments by attaching physical laws and external scripts. This Section aims to explain the most crucial components of Unity concerning the limits of this thesis.

Unity Projects

Firstly, Unity Projects are composed of multiple scenes. A *scene* itself is physically independent of other scenes and constitutes a virtual environment within the project. Hence, a scene refers to a specific configuration of objects, lights, cameras, and other elements with specific properties, interactions, and scripts. For example, in a Racing Game, a scene could be one track containing the drivers and cars, the surroundings such as the spectators, the street, the goal line, or any decoration. These obstacles are called *Game Objects* (GO) and build up the primary entities in a scene. However, to give GOs specific properties and behaviors *Components* are attached to them. Unity provides a wide range of components, including Rigidbody for physics simulation, Mesh Renderer for visual design, and scripts for adding custom logic. Furthermore, all Game Objects in Unity are organized in a hierarchical structure, which allows managing the relationships between objects and determines their interactions, positioning, and transformation. Furthermore, Unity provides its own engine to compute a realistic simulation of physics the *Unity Physics Engine*

Unity Physics Engine

The Unity Physics Engine is used to achieve a realistic simulation of the physics in the environment. Therefore, it implements a plethora of different techniques. Firstly, by adding a *RigidBody* to a Game Object, physical properties like gravity, mass, or velocity can be attached. Additionally, Unity provides a Scripting API, which can be used to add adjustable scripts to the Game Objects. Hence, it is, for instance, possible to apply forces on the object. An example could be to design a vehicle as a GO and attach a RigidBody. Subsequently, properties such as mass and friction can be assigned to the object in addition to associating it with a motor engine script. This script allows to apply acceleration to the wheel and can be triggered on certain events, like pressing the forward button. For such common tasks, Unity provides a bunch of libraries that meet the requirements to solve them, but additionally, it is also possible to self-implement algorithms. Note that Unity mainly approximates the continuous motion of objects by dividing it into discrete time steps and interpolating between them. Thus, it is possible to call the *FixedUpdate* method in order to receive the physical state of an object at any discrete point in time. This is very useful when combining it with Reinforcement Learning as, for instance, it is possible to request the state at any time step.

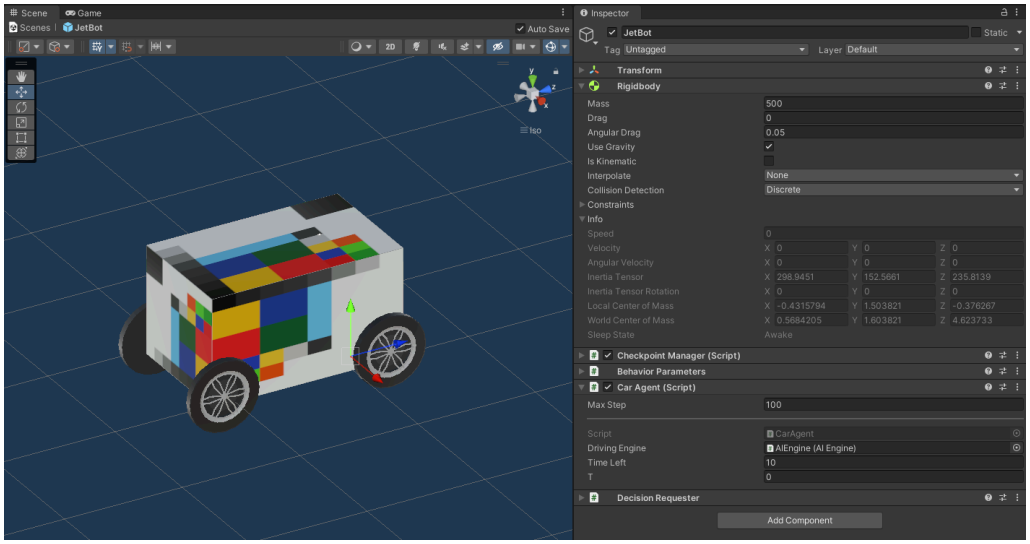


Figure 3.3.: Graphical interface of the Unity Editor, showing simulated environment with a game object on the left side and the property panel with properties, physics and custom scripts on the right side

However, Figure 3.3 shows an example of a Game Object that constitutes a vehicle with four wheels. The graphical interface is on the left side, and on the right side, there are the adjustable properties and the added scripts. For instance, the mass is set to 500, and an angular drag is defined. Furthermore, some custom scripts, like the CarAgent script, are added. However, another crucial component in the physics simulation is collision detection. Therefore, *Collider* objects can be added to Game Objects. By various collision detection algorithms, Unity is able to detect any intersections between colliders and, therewith, can compute the physical interaction between every item in the environment. Furthermore,

collision detection is useful to trigger certain events. Thus, it is possible to introduce invisible checkpoints, which, for instance, can be triggered when they are hit by a vehicle. For example, this technique is used in the experiments in this thesis to detect if a vehicle passes through or passes by a goal.

3.2.2. ML-Agents

This thesis utilizes Unity to simulate a virtual environment in order to research in the field of machine learning. A fundamental part of the project is constituted by the open source framework *ML-Agents* [3] [42, 44] as this provides a modern interface between simulation and Machine Learning. This Section outlines the principles of the ML-Agents framework.

ML-Agents is designed to integrate with Unity seamlessly. It provides a unity editor extension that allows setting up training scenarios, visualizing them, and monitoring the training progress. Furthermore, it is possible to train agents from scratch. Therefore, it provides optimized implementations of Reinforcement Learning algorithms, including PPO. With ML-Agents, it is possible to configure all crucial parts of the RL algorithm, including the hyperparameter. It is possible to define an agent and its actions and feed the algorithm with customized observations and rewards that are computed after interacting with the environment. The ML-Agents framework utilizes the advantages of Python and popular machine learning libraries such as TensorFlow [45] and PyTorch [46] to implement the algorithms. This ensures a very efficient implementation that can easily be further adjusted in the Python code. Using Python in the background provides plenty of advantages. Another main advantage is that the Deep Neural Networks (DNN) that result from the training are stored in a *onnx*-file. This ensures that they can be transferred and reused in any Python script. This enables to train Reinforcement Learning algorithms in the simulation, exporting the neural network to Python, and running the script in the real world. Thus, a state-of-the-art research pipeline from simulation to reality can be built by using ML-Agents.

[3] `https://github.com/Unity-Technologies/ml-agents`

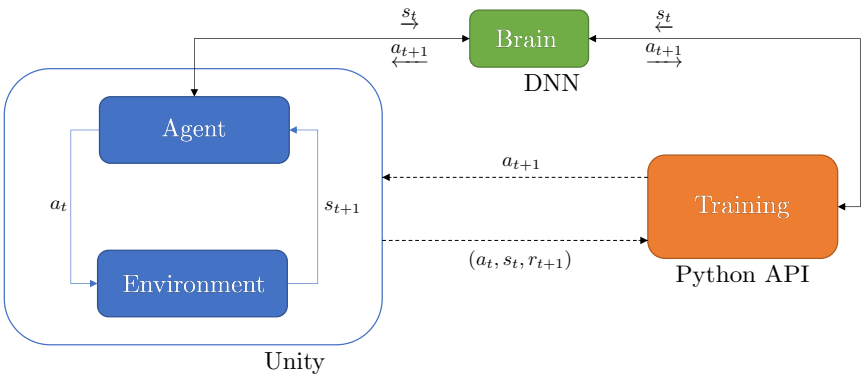


Figure 3.4.: Implemented training architecture of the ML-Agents framework when using it together with Unity

Figure 3.4 visualizes the training and inference architecture that is shared across Unity and Python using the ML-Agents framework. ML-Agents provides a communicator that ensures stable and efficient communication between the Unity environment and the Python code. Apparently, from the Figure, Unity sends the observations of every iteration to the Python API. There, the training is conducted, and, in training mode, the neural network is requested for the next action to take. Subsequently, the actions are replied to Unity and can be performed in the simulation. Meanwhile, in the background, the algorithms are trained on the Python side. For inference, the brain can be decoupled from Python and imported into Unity. Thus, the experiments can be examined efficiently without depending on communication with Python.

4. Implementation

To successfully be able to perform the experiments, in order to answer the research questions stated at the beginning, it is necessary to construct the simulation and training environment and implement the image pre-processing and the algorithms. Thus, this Chapter introduces the concepts for realizing this objective. Therefore, Section 4.1 provides all the details required to build the simulation. Following, Section 4.2 introduces the utilized image pre-processing, and, lastly, Section 4.3 describes the core components of the implemented training.

4.1. Simulation

This research utilizes Unity to develop a dedicated simulated environment that matches all requirements. This Section briefly explains how Unity was used to construct the simulation. Therefore, it is split into three subsections: Section 4.1.1 seeks to explain how the environment, including the arena and the surroundings, is built, Section 4.1.2 introduces all details of the simulated vehicle, and, following, Section 4.1.3 describes the design of the goal and map including the spawning mechanism.

4.1.1. Arena and Surroundings

Simulations are generally used to model reality virtually. This work utilizes an actual arena, which is housed at ScaDs, an IT research institute, as a role model. Thus, this arena is replicated in the simulation, i.e. the simulated arena has the same dimensions as the real world model. This Section constitutes the realization of the arena in the simulation. Figure 4.1 shows the finally constructed arena in unity. Note that one *grid* in the simulated environment serves as one meter in reality, which means that the constructed arena is one meter wide and two meters long, exactly as in reality. A *unit* in the simulation is expressed by one-tenth of a grid. Thus, one unit refers to $10cm$.

However, the arena's boundaries are represented by yellow walls, which can detect collisions with the vehicle. Moreover, the environmental parameters of the arena, including its background, can be customized according to the specific requirements of each individual camera itself. This means changing the background outside of the arena for every camera is possible. Furthermore, placing and configuring different light sources in the environment is feasible. The lighting and shadows will be computed according to the given properties. During the training and seeking of suitable parameters, the environment's light setting is optimal ambient, meaning there are no shadows, and the color of every object can perfectly be observed from every angle. Hence, this further helps the computer vision, as this will significantly reduce noise in the camera picture.

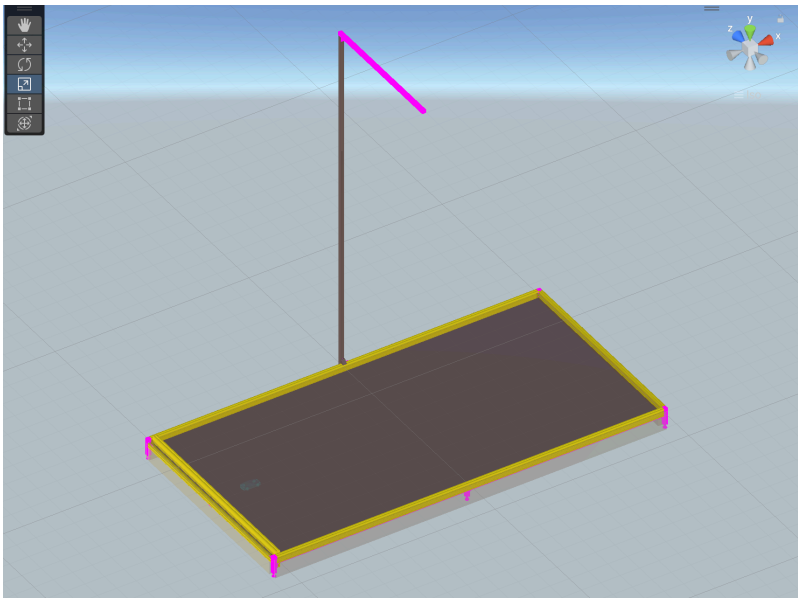


Figure 4.1.: The arena in the simulated environment in Unity

4.1.2. Vehicle

Besides the arena, another fundamental component of this project is the vehicle itself. This Section provides information on the implementation of the simulated vehicle. Continuing on the work of [2], this thesis uses the NVIDIA[4] JetBot[5] robot as inspiration. Thus, this robot is rebuilt in the virtual environment, following the dimensions and measures of the robot and the attached camera. Table 4.1 provides an overview of all requirements that need to be fulfilled, and Figure 4.2 shows the simulated robot in unity.

Moreover, the JetBot has a Camera that provides the visual input for the Reinforcement Learning algorithm and needs to match the requirements in Table 4.1 as well. However, for the purposes of this experiment, it is desired that there is a significant contrast between the color of the background and the goals. This facilitates easier detection of obstacles by

[4] https://www.nvidia.com
[5] https://jetbot.org/master/

	Component	Measure
Vehicle	Robot length	14 cm
	Outer spacing of both wheels	11 cm
	Inner spacing of both wheels	9.4 cm
	Wheel diameter	6 cm
	Wheel thickness	0.8 cm
Camera	Camera height	9 cm
	Rotation downwards	20°
	Field of View	60°
	Resolution	1640x1232

Table 4.1.: Shows the required dimensions of the Jetbot in the reality

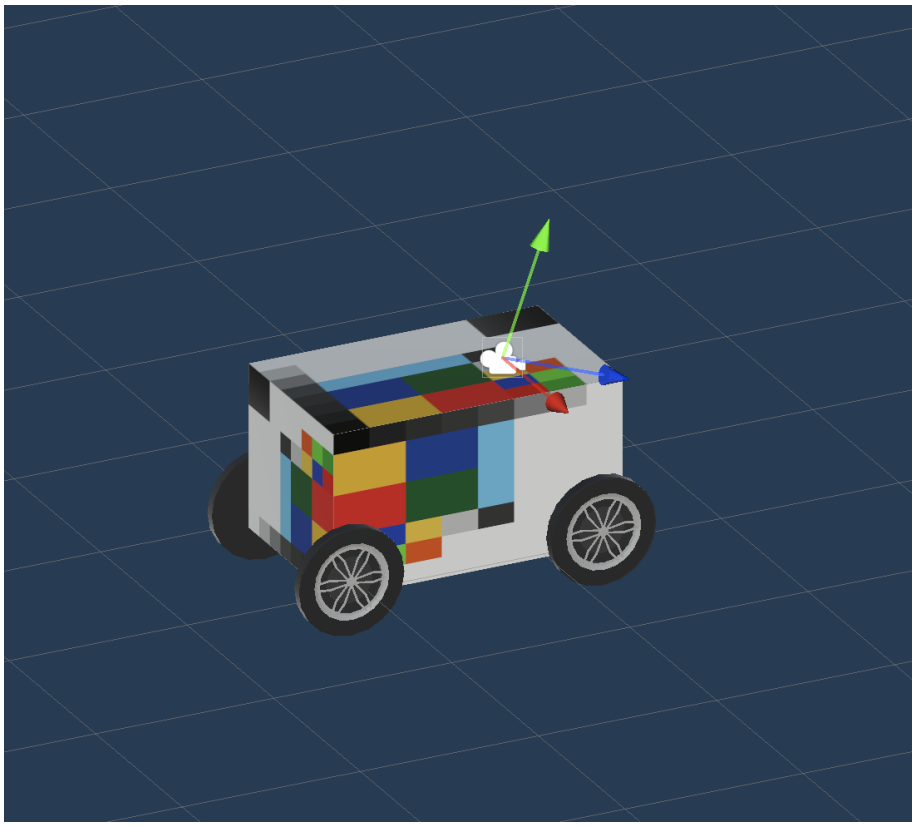


Figure 4.2.: The vehicle in the unity simulation

computer vision. As the background is fully configurable for each camera, the camera of the JetBot sees the background in pure white. Figure 4.3 shows a view from the vehicle's camera. It is evident that the background appears in white color.

However, the most important part of the vehicle is its control and motor engine. Therefore, in the manner of the Jetbot, it has two individual motor engines, one motor engine for each front wheel, but no proper steering control. This means that the steering is achieved implicitly via differential wheel speed, known as differential steering [47]. Hence, the vehicle's turn direction is controlled by varying the relative speed of the wheels. For instance, if the engines of both wheels apply the same force, the robot drives straight forward, but if the left motor operates at a higher speed, the robot will turn right. In Unity, the wheel is represented by a *WheelCollider* component with a property called *MotorTorque*. The MotoTorque is the acceleration force that is applied to the wheel. In this work, it is self-defined as a given input between -1 and 1. This input, i.e., can be the output of the neural network. This value is subsequently multiplied by a general value *MaxTorque* of $100Nm$ that transforms the input to an acceleration force. This means a force between $-100Nm$ and $100Nm$ is applied to every wheel. Note that both motors are perfectly linear. In addition, a drag force is applied to the vehicle to model the friction. The value of the drag force is set to 1.5, together with a mass of the vehicle of 500. Unfortunately, Unity does not provide the calculation or a formula for the drag force. Therefore, the used values have been found by, initially, following the used values in the work of [1]. Subsequently, those

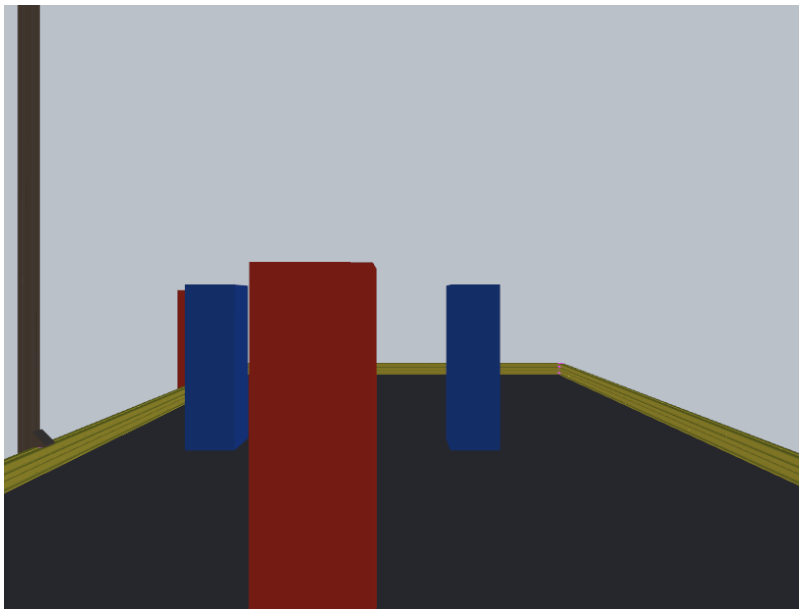


Figure 4.3.: View from the vehicle's camera in the simulated arena

values were adjusted through trial and error until a natural driving behavior of the robot appeared, which was close to the driving behavior of the JetBot in the real world.

4.1.3. Goals and Map

In addition to the vehicle and camera setup discussed in the previous Section, the goals and map components play a crucial role in the simulated environment, as they provide a structured setting for the vehicle's navigation and define the objective for the learning algorithm. This Section explores the characteristics of the goals and map layout in detail.

The goals in the simulated environment consist of pairs of equally colored posts. The color of each goal can either be blue or red. These goals serve as target objects for the vehicle to navigate between them. The goalposts themselves can detect collisions with the vehicle, and thus, certain events can be triggered. The distinction in color facilitates easy identification and differentiation of the goals by computer vision. Strategically placed throughout the map, the goals present a challenging driving course for the vehicle. The number of goals varies from three to four in one map. These goals are positioned in a manner that requires the vehicle to navigate through them sequentially, alternating between a red or blue goal. The goals are generated randomly within the simulated environment during the training process. However, the following requirements for the map's goals must be matched. The width between the posts of each goal is randomly determined within a range of 0.2 meters to 0.4 meters, and the distance between each goal is randomized between 0.6 meters and 0.8 meters. This ensures that it is possible for the vehicle to complete each map and prevents, i.e., that a goal is too small to pass it. With those requirements, it is possible to generate different maps.

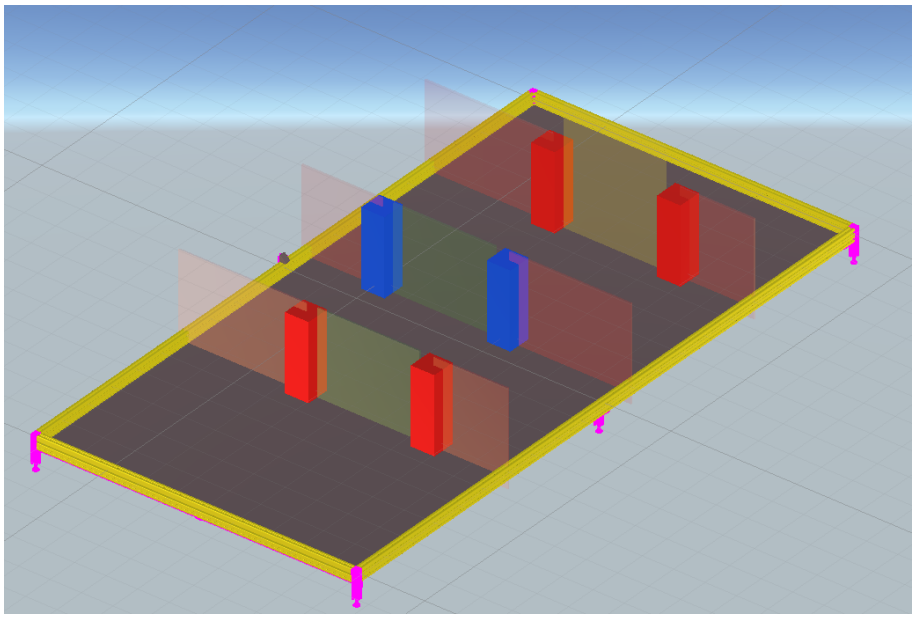


Figure 4.4.: Example of a generated map in the simulated environment, including the walls (yellow), three goals (red and blue) with the goal missed line (light red), the goal passed line (light green), and the finish line (light yellow).

Furthermore, lines are attached to the goals to assess the progress and performance of the vehicle. These lines are only visible in the Unity editor and can not be seen from the vehicle. They only serve as triggers to detect a pass or miss of a goal and the completion of the map. The invisible lines include a light green line and a light red line. The light green line checks if the vehicle successfully passes through a goal. Conversely, the light red line checks if the vehicle misses a goal and drives past by, allowing it to identify any errors in its navigation. The posts themselves have the same triggers and can also detect hitting them. The map also includes a for the vehicle's invisible finish line, which is light yellow and is placed in the last goal, where it substitutes the goal-passed line. By crossing the finish line, the vehicle indicates that it has successfully completed all the goals in the specified order and accomplished the map. This can be used to differentiate between passing a goal and completing a map afterward.

Figure 4.4 shows an example of a generated map in the simulated environment, where all explained components can be seen. The yellow boundaries represent the walls. The first goal is blue colored, and the green line has to be passed by the vehicle. The red lines detect if the vehicle misses a goal and drives by. It can be seen that the color of the goals is alternating and that the last goal contains a yellow finish line.

4.2. Image Pre-Processing

A crucial part of the research of this thesis is the utilization of images for training the machine learning algorithms. This often presents a significant limiting factor in the training procedure, primarily due to the exponential increase in input complexity. Therefore, in

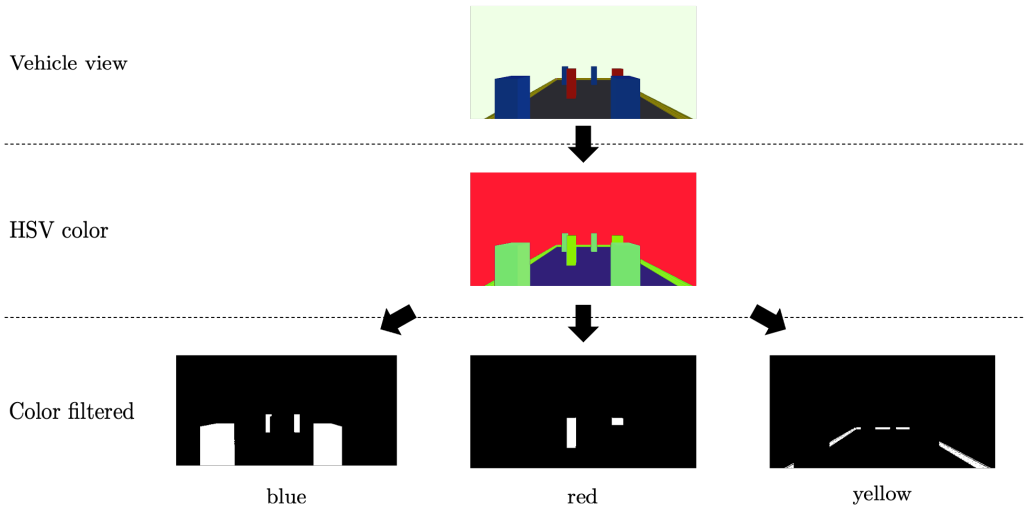


Figure 4.5.: Example of the image pre-processing pipeline where the original image (top) is converted into the corresponding HSV colors (second stage) and the resulting black-white images after filtering the HSV color picture by the particular color red, blue, and yellow.

most machine learning tasks that work with images, there will be an image pre-processing before feeding the pictures to the algorithm. Hence, following the preliminary work of [1, 2], this work utilizes *OpenCV* [48], an open-source image processing library. This Section briefly explains the image pre-processing implemented in this thesis.

The following pipeline processes every image input of the camera. Firstly, the image is converted into HSV-Colors [49], as the HSV color space provides more perceivable information than RGB, which benefits the pre-processing. Following, thresholds of the most critical colors, red and blue, for the goals posts and yellow for the wall, are defined. The image, subsequently, is filtered using those color thresholds and returns a black-white image. Figure 4.5 illustrates this processing pipeline. The first stage shows the original image recorded by the camera. The following stage shows the image converted into HSV colors, and lastly, stage three provides the black-white images of the filtered colors blue, red, and yellow. However, the filtered image is then processed by the *FindContours* method of OpenCV, which identifies the boundary points of all recognized red, blue, and yellow objects in the form of rectangles. Note that the coordinates of those points are represented in image coordinates and not in world coordinates.

The detected contour points are used to generate boundary rectangles, which summarize the contour points into rectangular shapes. Each rectangular consists of x and y coordinates in the image and a height and width. Every rectangle is the outer boundary of a detected, connected obstacle. Small rectangles that may be considered as noise are filtered out to ensure accuracy.

After the *FindContour* method is applied, there is now a list of rectangles of all recognized red and blue goal objects, as well as yellow objects representing walls. Figure 4.6 visualizes this process and shows an example of a state s_t after processing the camera input at time

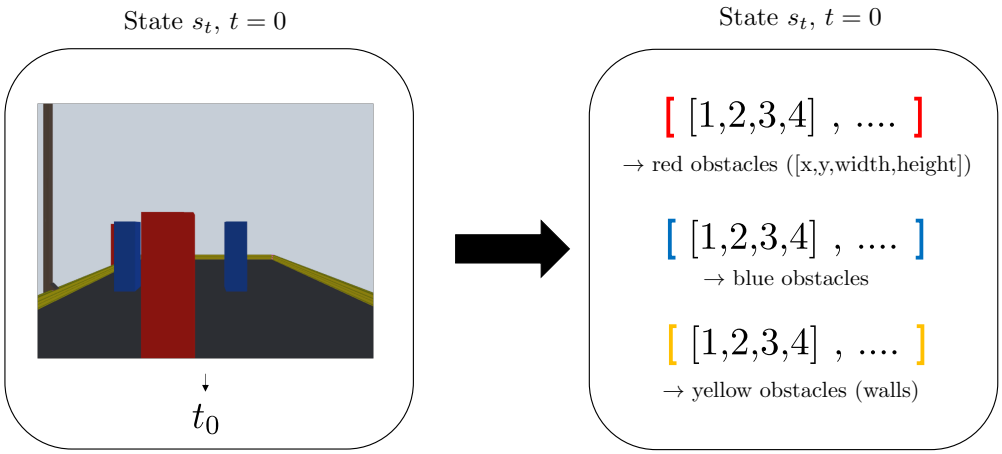


Figure 4.6.: Example of the image at time step t=0 captured by the vehicle's camera and the corresponding computed state s_0 after the pre-processing

step t. From the right side, it is apparent that there is a list that contains another list of rectangles for every object named above.

However, to further reduce the complexity, only a limited number n of these recognized objects are used as input for the Reinforcement Learning algorithm. The idea here is that providing the coordinates of all detected objects is probably unnecessary because the vehicle only needs the information about the next goals or, at most, the two next goals to navigate through the map successfully. Thus, the number of fed obstacles is limited to the four closest recognized obstacles per type.

It is evident that the contours could contain small errors or inaccuracies in the image coordinates, i.e., due to lighting and shadows, image noise, and inaccurate thresholds. However, the information about the position, width, and height of the obstacles implicitly provides the RL algorithm with the relative positions of the obstacles in relation to the vehicle. Hence, even though these may not entirely correspond to their true-world coordinates, as this error is continuously and constantly part of the input of the RL algorithm, the algorithm should still be able to navigate the vehicle successfully through the map. The idea is that the RL algorithm is designed to find the optimal policy and maximize the overall reward according to the given states, and, hence, it includes the error in its calculations.

However, the error is introduced by not taking the actual picture but rather intending to extract the obstacle coordinates. Nevertheless, it is indispensable to accept this trade-off, as the achieved reduction in state size is tremendously beneficial. Even if the image would be grey-scaled to 8 bits per pixel, since the resolution of the image is 1640x1232 pixels, the state size of only one image would be $16,163,840$. By pre-processing the image, as described above, and feeding only the next four blue posts, four red posts, and four rectangles representing the wall, where each rectangle consists of four values, the state size is reduced to $(4+4+4)\times 4 = 48$ different states. This reduction amounts to approximately 99.9997%

of the grey-scaled image's state size. Thus, despite the slight inaccuracy introduced by the pre-processing, the significant reduction in state size greatly benefits the training process, making it more efficient, and, therefore, rectifies this processing.

4.3. Training Architecture

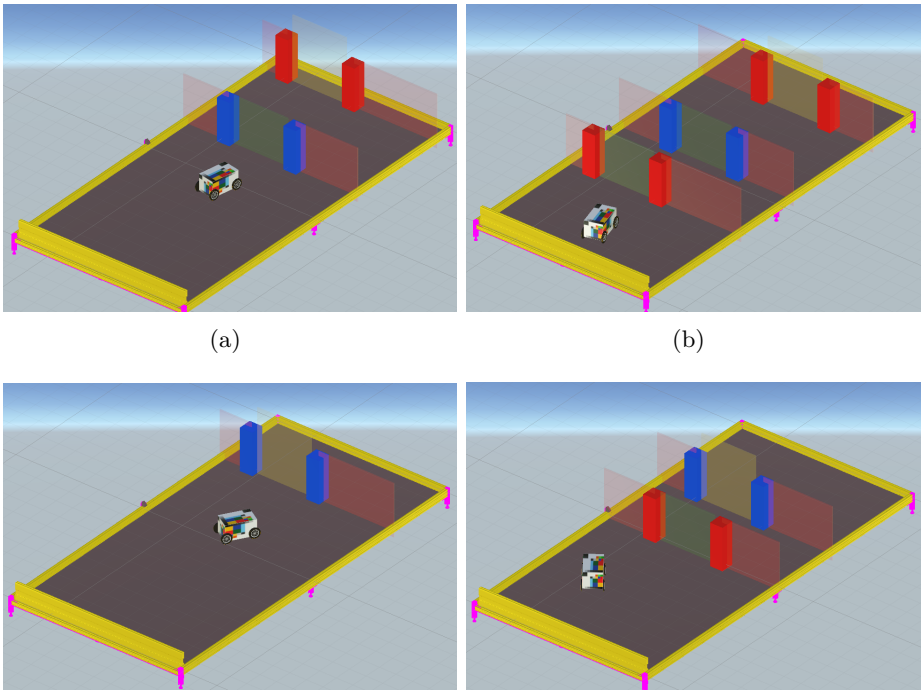
The training process is a crucial component in successfully solving any machine-learning task. This likewise applies for RL. In Reinforcement Learning , the training process focuses on iterative improving the policy, gradually converging towards the optimal policy that leads to the optimal solution. This Section provides an overview of the typical training setup and introduces the core concepts employed in the experiments with different algorithm configurations. Thus, the Sections from 4.3.1 to 4.3.9, each introduce an elementary part of the training setup.

4.3.1. Initialization

All core components must be initialized at the beginning of the training. This, on the one hand, includes the environment, as described in Section 4.1, which is generated first, including the arena with the walls and cameras. Subsequently, the vehicle is placed at a determined spawn point, and the first map is initialized. This means that the goals for the training are randomly placed, and the line checkpoints are attached according to the restrictions explained in Section 4.1.3. On the other hand, the RL parameters are initialized, including arbitrarily setting the policy and the weights of the neural networks. The reward is set to 0, and the first state is initialized.

4.3.2. Spawning the Vehicle

As mentioned, the vehicle is spawned on the map at the beginning of each episode. Therefore, a random position within the map is determined. A minimum distance from the end of the map is respected, and the robot, at least, has to pass one goal. Furthermore, the rotation of the vehicle at the start position randomly varies between $-45°$ to $45°$ degree towards the obstacle track. Figure 4.7 shows four different possible spawn positions of the vehicle. This ensures that the start position is uniformly distributed across the map, situating the vehicle in different positions and angles and, thus, different states. This explicitly affects training the last goal more frequently, as it has a different state as there is a wall behind it. This procedure aims to achieve a more generalized and stable algorithm.



(a) (b)

(c) (d)

Figure 4.7.: Four distinct examples of different spawn positions and angles of the vehicle and different map configurations after the initialization

4.3.3. Training episode

Once everything is initialized, the training begins. A full training is divided into training episodes, each further divided into smaller steps, denoted as t. A training episode in this context refers to an Episode in RL, as defined in Definition 9. It is apparent that the self-driving vehicle is the Reinforcement Learning Agent from Definition 4, and the Unity environment constitutes the RL Environment, introduced in Definition 3. Figure 4.8 illustrates one iteration t of a training episode in this context. It is apparent that it matches all required core components from the RL cycle explained in Section 3.1.1.

However, an episode starts after the initialization or a re-start after the end of the previous episode. Hence, the vehicle is spawned at the starting point and consumes the initial state. Note that the state, action, and reward will be described in more detail later. Following the diagram in Figure 4.8, the policy evaluates the state and returns the actions. Consequently, the agent executes the actions, which will lead to a new state, according to the state transition function from Equation 3.1. After that, the environment calculates the reward, based on the state change, according to Equation 3.2. Note that in every step, the agent collects a state-action-reward-triple. Afterward, the next step $t + 1$ of the episode begins. Passing multiple iterations leads to trajectories of state-action-reward-triple that refer to the trajectories introduced in Definition 8. Hence, at any time step of an episode, a history is represented by the according trajectory. This trajectory implicitly includes the course that is driven by the vehicle.

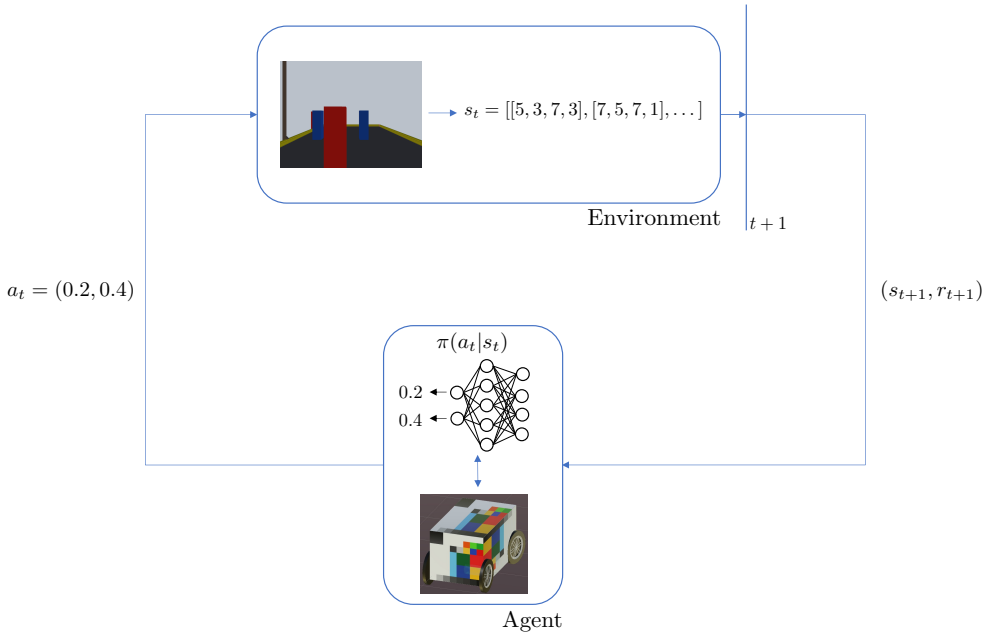


Figure 4.8.: Top-level view of the core components of RL modeled in this thesis applied to the RL cycle

However, an episode is terminated, according to a termination signal. This could, for instance, be running out of time or hitting an obstacle. The exact terminal signals used will be introduced in the following paragraph. Note that a full training consists of several episodes that are distinct from each other. In the case of this thesis, the end of a full training is determined by $K = 3 \times 10^6$ steps.

4.3.4. Termination of an episode

The termination of an episode is an essential aspect of the training process, as it can accelerate the training tremendously. In the scope of this work, there are several abortion signals defined that appear after certain events. This Section introduces all implemented termination signals.

Out of time is an event, that is triggered after the *Global Environment time* T that is set to $T = 8s$, runs down to 0. This is the time for the vehicle to complete an episode. There are training variations, where 10 seconds are added to this time for every goal the vehicle correctly passes through. Those two values appeared reasonable after testing to drive through the maps manually. Within $8s$ plus $10s$ for every goal, there is enough time to complete a map. This accelerates the training tremendously, as it resets the vehicle and prevents it from getting stuck, which would cause the episode to take much time. This, particularly, is important at the beginning of the training, when the agent has not yet learned that it has to drive forward. In this case, the vehicle stays at the initial position. By aborting after 8 seconds and punishing this behavior, the agent faster learns that it has to drive forward in order to reach the next goal.

Name	Parameter	Value
Clipping range	ϵ	0.2
GAE Parameter	λ	0.95
Discount factor	γ	0.99
Entropy Coefficient	β	0.01
Learning rate	r	0.003
Maximum training steps	K	3×10^6
Buffer Size		2048
Batch Size		512
Number of epochs		3

Table 4.2.: Overview of the Hyperparameter of the PPO algorithm

Furthermore, an episode is reset after the vehicle hits an obstacle. This includes the *Hit post* event, which is triggered when the vehicle collides with a goal post, and the *Hit wall* event, which is triggered after the vehicle touches a wall. Moreover, an episode ends if the *Goal missed* event is prompted, which indicates that the vehicle passes by a goal. All of those events constitute unwanted behavior of the agent during the training process. Thus, the training is massively accelerated by instantly terminating the training and penalizing this erroneous behavior.

Additionally, there are positive cases in which the episode ends. This could either be after correctly passing through the first goal, which will lead to the *Goal passed* signal, or by reaching the finish line that marks the map as completed and triggers the *Map completed* event.

It is important to note that episodes can have different lengths due to these termination conditions.

4.3.5. Shared Hyperparameter

From Chapter 2, it is apparent that hyperparameters are clearly a bottleneck of RL algorithms. Therefore, this Section introduces the shared hyperparameters that are used across all experiments. Note that the parameter configurations are mainly drawn from reviewing the literature mentioned in Chapter 2. Most of the studies provide overviews of the used parameters together with brief explanations.

Hyperparameter of the PPO algorithm

This Paragraph introduces the hyperparameters controlling the Proximal Policy Optimization. Therefore, the study in [40] constitutes the main influence for the final configuration. Nevertheless, all studies from Chapter 2 have been considered in the final decision for the parameter values.

Clipping range ϵ: is set to $\epsilon = 0.2$, which has been pointed out as a statistically robust value in recent research. The clipping range determines the maximum deviation the policy can make from its former version.

GAE Parameter λ is set to $\lambda = 0.95$. This parameter is a smoothing parameter for the Generalized Advantage Estimation and reduces the variance in training. This makes the training more stable [40].

Discount factor γ is set to $\gamma = 0.99$ and determines how strong states that are more in the future are considered in the calculation of the expected future reward. The bigger γ, the more farsighted the algorithm works [4].

Entropy coefficient β is set to $\beta = 0.01$. This parameter adds entropy to the objective function, increasing the exploration of the agent and making the policy act more randomly.

Learning rate r is set to $r = 0.003$ and defines the step size in each gradient descent update.

Maximum training steps K is set to $K = 3 \times 10^6$ and determines the number of training iterations performed in one training. The training terminates after K steps.

Buffer Size defines how many State-Action-Reward-Triples are collected before the policy is updated. The value is set to $2,048$.

Batch Size is set to 512 and determines the fraction of collected tuples from the buffer that are used during one update. Using the buffer together with the *Batch Size* means collecting a set of experiences of the *Buffer Size* and using a subset to train the algorithm on the collected experiences.

Number of epochs defines the number of subsets of the *Batch Size* that are drawn from the buffer. In this case, the value is set to three. This means that for every *Batch Size* step, three SGD updates are done on *Batch size* collected triples.

Hyperparameter of the policy neural network

Subsequently, this Paragraph introduces the parameters that are used for configuring the DNN that models the stochastic policy. The DNN consists of an input layer that consumes the state s_t as input. The number of input neurons corresponds to the size of the state. The network is fully connected. After the input layer, there are several hidden layers and, finally, the output layer, which comprises as many neurons as the size of the action space.

Normalize is set to true and, thus, normalizes every input before feeding it into the network.

Number of layers constitutes the number of hidden layers in the Deep Neural Network and is set to two.

Hidden Units is set to 256 and indicates the number of neurons, that each hidden layer has. Hence, in total, the neural network holds 512 hidden neurons.

Iterative Improvement and Fine-tuning

Furthermore, it is important to mention that the experiments in this work differ in some crucial parts from the work in the previous studies. Although they share similarities, the parameter configuration that led to success in the other studies are not necessarily applicable in the experiments in this thesis. Therefore, all parameters were tested and further evaluated over two months. In this period, the algorithms have been trained, and the performance has constantly been monitored. Hence, potential areas for improvement have been identified, and thus, some of the parameters subsequently have been adjusted in an iterative fine-tuning process. This is a common process in RL and leads to improving the learning algorithm.

4.3.6. Updating the policy network

The most crucial part of the training process in the Proximal Policy Optimization algorithm is updating the policy network. This Section explains how the policy network is updated during training.

Figure 4.9 illustrates this process. The part on the left side of the Figure is managed by the unity engine. This comprises the simulation, in which the agent navigates the vehicle through the map. On the right side, in the orange outline, there is the part that updates the policy. This part is implemented in Python and communicates with unity as explained in 3.2.2. The two parts can be decoupled from each other, which makes this environment modular, and thus, the training algorithm could easily be substituted for further investigations. From the Figure, it is apparent that both parts share live data, according to the interaction of the agent with the environment, which includes all the necessary information that the algorithm needs to train. It is important to note that

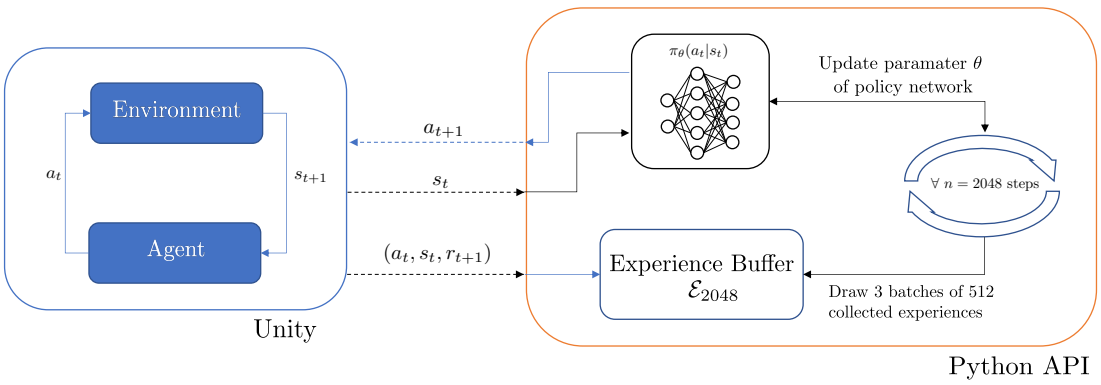


Figure 4.9.: Top-level of the implemented updating procedure of the policy during training

updating the neural network is done in parallel to using it to predict the next action, which the agent should perform in the current state. This means the neural network is trained and updated while using it to navigate the vehicle.

During the interaction with the environment, the agent collects partial trajectories consisting of state-action-reward triples and stores them in an experience buffer. The size of the experience buffer is defined by the buffer size, which in the scope of this work is set to 2048. Periodically, for every 2048 training step, three subsets of size 512 (batch size) are drawn from the experience buffer. These subsets are then used to update the policy neural network. Updating the policy network means pushing the parameters θ in the direction where the objective function is maximized, following the equation from 3.9. Section 3.1.3 briefly describes this update process.

Algorithm 2 PPO-CLIP Training Implementation

Require: Initialize arbitrary policy parameter θ_0

1: **for** training step t = 0, 1, 2 ... K **do**
2: Construct the map and place all obstacles on the map
3: Spawn the vehicle in s_0
4: Initialize state s_0 of current episode
5: $EpisodeIsRunning \leftarrow True$
6: **while** $EpisodeIsRunning$ **do**
7: Let the agent navigate through the environment, following the current policy $\pi_k = \pi(\theta_k)$
8: Collect a set $\mathcal{D}_k$ of size *BufferSize* of state-action-reward triples
9: Estimate the Advantage A_t on current value function V_{ψ_k}
10: Use $NumberOfEpochs \times BatchSize$ samples from the experience buffer to calculate the policy update by maximizing the objective function:

$$\theta_{k+1} = \arg\max_{\theta} L^{CLIP}(s, a, \theta, \theta_k)$$

11: Fit the value approximation function V_ψ utilizing the current collected reward R_t
12: **if** Termination Signal **then**
13: $EpisodeIsRunning \leftarrow False$
14: **else**
15: $t \leftarrow t + 1$
16: **end if**
17: **end while**
18: Reset the environment
19: **end for**

The policy update process is repeated throughout the entire training process with the goal of pushing the policy toward the optimal policy. Note that the objective function L from Equation 3.8 uses another neural network besides the one that models the policy. This network is called the value network. It calculates the advantage A_t as described in Algorithm 1. This network seeks to estimate how valuable a specific state s_t is by predicting the future reward from this state onward. In order to ensure an accurate estimation, this network also needs to be trained. Therefore, it is trained by computing the loss using the

Mean Squared Error (MSE) between the predicted reward and the actual collected reward. Algorithm 2 summarizes the implementation of the PPO-CLIP, which is used across all configurations in this thesis as pseudo-code. Note that the state, action, reward function, and termination signal vary between the different algorithms.

4.3.7. Multi Agent Training

In Figure 4.10, it can be seen that multiple agents can be trained simultaneously. The reason is that every training arena is an identical copy of each other, and the trained agents share the same network that is updated. Thus, they use the same model. This means the training is exactly done as explained, and due to the fact that the episodes are independent of each other, multiple agents can train in different episodes at the same time and contribute to the same model. For the used hardware, introduced in Section 5.1.1, training four agents in parallel leads to the maximum possible speed for training. Training more agents leads to side effects that cost more computational power and, thus, lead to slower training. Note that on other machines, this can be different.

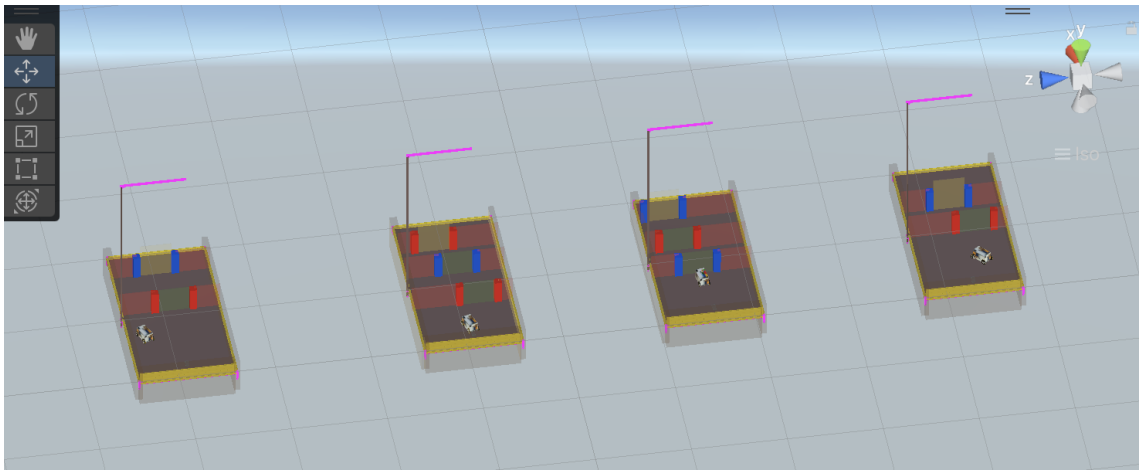


Figure 4.10.: Unity simulation showing multiple agents training at the same time

4.3.8. Training Configurations

This Section introduces two different training configurations that are used during the training of the algorithm. The Single-Goal-Training (SGT) Full-Map-Training (FMT), which are used to train the algorithm and examined in the evaluation.

Single-Goal-Training

The SGT configuration's objective is to capture as many different situations as possible. This is achieved by ending an episode after the vehicle reaches the first goal using the *Goal passed* termination signal introduced in 4.3.4. The idea is that this approach can speed up the training process, as each episode ends much faster and the vehicle is spawned much more often in different positions and angles, and, hence, has the chance to learn on more

different data. However, it may not cover all situations. For example, if the vehicle needs to drive through two goals and the second goal is offset, it may be necessary to drive in a certain path through the first goal to be able to reach the second goal. Such situations where it is necessary to consider the relation between multiple goals can not be covered in this configuration, as the agent's objective is only to reach the first goal. Nevertheless, this approach is still interesting. Since the scope of this thesis is very limited in time, the number of training steps that can be taken is probably not sufficient to completely solve the task. This approach considers this trade-off as it simplifies the task and sacrifices possible useful information but accelerates the training. Furthermore, in many Machine Learning tasks, simplifying the problem and training has led to surprisingly good performances. Thus, the hypothesis for this approach is that, despite the lack of information, the policy will converge faster towards the optimal policy, and, in the limited time, it will outperform other approaches.

Full-Map-Training (FMT)

In this configuration, the objective is to train the vehicle to pass through all the goals and reach the finish line. The training episodes are not aborted after the first goal. This ensures that the agent learns to maneuver the vehicle through the entire map and reaches the final destination. However, since the episodes are longer, the training process is probably slower compared to the SGT configuration, but, on the other hand, it challenges the problem of the lack of information of Single-Goal-Training, and it can abstract the relation between goals. Furthermore, in this configuration, it is possible to differentiate between a goal and the final goal of the map, making it possible to return an extra reward for completing the full map. In this case, the hypothesis is that the agent focuses on completing the full map rather than only passing the next goal. This should lead to a higher success rate in finishing courses.

4.3.9. Exploration of Parameters and Configurations

This section introduces the method used to determine promising parameter, reward, or state configurations. Before experimentation, there was a phase of two months to seek potential configurations that should be further examined in the experiments and evaluation. As already mentioned in 4.3.5, the results of previous studies serve as a basis for all hyperparameters. This includes the design of the neural network and the reward function. Nevertheless, this work differs from the other works in some crucial aspects, such as the design of the vehicle or the overall objective. Thus, it is very likely that it needs another proper configuration.

To find potential parameters, subsets of configurations were created based on the results in previous studies, which are further investigated. The hyperparameters that have been consistent across all the previous work have directly been adopted in this work. Additionally, various reward functions have been considered, taking inspiration from previous

Figure 4.11.: The tensorboard dashboard that offers live visualization of the training progress of the examined learning algorithms

work. Nevertheless, as the experiments differ in major aspects from other works, the reward functions have been designed novelly by considering proper ideas, such as giving a reward for speed. The most reasonable designs for reward functions have been evaluated and will later be introduced.

However, to test the configurations, the agent is trained in a Single-Goal-Training setting to achieve the fastest learning process. The decision of whether a configuration was considered as good was based on two factors. Firstly, by visually observing the agent's behavior in the simulation. This enabled to detect issues early in the training stage. An example of an issue is that the agent only drives in circles to gain speed reward and, thus, is stuck in a local optima. In this case, the training was aborted, and the examined configuration was considered inappropriate. Secondly, ML-Agents serves as a great opportunity to observe live training data, as it provides a dashboard using tensorboard.

4.11 shows the dashboard analyzing different learning algorithm examples. It is evident that the dashboard tracks plenty of live measurements. The most important metrics that have been utilized are the Cumulative Reward and the entropy. The Cumulative Reward tracks the development of the sum of all rewards in a training episode. In a successful training episode, this value should increase over time. Note that the Cumulative Reward is described in more detail in Section 6.1. By observing this, it was possible to detect attempts with a much slower increase in the reward. This helped to abort the training early and not consider the tested configuration. Using this strategy led to the decision for

the final hyperparameter in Section 4.3.5 and the current design of the algorithms. On the other side, the entropy indicates the level of randomness that the reinforcement learning agent uses while acting in the environment. The smaller this value is, the less random the agent acts. Thus, in this case, the agent will not be able to make further progress. Therefore, if this value is small and the agent still acts poorly, the training can be aborted as well.

4.4. Proximal Policy Optimization Algorithm Using Pre-Processed Camera Input

Following the preceding Sections, which introduced all common configurations and settings, this Section introduces the final examined configurations of the Proximal Policy Optimization algorithm.

However, to successfully reach the objective of this work, the used algorithm is an implementation of the PPO — Variant with clipped objective, which is an RL algorithm as explained in Section 3.1.3. According to the Reinforcement Learning cycle from Figure 3.1, it is apparent that the state, action, and reward need to be modeled to be able to formulate the problem in this thesis as a RL problem and answer research question 1.

Action

The set of actions $\mathcal{A}$ is equal across all different algorithms and comprises accelerating the left motor engine at any time point t a_{0_t} and accelerating the right motor engine a_{1_t}. The acceleration is clipped between -1 and 1, which means in every step, the policy network predicts a value between -1 and 1 for every action. According to Section 4.1.2, the behavior of the car represented in Table 4.3 is derived from the relation of these actions.

a_0	a_1	relation $a_0 : a_1$	Behaviour of the vehicle
$a_0 = 0$	$a_1 = 0$	$a_0 = a_1$	No acceleration
$a_0 > 0$	$a_1 > 0$	$a_0 = a_1$	Driving Straight forward / Accelerating
$a_0 > 0$	$a_1 > 0$	$a_0 < a_1$	Driving forward & Turning left
$a_0 > 0$	$a_1 > 0$	$a_0 > a_1$	Driving forward & Turning right
$a_0 < 0$	$a_1 < 0$	$a_0 = a_1$	Driving Straight Backward / Breaking
$a_0 < 0$	$a_1 < 0$	$a_0 > a_1$	Driving backward & Turning right
$a_0 < 0$	$a_1 < 0$	$a_0 < a_1$	Driving backward & Turning left

Table 4.3.: Overview of the agent's behavior according to the values of action a_0 and a_1

State

Considering the state, there are two major differences in the PPO-CLIP algorithm that are used. One approach is to use a *basic state* by directly feeding the obstacle coordinates that are gained by the processing, as explained in Section 4.2, to the algorithm. To keep the state space small, the number of obstacles is limited to the four nearest observed obstacles per type. This implies that the state includes the nearest four red posts, four blue posts, and four detected wall coordinates. As explained, the coordinates constitute rectangle bounding boxes, described by x and y coordinates and the height and width of the bounding box. The state is filled with vectors containing zeroes if less than four obstacles are detected. The algorithm that uses this representation of the state will further be called Proximal Policy Optimization Without Memory (PPO-BASIC). Figure 4.6 displays the final modeled state's shape after pre-processing one image at time step t.

However, besides this state representation, an additional idea appeared promising. This approach introduces a simple kind of memory, making the agent capable of remembering its own traces. For instance, Figure 4.12 illustrates a situation in which the agent without a memory possibly has a weakness. The Figure shows the initial view of the vehicle on the left side and the view after the vehicle turned to the right on the right side. It is apparent that after the turn, only one goalpost is left in the view. By having the information of the first image, it is evident that the vehicle has to turn back. But without knowing this, in the second picture, it needs to be clarified on which side the other goalpost is, and the policy network might predict to turn right to find the second post. In such cases, it could be useful to provide memory so the vehicle remembers that it has already seen the full goal. Therefore, to challenge this problem, a memory is added in the form of passing the last five processed pictures to the agent, which means the agent always has access to the coordinates of the four states before the actual one and the actual one.

Adding the memory of the n last steps introduces another hyperparameter that can be further investigated. The exploration indicated that five is a promising value. This algorithm, which uses PPO together with memory, will be called Proximal Policy Optimization Using Memory (PPO-MEM). Figure 4.13 visualizes the process of constructing a state with memory. On the left side, there are five recorded images of the camera. The first picture is the captured one in the current state s_5. The underlying images are from the former

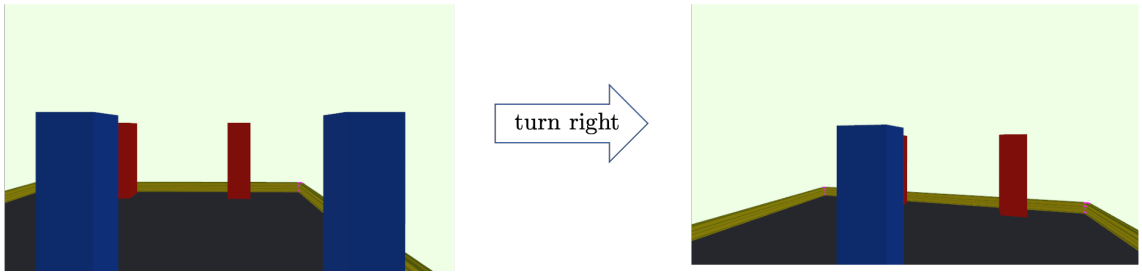


Figure 4.12.: Example where a state with memory is useful. On the left side is the vehicle's camera view before a turn, and on the right side, after turning right

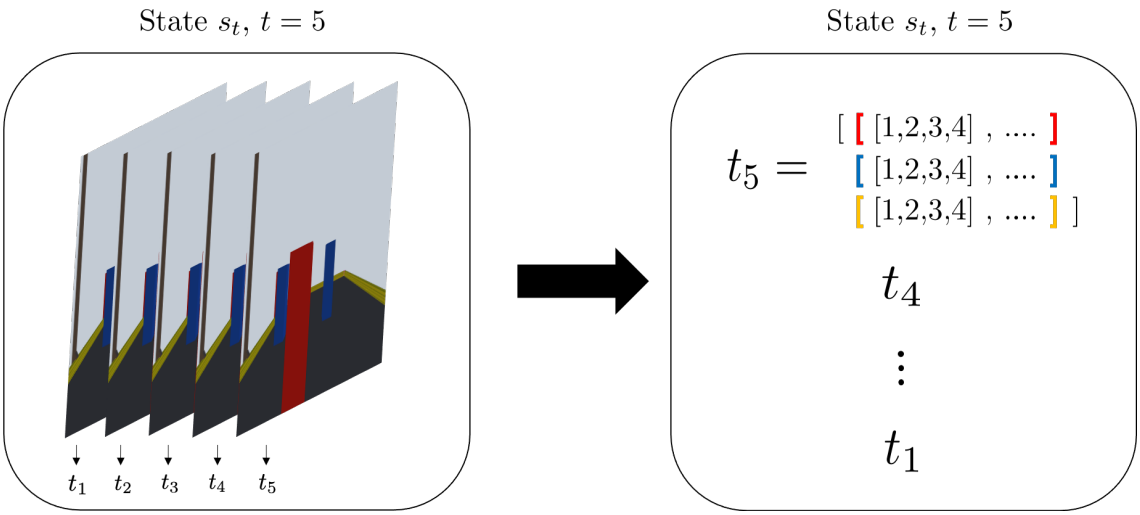


Figure 4.13.: Construction of a state s_5 at time step $t = 5$ including the four last pictures before the actual frame as memory

states s_4, s_3, s_2, s_1. The right side shows the final data structure that constitutes the state at time step 5. This includes the crucial coordinates of the current state and the four states before.

Reward

As already mentioned in Section 4.3.9, the reward functions in this work introduce a novel approach. This approach is inspired by the studies introduced in Section 2.

Following the recent work from Section 2, engineering a good reward function for the specific task is crucial to gain a successful learning process besides tuning the hyperparameter. In this work, there are two major different training settings, the Single-Goal-Training and Full-Map-Training, that lead to very different episodes. In this paragraph, one particular reward function is introduced for each of the training settings that covers the specific situations and should lead to successful learning.

In the SGT scenario, the reward function $R_{SGT}(s_t, a_t)$ is defined as in Equation 4.1:

$$R_{SGT}(s_t, a_t) = \begin{cases} (v/10) * \Delta T, & \text{for } \forall t \\ 1, & \text{for } \textit{Goal passed} \\ -1, & \text{for } \textit{Goal missed} \\ -1, & \text{for } \textit{Hit post} \\ -1, & \text{for } \textit{Hit wall} \\ -1, & \text{for } \textit{Out of time} \end{cases} \tag{4.1}$$

It consists of one dynamic reward and five static rewards that are returned on certain events. The dynamic part of the reward is represented by the current velocity v of the

vehicle, which, additionally, is multiplied by the fraction ΔT. ΔT is the time duration of one step t. By multiplying this, the reward for the velocity is kept low, considering a complete episode. This part of the reward has two major advantages. First, it encourages the agent to drive forward, which, especially in the beginning, accelerates the training, and second, it motivates the agent to drive faster and, thus, to complete an episode faster. Note that it is essential to avoid the velocity being excessively regarded. In this case, the agent would fall into a local optima, where its main objective changes to collect the most speed reward rather than finishing the map. For instance, the agent could drive in circles as fast as possible to collect the highest speed reward instead of finishing the map. To avoid this behavior, the velocity reward is further reduced by dividing it by 10.

Besides the dynamic reward, there are five static rewards. Apparently, only one of them is positive. This reward of 1 is given after the goal passed event is triggered, and thus, it rewards the agent if it passes the first goal. As explained, after passing the goal, the episode ends. On the other side, four negative rewards serve as a punishment for failures. This includes hitting a goal post, missing a goal, hitting a wall, or running out of time. This should incentivize the agent to avoid this behavior.

However, as the training situation in the Full-Map-Training differs, it is promising to model a novel reward function to capture this difference. Equation 4.2 presents the reward function $R_{FMT}(s_t, a_t)$:

$$R_{FMT}(s_t, a_t) = \begin{cases} (v/10) * \Delta T, & \text{for } \forall t \\ 1, & \text{for } \textit{Goal passed} \\ 100, & \text{for } \textit{Map completed} \\ -1, & \text{for } \textit{Goal missed} \\ -1, & \text{for } \textit{Hit post} \\ -1, & \text{for } \textit{Hit wall} \\ -1, & \text{for } \textit{Out of time} \end{cases} \tag{4.2}$$

This reward function attempts to take advantage of the difference in training that the vehicle now passes through multiple gates and completes a full map. This is done by providing an additional reward of 100 after passing through the last goal, which, furthermore, constitutes the only difference to the R_{SGT} reward function.

This emphasizes the importance of completing the full map, reinforcing the idea of the FMT. It also encourages the learning of goal-to-goal relationships, which are essential for effective navigation, as there are cases where passing through a goal in the future depends on the way the current goal is passed. The hypothesis is that this training should outperform the SGT approach in a long-term view.

Finally, Figure 4.14 shows all final approaches that are examined in the experiments. This includes both basic approaches, the PPO-BASIC-SGT and PPO-BASIC-FMT . And both approaches, that are using memory, the PPO-MEM-SGT , and PPO-MEM-FMT .

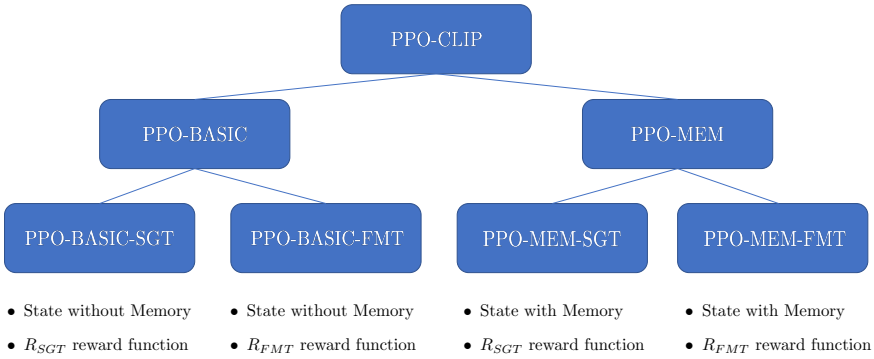


Figure 4.14.: Overview of the final configurations of the PPO algorithm that are examined in the experiments

5. Experimentation

Remembering the introduction, the goal of this thesis is to answer the four following research questions: 1. How can the problem of training an autonomous driving agent be effectively modeled in the context of reinforcement learning? 2. How can the camera input, which provides visual information about the environment, be processed and utilized to feed into the RL algorithm and achieve learning success? 3. How good is Reinforcement Learning in solving the problem of training an agent to drive a vehicle in the simulated environment and passing all goals? How robust are the developed Reinforcement Learning algorithms against varying external influences, including light settings and motor engine power? Chapter 4 introduces the ideas and their realization to be able to find answers to these questions. Subsequently, this Chapter introduces the experiments with which this work should gather valuable results in order to evaluate the ideas in terms of performance, stability, and robustness. Therefore, this Chapter is divided into two Sections: Section 5.1.1 provides an overview of the underlying hardware and software. Section 5.2 introduces the concrete experiments to assess the implementations.

5.1. Hardware and Software Setup

This Section focuses on the hardware and software aspects of the project. It provides an overview of the tools and technologies used for the simulation, machine learning, and visualization, as well as the hardware setup. Thus, Section 5.1.1 provides an overview of the hardware and Section 5.1.2 of the software.

5.1.1. Used Hardware

The hardware used for training and experimentation in this project is a MacBook Pro (2020) with a 2 GHz Quad-Core Intel Core i5 processor and 16GB of 3733 MHz memory. The MacBook Pro provides sufficient computational power to run the simulation environment and train the machine learning models. The use of this hardware across all experiments enables meaningful comparisons of training times and performances and, thus, ensures reliable and reproducible results.

5.1.2. Used Software

The simulation environment is developed using Unity with editor version 2021.3.1f1, a game development platform known for its realistic physics simulations., together with ML-Agents as an open-source Unity plugin, utilized for implementing the machine learning algorithms and training process as explained in Chapter 3.2. The simulation environment is implemented in C#, as this is required for the custom scripts in Unity. OpenCV [48], a

powerful computer vision library, is employed for image pre-processing and is embedded in the C# scripts. TensorBoard, provided by TensorFlow, is used for real-time training data visualization and can be configured in ML-Agents.

Besides TensorBoard, Python is used together with the libraries Matplotlib [50] and Seaborn [51] to write custom scripts to visualize the results. The library Pandas [52] is used to manage the data structures. Furthermore, Git is used for version control, and the finalized code can be accessed at GitHub[6].

5.2. Experiment Design

The previous Section briefly introduces the surroundings in which the experiments take place. This Section, consequently, discusses the experiment design that is used to evaluate the performance of the trained algorithms and their stability and robustness against external influences. Therefore, Section 5.2.1 explains the design of the different tracks that are used to examine the implemented algorithms and to perform the experiments. Subsequently, Sections 5.2.2 to 5.2.4 introduce the three concrete experiments that are evaluated in this work.

5.2.1. Evaluation Maps

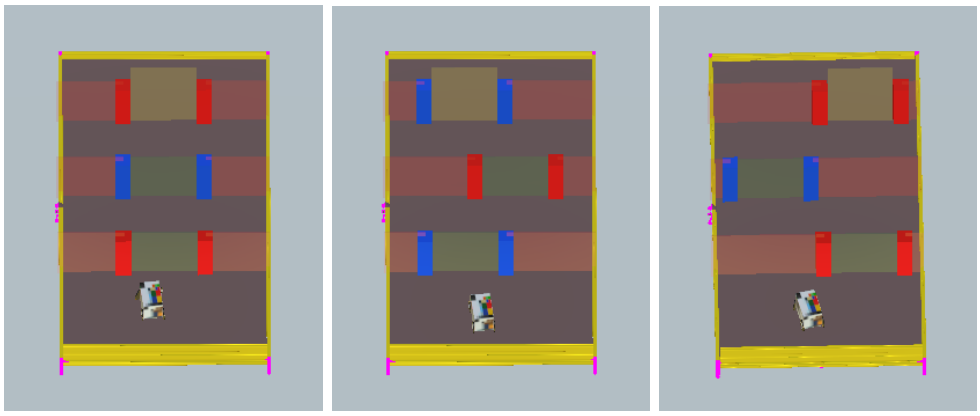
In order to ensure that the experiments produce comparable and valuable results, it is crucial to have an expressive map design. All experiments must be conducted on the same tracks to ensure that they can be compared with each other. This Section describes the utilized map designs in the experiments.

Therefore, three distinct maps are designed, each representing a different track difficulty. Figure 5.1 shows a track of each difficulty. The first track (a) in the picture shows an *easy* parkour. Successfully completing this track indicates that the algorithm can properly interpret the task, accelerate, and drive through the goals.

The second track (b) represents the *medium* difficulty. To pass all goals in this map, the algorithm needs to combine steering and acceleration correctly, demonstrating its understanding of the task and its ability to determine the position of goals, turn the vehicle, and find a path towards the following goal.

Lastly, illustration (c) introduces the most complex parkour, the *hard* track. This track is challenging for the agent as it adds more crucial problems. First of all, it requires the agent to navigate through highly steep curves. This is an exceptionally complex control task. Thus, the agent needs to be fully capable of correctly applying differential steering and controlling the acceleration of the two motor engines. Moreover, after passing the first goal, the agent will be in a situation where it does not observe the next goal to pass

[6]GitHub Code Repository. `https://github.com/Maddi97/master_thesis`



(a) (b) (c)

Figure 5.1.: Three tracks of the three different map difficulties that are used in the experiments to evaluate the performance of the algorithms. (a) Easy map (b) Medium map (c) Hard map

but rather the goal behind. Therefore, it has to make a complete turn to the other side, understanding that the visible goal is too far away to be the next target and, thus, that there must be a closer goal with another color. To solve this, the agent needs to understand the depth of field, differentiating between goals by their distance. Solving this challenge demonstrates a deep understanding of the problem and effective control.

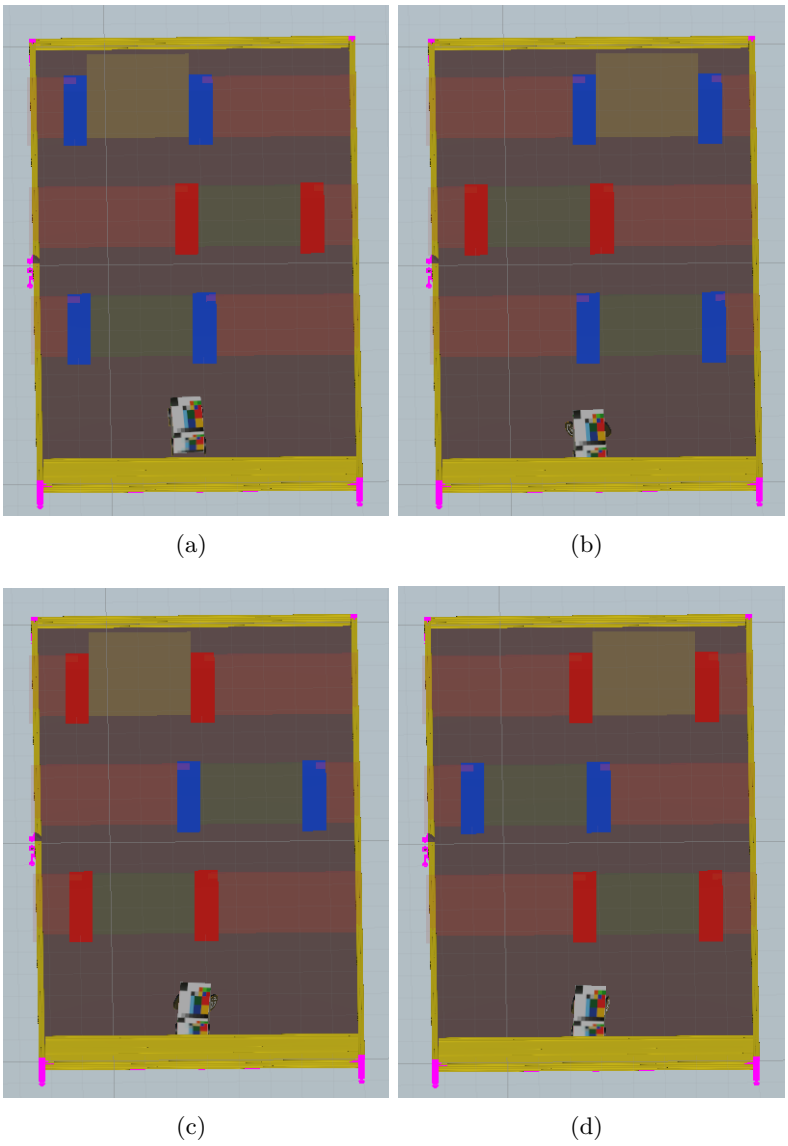
However, it is important to note that each map used in the evaluation has the exact same formation as shown in Figure 5.1 for the particular difficulty level. Nevertheless, different variations of the maps will be tested to evaluate all possible configurations of every formation.

For the *easy* map, two variations will be evaluated, one starting with a red goal and another with a blue goal to pass. This ensures that the algorithm's performance is assessed in different starting scenarios.

This is similar to the *medium* and *hard* parkour. Both comprise four different formations, where the map will vary between beginning with the first goal on the left or right side and starting with a red or blue goal. Figure 5.2 illustrates all four different configurations of the hard map. As mentioned, the same configurations apply to the medium map. Only the distance of the goals to the walls differ. It is important to note that the color of the goals alternate throughout the map, and the distance between two goals is fixed at 0.5m during the evaluation. Thus, every map has exactly three goals to pass. The same obstacles as explained in Section 4.1.3 are used in all experiments.

In total, ten maps will be tested: two maps for the *easy* difficulty and four maps each for the *medium* and *hard* difficulties. Subsequently, the following experiments will be conducted with 100 runs on each evaluation map, resulting in 1,000 runs for every algorithm in every experiment. This is necessary to compute the average performance of every examined test.

All four algorithms described in Section 4.4 will be examined in each experiment. This includes the PPO-BASIC-SGT, PPO-MEM-SGT, PPO-BASIC-FMT and PPO-MEM-FMT algorithms.



(a) (b)

(c) (d)

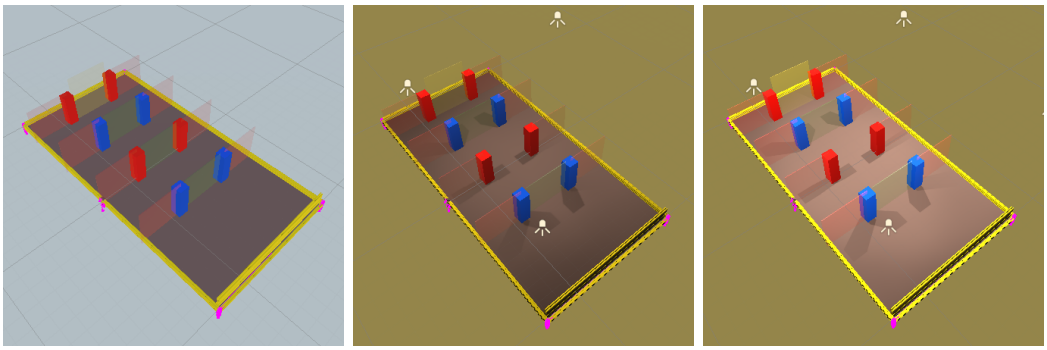
Figure 5.2.: Different configurations of the hard map. Every map has the same difficulty. The goals have the same distance to each other and to the walls. Figure (a) shows the map starting with the blue goal on the left side, (b) the map starting with the blue goal on the right side, (c) starting with the red goal on the left side, and (d) the map with the first goal being red and on the right side.

5.2.2. Experiment 1 - Performance in Optimal Conditions

In experiment 1, the algorithms will be tested under the same conditions as during training, using the ambient light setting as introduced in Section 4.1.1. This experiment specifically addresses the proposed research question 3 from the introduction, as it seeks to determine the general performance and effectiveness of the RL algorithms under optimal conditions. In this experiment, 1 scenario $\times$ 4 algorithms $\times$ 10 tracks $=$ 40 different runs are evaluated. Figure 5.3 (a) shows an easy map example with ambient light. It is apparent that there are no shadows, and the colors of the obstacles do not change.

5.2.3. Experiment 2 - Robustness against Different Light Settings

In Experiment 2, the algorithms will be tested under two different light settings: a dark setting and a bright setting, as shown in Figure 5.3. Six different directional spotlights constitute the light source in the experiment. Three of the lights are above each wall of the long arena side, where two lights are above the corners, and one is in the middle. The Figure illustrates that by using light sources, shadows appear across the map, and the colors of the obstacles vary with respect to the direction of the light. This probably causes problems with the image pre-processing, as it uses color thresholds to recognize the obstacles. If the color is changed to be outside the thresholds, the preprocessing will not work correctly. As it is a crucial part of the algorithms, this will cause major problems. This experiment examines the algorithms' stability and robustness in varying light conditions. Thus, Experiment 2 addresses research question 4. To summarize, this experiment includes three new scenarios in which the four algorithms are tested on the medium map difficulty. Limiting the tests to the medium maps is necessary, as it should provide a possibility to compare the results to the results in the perfect settings. To stick to the scope of this thesis, it is essential to reduce the number of maps that are used for evaluation, as it otherwise would take too much time. Hence, the 2 light settings of the four algorithms are tested on the four tracks of the medium map $=$ 32 different tests.



(a) (b) (c)

Figure 5.3.: Three tracks of the three different light settings, that are used in the experiments to evaluate the performance of the algorithms. (a) Ambient light setting (b) Dark light setting (c) Bright light setting

5.2.4. Experiment 3 - Robustness against varying motor power

Experiment 3 consists of two sub-experiments: one with 20 percent more motor power and another with 20 percent less motor power. To ensure consistency and focus on examining the engine power, the experiments will be conducted using the ambient light setting. This experiment aims to investigate the robustness of the developed algorithms against variations in motor power, thus answering research question 4, as proposed in the introduction. This experiment again examines the robustness of the algorithms by using the medium map difficulty. This experiment, in total, comprises 2 motor power settings $\times$ 4 algorithms $\times$ 4 tracks = 32 different test runs.

Table 5.1 provides an overview of all the test runs done throughout the experiments. In total, 104 tests are examined, each consisting of 100 runs. From previous tests, it could be assumed that one run will need approximately 10s to finish. This sums up to 104 tests $\times$ 100 $\times$ $10s = 100000s \approx 28h$. This undermines the importance of limiting the number of experiments as, in reality, the tests require extensive time, exceeding this thesis's scope.

	Algorithms	Experiment variations	Tests in Total	Objective
Experiment 1	4	1	40	Performance
Experiment 2	4	2 $\times$ light	32	Robustness
Experiment 3	4	2 $\times$ motor power	32	Robustness
			104 tests	

Table 5.1.: Overview of the examined experiments

6. Evaluation

This Chapter focuses on evaluating the proposed experiments in Section 5.2 and presents a comprehensive analysis of the training behavior and the performance and robustness of the algorithms in the experiments. Therefore, it is divided into three Sections: Section 6.1 offers an overview of the utilized metrics. Section 6.2 covers the efficiency of the algorithms in the training. Finally, Section 6.3 concludes the performance and stability of the approaches in the experiments.

6.1. Metrics

This Section explains the most crucial metrics used to evaluate the training process and assess the algorithms' performance in the experiments.

Cumulative Reward

The Cumulative Reward CR [3] derives from the reward introduced in Definition 6 and constitutes the sum of the total collected reward in one training episode and can be described by the following equation:

$$CR_T = r_0 + r_1 + ... + r_T = \sum_{t=0}^{T} r_t \tag{6.1}$$

One training episode refers to one agent's trial to navigate the vehicle through the map. In most cases, T is the end of the episode, but considering the equation, the cumulative reward can be calculated at any point in time of an episode, which has not necessarily to be the end. However, considering the two designed reward functions r_{SGT} from equation 4.1 and r_{FMT} from equation 4.2, it is apparent that the better the performance of an agent is in one episode, the larger the sum of all collected rewards should be. Therefore, as one training consists of multiple episodes, the Cumulative Reward is a great possibility to track the training process. In successful training, the Cumulative Reward should increase over the training time. Another used variation is the Normalized Cumulative Reward (NCR) [53]. From the different reward functions r_{SGT} and r_{FMT}, it is evident that the value of the Cumulative Reward that an algorithm can achieve in one episode distinguishes from each other, depending on which reward function they use. The NCR can still compare the training process of the different algorithms. Therefore, the Cumulative Reward is scaled in an interval of $[0, 1]$.

Success Ratio

The *Success Ratio* is used to evaluate the performance of an examined algorithm in an experiment. It expresses the percentage of successfully completed tracks. Note that all experiments are tested on exactly 100 tracks. This results in the percentage being equal to the number of tracks completed. The greater the success ratio, the more often the agents completed a full track in an experiment, and, thus, the more successfully the algorithm mastered an experiment.

Velocity

The *velocity*, which is used to evaluate the different approaches, states the average velocity in Unit/second that the algorithm had during an experiment. As explained in Section 4.1.1, one Unit in the simulation equals 10cm. This means that the velocity is specified in $10\frac{cm}{s} = 1\frac{dm}{s}$. An objective modeled in the reward function is to drive tracks as fast as possible. Therefore, the higher the velocity, the better the agent performed.

Time

Finally, the average *Time*, that an algorithm necessitates for finishing a map in an experiment, is measured. This value expresses the effectiveness of an algorithm in completing a task. Similar to the velocity, it is an objective that the algorithms need as little time as possible for completing a track.

6.2. Training

Besides the actual results, the behavior of the algorithms throughout the training is another exciting part to evaluate, as the efficiency of the training is a major cost of computational power in developing new machine learning algorithms. Thus, a crucial part is to design the algorithm so that the training is as fast and efficient as possible in order to save expenses or to achieve valuable results in a reasonable time. Therefore, this Section states the main issues during the training in Section 6.2.1 and assesses the training efficiency of the approaches in Section 6.2.2.

6.2.1. Challenges

Throughout the model training, several issues have appeared. This Section aims to address these challenges, provides an understanding of the solutions to overcome those, and supplies important conclusions for further research.

The first erroneous behavior was observed when the vehicle faced the last goal. In this stage of the training, the robot has been consistently positioned at the start of the map, always facing straight forward. It became evident that during the training, the vehicle encountered more problems attempting to drive through the last goal of the map. This can be explained as the states related to the last goal differ from other states, as there is a wall directly behind the last goal. Since the vehicle has to pass all other goals first in order to reach the last one in this training setup, it observed those states significantly less than other states. Thus, the agent trained those states viewer times and consequently needs to be more confident driving through the final gate. To solve this problem, the training has been adjusted such that the vehicle now spawns randomly across the map. Hence, there are training episodes in which the vehicle is positioned directly in front of the ultimate goal. This increases the number of training situations where the vehicle attempts the last goal. This significantly increases the overall performance.

In a separate trial, the agent has been trained without pre-processing the images but rather feeding them directly into the algorithm. Consequently, the results were unsatisfactory, as the agent showed minimal progress. The reason for this is that the state size of the unprocessed image is huge compared to the processed picture. Thus, the agent required considerably more time to be able to interpret the situations correctly. The introduction of the image pre-processing accelerated the training progress and led to noticeable improvements, as applied in the experiments [12, 13, 17]. Nevertheless, it was a promising trial to feed the raw image directly to the agent as it includes all information, whereas processing images often includes a loss of information. This is a trade-off between access to more information and the reduction of complexity.

An additional challenge arose while testing various reward functions. In some instances, providing a high-speed reward to the agent resulted in the agent losing sight of the overall objective. Hence, the agent becomes stuck in a local optimum. It prioritized gathering as much speed reward as possible rather than completing a track. This prevented the cumulative reward from increasing, and, in the simulation, the agent tried to drive in circles to accelerate as fast as possible. To address this issue, the reward for the velocity has been lowered.

Challenge	Solution
Rare appearance of states around the last goal	Randomisation of the spawn position of the car
Feeding raw images causes too complex state inputs	Pre-processing of the images
Agents loosing the object, because of too high velocity rewards	Lowering velocity rewards
Punishments stop the agents from making progress	Lowering punishments

Table 6.1.: Summary of the appeared challenges during the experiments

Lastly, in certain scenarios, an erroneous behavior appeared in which the agent would stop moving and wait for the end of an episode. This typically occurred when penalties were given to the agent, and either the punishment was too big, or the punishment for the out-of-time event in relation to the penalties was too small. Thus, the agent tried to avoid making any failures but rather let the episode terminate. The last two challenges are very common in RL tasks, and it is a general concept to examine different reward modelings until a promising result can be achieved.

Table 6.1 gives an overview of the appeared challenges and their solutions.

6.2.2. Training Performance

This Section provides insights into the performance of the different approaches PPO-BASIC-SGT, PPO-MEM-SGT, PPO-BASIC-FMT and PPO-MEM-FMT during the training.

Primarily, the performance is evaluated based on the development of the cumulative reward throughout the training period. By observing the development, it is possible to assess how fast the algorithm converges toward the optimal policy. With respect to the different reward functions used in the SGT and FMT approaches, they are analyzed separately.

Figure 6.1 illustrates the progression of the cumulative reward for the PPO-BASIC-FMT approach (blue line), and the PPO-MEM-FMT (red line). The solid line denotes a smoothed line of the actual observed cumulative reward at each training step, while the transparent line represents the raw collected reward in each step. The y-axis indicates the cumulative reward, while the x-axis denotes the current training step. Apparently, the approach that uses memory achieved a significantly higher reward at the end of the training. This indicates that it learned a better policy during the training and will likely outperform the BASIC algorithm in the experiments. Surprisingly, from the beginning on, the MEM algorithm converges faster than the BASIC. This disproves the hypothesis that the BASIC approach shows faster success in the beginning due to the simplified state. This can be explained by the fact that by adding memory, the algorithm has access to additional relevant information. Hence, providing memory is crucial for success and indicates that the MEM approach will outperform the other algorithms in the experiments.

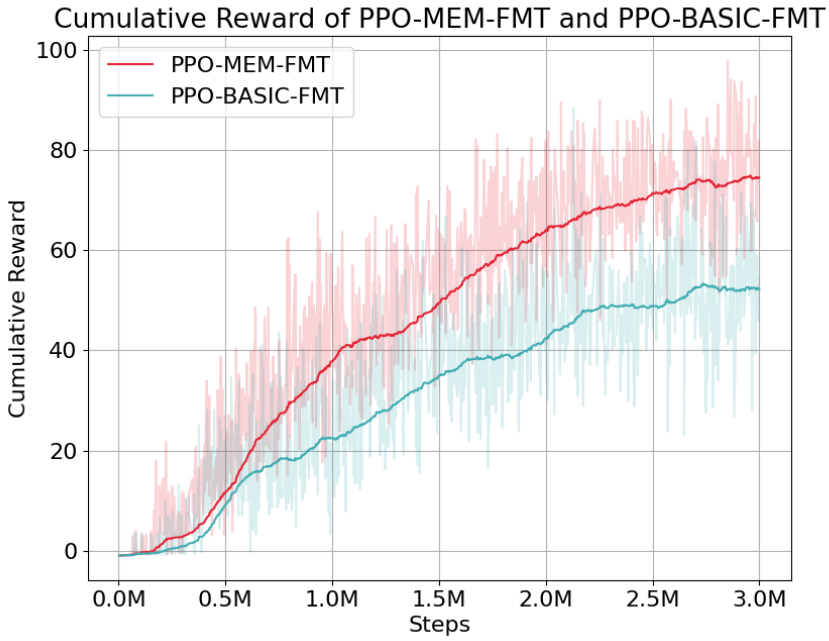


Figure 6.1.: Comparison of the development of the cumulative reward of the PPO-BASIC-FMT and PPO-MEM-FMT algorithms. Both algorithms are using the FMT training setup.

Furthermore, it is evident that the real collected reward has a high variance (transparent line). This results from the design of the reward function, where there is a high reward for completing the map, and in all failure cases, the reward is significantly less. Throughout the training, the agent always has cases in which it fails to complete the map. Thus, the reward jumps between the rewards. On average, the success rate increases over the training, and the agent more often completes the map. Hence, the collected cumulative reward increases throughout the training, and it can be drawn that the policy improves. Note that the slope of the curve is still positive at the end of the training. This indicates that the optimal policy has not been reached and that the agent will further make improvements as the training continues. As already explained, the training is terminated at 3 million steps because of the time limits in this thesis.

However, Figure 6.2 shows the development of the cumulative reward, but in this case, of the SGT approaches PPO-BASIC-SGT, PPO-MEM-SGT. From the Figure, it can be drawn that the curve initially is very steep. This indicates fast learning at the start. Subsequently, from around step 0.5 million, it flattens enormously and converges. This indicates that it found any optimum of the policy function. This optimum could either be a local one or a global one. Thus, probably, even if the algorithm is trained further, it might not make significant progress in enhancing the policy. Additionally, as in Figure 6.1, the attempt with memory outperforms the other and reaches a higher maximum cumulative reward.

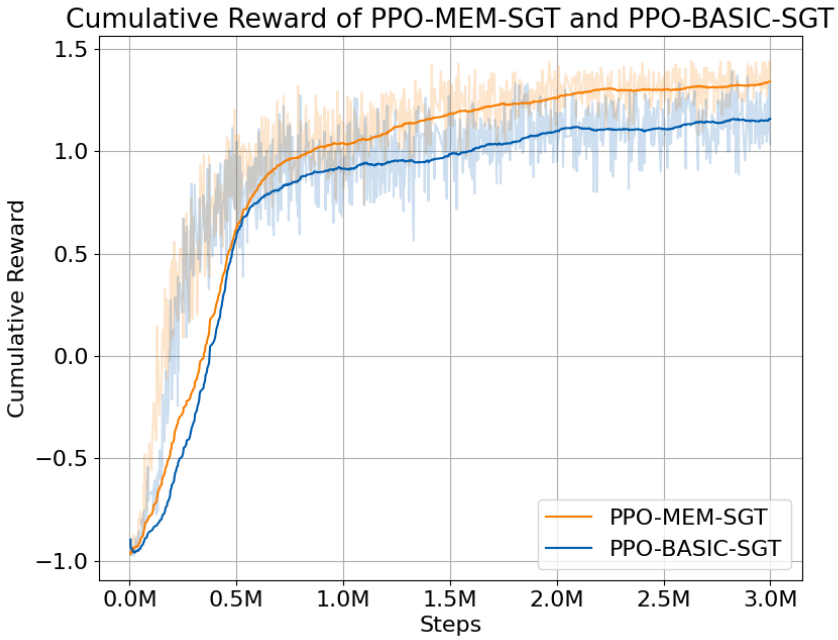


Figure 6.2.: Comparison of the development of the cumulative reward of the PPO-BASIC-SGT and PPO-MEM-SGT algorithms. Both algorithms are using the SGT training setup.

Furthermore, Figure 6.3 shows the development of the Normalized Cumulative Reward of all algorithms. In this case, the reward is normalized between 0 and 1 to be able to compare the course of the curve, even if the algorithms have different reward functions. It is apparent that the SGT approaches converge much faster at the beginning. Regarding Section 4.3.8, this is an expected behavior since the SGT training is heavily simplified and should show an accelerated training process. At around step 2.7 million, the FMT algorithms overtake the SGT attempts. By analyzing the course of the curve, it is furthermore apparent that the FMT algorithms show a more linear development, and it is very likely that the cumulative reward will still increase after step 3.0 million in opposite to the SGT approaches. As the values are normalized, reasonable predictions about the performance of the algorithms can not be drawn. Nevertheless, the Figure provides a good possibility to compare the development of the policy of the different approaches, showing that the SGT approaches are more efficient. The observed difference between the FMT and SGT approaches matches the expectation, and the hypothesis that the FMT approaches will outperform the SGT algorithms can be further supported.

Moreover, Figure 6.4 shows a bar plot where each bar represents the time that each approach necessitated for one training. It is evident that all algorithms require around 25 to 26 hours to complete the 3 million training steps. Although the algorithms need similar training times, the attempts using SGT need slightly less than the FMT approaches.

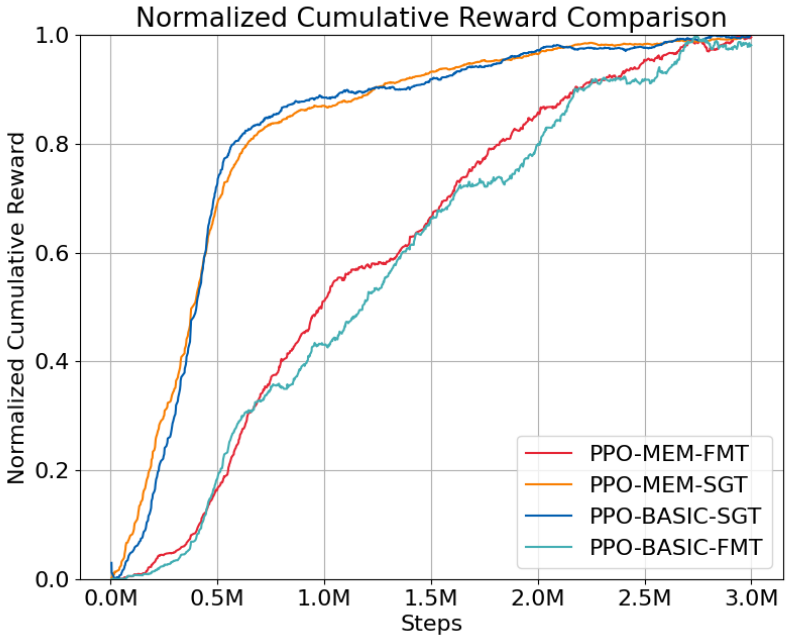


Figure 6.3.: Comparison of the normalized cumulative reward of all approaches during training

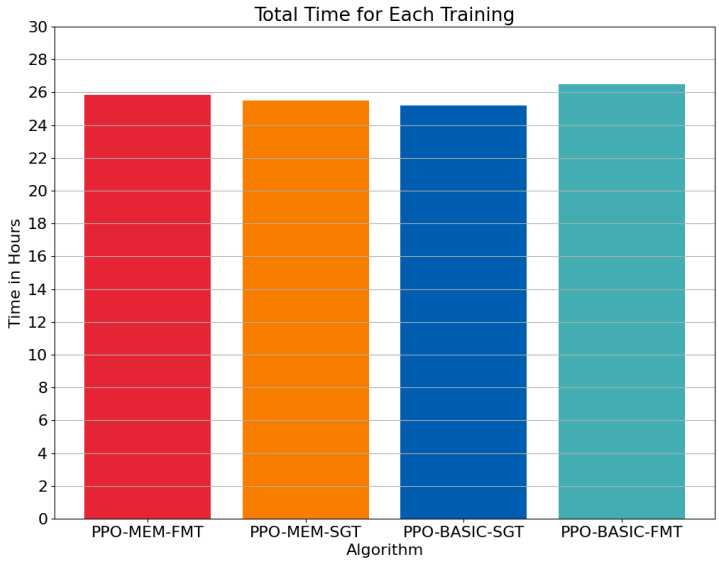


Figure 6.4.: Comparison of the total time that the different approaches need to train the full 3 million steps

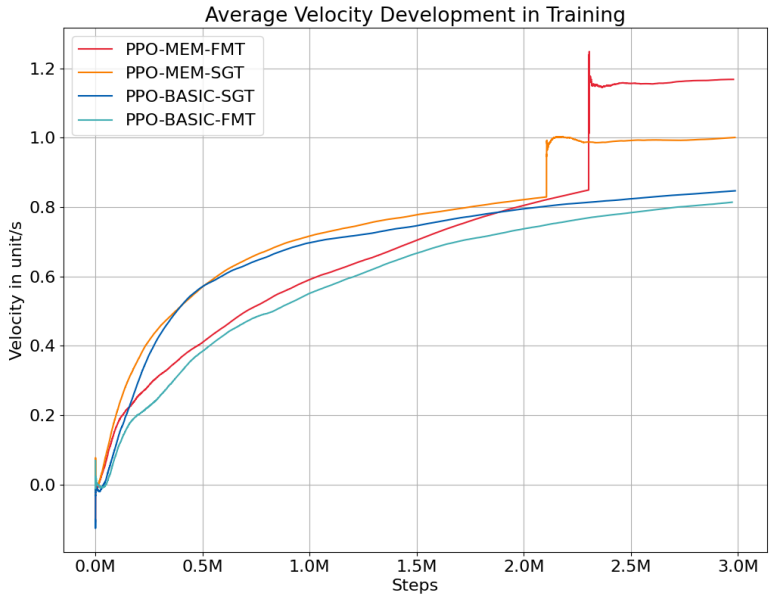


Figure 6.5.: Comparison of the development of the velocity of the different algorithms during the training

Figure 6.5 illustrates the curve progression of the velocity throughout the training. All approaches show a very steep increment of the velocity at the beginning of the training, which aligns with the observations that were made during the training. Initially, the agent does not accelerate or move forward directly. Instead, it explores different acceleration strategies. Since the velocity reward is provided without the requirement that the agent has to complete a sub-task, such as passing a goal, it learns fast that it needs to drive forward and accelerate in order to gain a positive reward. However, as the training progresses, the agent focuses more on passing a goal, as this exceeds the velocity reward. This explains the flatter slope at the end of the training. It is apparent that the initial part of the curve reminds of the development of the cumulative reward. This confirms the strong influence of the velocity on the gained reward in the initial stage of the training. The algorithms that utilize memory generally demonstrate the ability to drive at higher velocities.

6.3. Experiment performance

Finally, this Section provides the results of the experiments from Section 5.2 that lead to the answers to the research questions of this thesis. Therefore, Sections 6.3.1 to 6.3.3 present and evaluate the results of all experiments 1 to 3.

6.3.1. Experiment 1 - Performance in Optimal Conditions

As already mentioned in Section 5.2, in Experiment 1, all selected algorithms compete against each other on ten map configurations, comprising three different difficulties. In this case, the light setup is perfect ambient, and the objective is to evaluate the performance of the algorithms in terms of the success ratio and time, as introduced in Section 6.1.

Therefore, Figure 6.6 shows the success ratio of the four algorithms on all difficulties. Hence, the Figure shows the number of successfully completed maps on the y-axis and the particular map level on the x-axis. The success ratio of every difficulty is the averaged result across the different configurations of this level. For instance, the success ratio in the easy map is the mean of the result of the easy map starting with a blue goal and the easy map starting with a red goal.

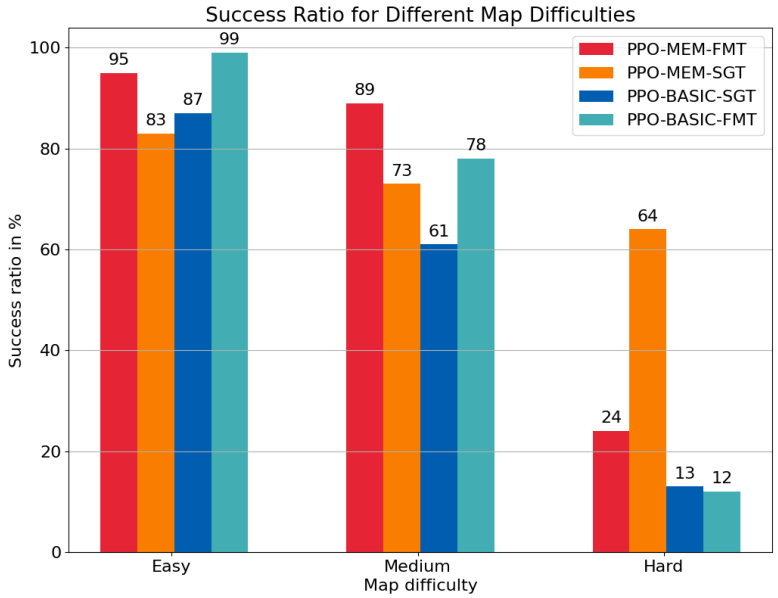


Figure 6.6.: Success ratio of experiment 1 across all algorithms on all map difficulties

By evaluating the Figure, it is evident that all four algorithms show strong results on the easy map. At this level, the FMT training methods outperform the SGT variants. The PPO-BASIC-FMT algorithm even completes 99% of the runs. This result indicates that all algorithms are able to understand the task that they have to drive forward through the goals until the end of the map. The results are slightly lower with the medium level, but the algorithms still demonstrate good performance, indicating their ability to steer and determine the next goal.

However, all approaches show poor performance when confronted with the hard difficulty level. Surprisingly, the PPO-MEM-SGT algorithm outperforms the others in this challenging setting.

This indicates that it benefits from simplified training, where its simplified objective is to find the next goal. This suggests that it benefits from its simplified training approach, focusing on finding the next goal. Given the limited training time, this is the only approach that gathers the capability of solving the most complex steer task and surpasses the depth of field problem.

Together with the limited training time, this approach can accomplish the complex steering task and solve the depth of field problem, as explained in Section 5.2.1.

To further understand the reason the SGT algorithms outperform the FMT approaches, Figure 6.7 examines the success ratio of the different algorithms on the hard track in more detail. Thus, the Figure shows the success ratio on the four configurations of the hard map. The four configurations are comprised of the first goal being blue and on the left side (hard blue left), blue and on the right side (hard blue right), red and on the left side (hard red left), and red and on the right side (hard red right). Figure 5.2 illustrates the four distinct hard map configurations. However, Figure 6.7 shows the performance of the SGT approaches PPO-BASIC-SGT (a) and PPO-MEM-SGT (b) on the top and the algorithms utilizing FMT, the PPO-BASIC-FMT (c) and the PPO-MEM-FMT (d) on the bottom.

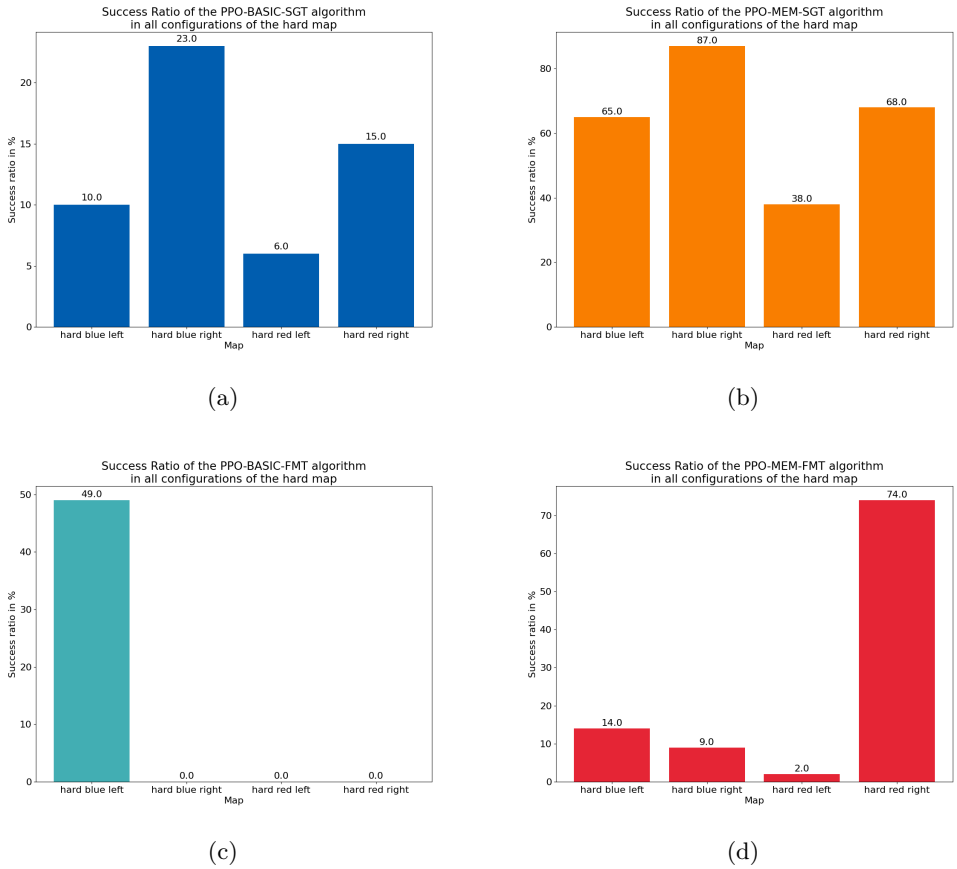


Figure 6.7.: Bar plot of the four algorithms (a) PPO-BASIC-SGT (b) PPO-MEM-SGT (c) PPO-BASIC-FMT and (d) PPO-MEM-FMT showing the distributed success ratio over all different configurations of the difficulty hard

It is evident that both SGT approaches have a similar distribution, as do both FMT attempts. Nevertheless, there is a difference between the two groups. In the SGT attempts, the success ratios are more evenly distributed across different map configurations, whereas the FMT algorithms only show strong results in one configuration while underperforming in the others. This aligns exactly with the expectations regarding the limited amount of training time.

Firstly, as the maps in the training are randomly generated, there are only a small amount of hard map configurations in the limited training time. Furthermore, at the beginning of the training, the algorithms are not capable of completing the hard configurations. Therefore, it is necessary to wait until the algorithms are strong enough to complete some of the difficult tracks and collect some positive experiences to learn how to solve the hard tasks.

Following the conclusion from Figure 6.3, that the SGT algorithms learn faster in training, these attempts are much sooner capable of training on hard tracks. Thus, they have more positive training cases on the difficult map, and the training cases where the agent successfully completed the tasks are overall more distributed across the different difficulties and map configurations. Therefore, they achieve a more generalized solution. This ends up in a more evenly distributed success ratio across the different map configurations, independently of the color or the position of the goals.

In opposite, the FMT approaches do not reach such a generalized solution. Again drawing the inferences from Figure 6.3, the FMT attempts to convert much slower towards a strong performance. Thus, they are only able to do the more difficult training cases in the final stage of the training, and, thus, they trained these variations much less than the SGT approaches. In the case of the, PPO-MEM-FMT, from Figure 6.7 (d), it can be seen that it was only able to complete the *hard red right* tracks. This indicates that in the final training stage, it solve more difficult cases where the first goal was red and on the left side, but too few difficult maps of other configurations. Thus, the algorithm was not able to achieve a general solution that can be applied to all of the map situations.

Nevertheless, the FMT approaches both have been successful at least at one of the hard track configurations, implying their general capability of solving the complex steering and depth of field problem. This underlines the hypothesis that by training them for a longer time, they would be additionally successful on the other hard map configurations and reach a more general successful solution, such as the SGT approaches.

However, Figure 6.8 compares the time that the algorithms need to complete a map for each difficulty. It is evident that the time only rises by a small amount from the easy map to the medium map. This can be explained as, in both cases, the agent does not have to steer a lot, and the variations are very similar. Nevertheless, the small increment indicates that the agent requires more time to navigate through the medium map. Assessing the time consumption on the hard map, the Figure illustrates that the algorithms necessitate more time to complete them. This is obvious, as for completing the hard map, the algorithm

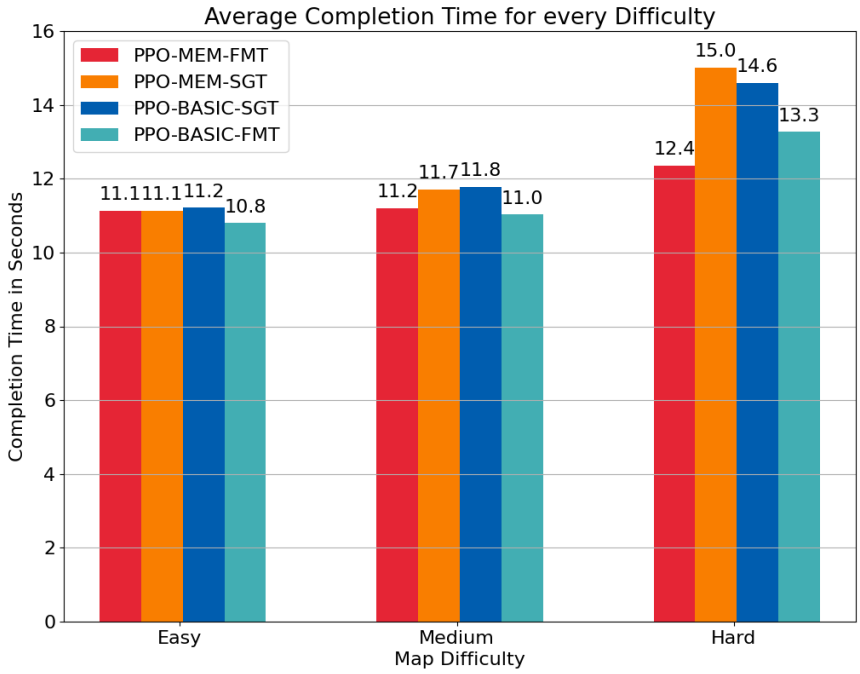


Figure 6.8.: Required time of experiment 1 across all algorithms on all map difficulties

needs to steer much more and, additionally, needs to cover more distance. Nevertheless, the results should be treated with caution due to the low amount of completed hard maps. This means, that there is a high variance in the results from the algorithms, except the PPO-MEM-SGT attempt.

6.3.2. Experiment 2 - Robustness against Different Light Settings

The hypothesis stated in Section 5.2.3, that changing the light setting by introducing directional light sources that produce shadows will cause major problems, can be confirmed. The algorithms PPO-BASIC-SGT, PPO-MEM-SGT, and PPO-MEM-FMT are not able to complete a single track in the changed light setting. Only the PPO-BASIC-FMT algorithm has finished 5 out of 100 tracks in the dark setting and 2 out of 100 in the bright light setting. This is statistically too low to be able to claim that the algorithm is more stable in lower light conditions. Figure 6.9 shows an example of the pre-processed image comparing the same situation in the perfect ambient, dark, and bright light settings. Therefore, (a) constitutes an example, where the map starts with a red goal and where the image is filtered by the red color. In opposite, (b) illustrates a scenario in which the map starts with a blue goal, and the blue color is filtered. It is evident that the recognition in both adjusted light settings works significantly worse than in the ambient light setting. In particular, the red filter in the dark light setting and the blue filter in the bright light setting totally failed to accomplish the task. Thus, the algorithm is fed with imprecise data. As the algorithm is not designed to deal with this kind of inaccurate input data, it

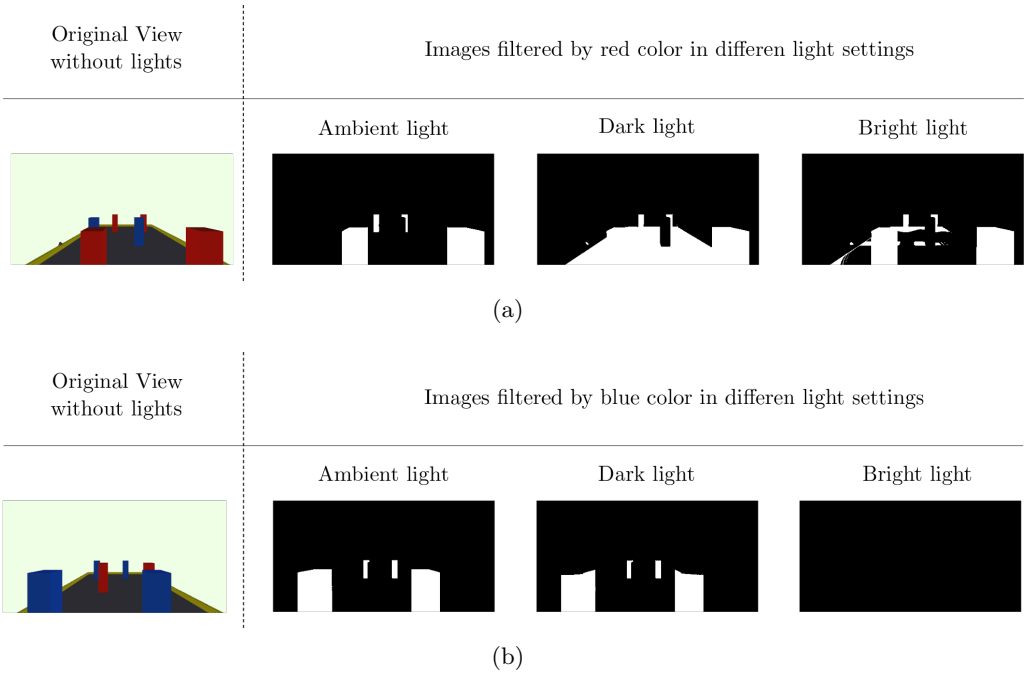


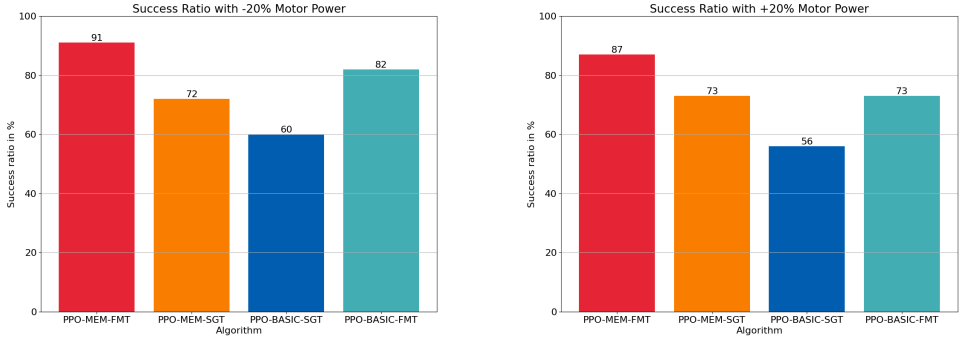
Figure 6.9.: Comparison of the processed image after filtering by the red color (a) and the blue color (b) in the different light settings

matches the expectations that the algorithm was not able to reach its objective. To ensure a better behavior in varying light settings, the image pre-processing should be improved in order to deliver accurate coordinates of the obstacles in worse light settings. This experiment further reveals the dependence of reinforcement learning algorithms on accurate input data.

6.3.3. Experiment 3 - Robustness against varying motor power

According to Section 5.2.4, Experiment 3 examines the robustness of the algorithms against varying motor engine power. Thus, all approaches have been evaluated, one time with 20% additional and one time with 20% less engine power. Figure 6.10 shows the success ratio of the algorithms in 100 runs on the medium map difficulty.

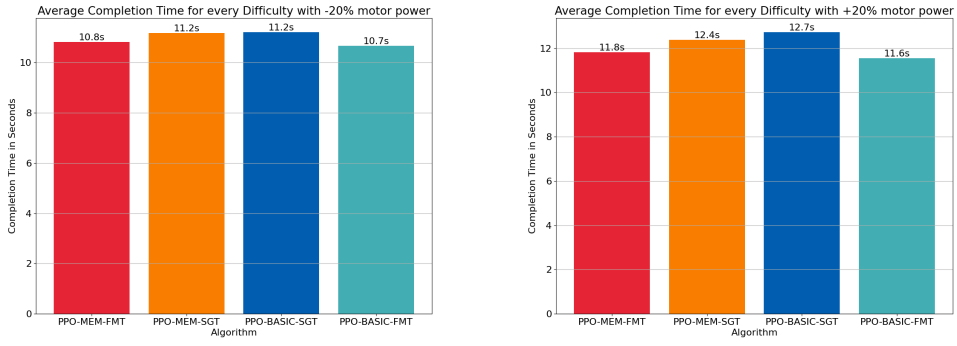
Consequently, Figure, 6.10 (a) shows the results of the experiments with +20% and (b) with −20%. It is evident, that the success ratio of each algorithm is similar to its success ratio in Experiment 1. This indicates that all approaches are very stable considering varying engine power in a range of −20% to +20%. This could be explained as the states in which the agent is, are similar. Thus, the agent can make the same predictions of accelerating the motors. The only difference is that with different motor power, the vehicle drives and turns a bit less in the −20% scenario and a little bit farther in the +20% scenario. This concludes in a behavior in which the agent with less motor power just needs to repeat the same action to reach the same following states, and with more motor power, it needs to reconsider its position and calculate a new path to the next goal.



(a) (b)

Figure 6.10.: Success ratio of the examined approaches, where (a) the motor engine power is lowered by 20% and in (b) increased by 20%

Figure 6.11 subsequently shows the time that the algorithms necessitated in both variations to complete the medium difficulty map. The performance of all approaches using less motor engine power equals the performance from Experiment 1. This can be explained as the agent driving the same path with less motor power by just executing the actions multiple times. Executing actions multiple times only makes an indistinguishable difference in the time consumption. It is noticeable that with +20% motor engine power, the algorithms need a small amount more time to complete the map. This underlines the explanation that, in this case, the agent drives a bit too far and needs to adjust its path. Thus, it needs to correct its route by steering and driving a larger distance. This ends up requiring more time to complete a track. Finally, Table 6.2 provides an overview of the results of all examined experiments on the medium map difficulty.



(a) (b)

Figure 6.11.: Bar plot that shows the average time that the examined algorithms required to complete a map, where (a) the motor engine power is lowered by 20% and in (b) increased by 20%

		Exp. 1	Exp. 2		Exp. 3	
		/	Dark	Bright	-20%	+20%
PPO-FMT-MEM	Success ratio in %	89	0	0	91	87
	Time in s	10.7	-	-	10.5	11.4
PPO-SGT-MEM	Success ratio in %	73	0	0	72	73
	Time in s	10.5	-	-	10.3	11.0
PPO-FMT-BASIC	Success ratio in %	61	0	0	60	56
	Time in s	10.5	-	-	10.4	10.9
PPO-SGT-BASIC	Success ratio in %	78	5	2	82	73
	Time in s	10.1	-	-	10.2	10.4

Table 6.2.: Overview of the results of all experiments on the medium map difficulty

7. Conclusion and Future Work

This ultimate Chapter of this master thesis summarizes the achieved results of the experiments and the consequent contribution of this work in Section 7.1. Finally, it provides an outlook of further investigations that yield promising outcomes in Section 7.2.

7.1. Conclusion

This master thesis aimed to investigate the use of reinforcement learning techniques for training an agent to drive a vehicle in a simulated environment by consuming visual camera input. Therefore, four research questions have been proposed at the beginning of this work. This Section summarizes the implemented methodology and concludes the results that have been achieved in order to answer the addressed questions. The research questions addressed were:

1. Is it possible to model the problem of training an autonomous driving agent in the context of reinforcement learning?
2. How can the camera input, which provides visual information about the environment, be processed and utilized to feed into the RL algorithm and achieve learning success?
3. How good is RL in solving the problem of training an agent to drive a vehicle in the simulated environment and passing all goals?
4. How robust are the developed RL algorithms against varying external influences, including light settings and motor engine power?

To answer the first question, a suitable action, two distinct state representations, and two particular reward functions have been successfully designed to model the task as a reinforcement learning problem. This ended in four algorithms that are capable of understanding the problem and that have been able to show a satisfying performance by navigating the agent through the track. Thus, the conclusion from the experiments is that it is possible to model the problem of training an autonomous driving agent in the context of reinforcement learning.

Secondly, the camera input, which provides visual information about the environment, was successfully processed by a straightforward approach. This attempt extracted the coordinates from the picture and utilized them to construct an RL state, which is then fed to the algorithm. The experiments proved that this procedure is highly efficient and provides encouraging results in the general task.

Furthermore, experiment 1 has been designed to examine the general performance of the designed RL algorithms that are consuming visual input and, thus, particularly addresses question three. The results have demonstrated promising results. All four algorithms showed great performances on easy and medium tracks, indicating the capability

to understand and solve the task. The PPO-MEM-SGT was even able to successfully complete the most difficult maps, showing that Reinforcement Learning is able to solve the most challenging tasks. Furthermore, the training was conducted on a low-powered computer within a relatively short time frame. This further underlines the effectiveness of the learning algorithm and implies that further training of the algorithms holds a high potential to improve their performances.

Lastly, experiments 2 and 3 examined the robustness of the algorithm against varying external influences, including light settings and motor engine power, and, thereby, answered question 3. The experiments yielded different results. It can be concluded that different external influences challenge particular aspects of the algorithm. All algorithms deliver weak performances in the changing light setting. The results showed that the light setting modifies the camera input and, thus, breaks the pre-processing of the image. Consequently, the algorithms have been fed with poor information and were not able to successfully approach their objective. In opposite, all approaches react affirmatively to the varying engine power and are able to adjust their behavior to these novel circumstances. To sum up, each change in the environment must be treated specifically. There are external influences on which the algorithm has to be further developed, such as the light setting, and on others, it already achieves great success, such as in the varying motor engine power.

In conclusion, the research presented in this thesis provides a promising alternative attempt at training an agent to drive through a simulated environment compared to the approach in the preceding thesis from [1]. Moreover, the challenges that occurred in the bachelor thesis of [2] have been addressed from the beginning of this work, especially by improving the image pre-processing and modeling the reality more accurately in the simulation. Thus, this work provides a solution that can be evaluated on the real existing robot in future works. Additionally, throughout the research, a modular framework has been developed that can be used to improve and reuse the virtual environment easily, train the algorithm for more time in different settings, and exchange the pre-processing or the learning algorithms. Thus, a valuable basis for future research in this domain has been set.

Overall, this thesis contributes to the field of autonomous driving and robots, approaching reliable AI-optimized driving systems that enhance road safety, reduce traffic, and minimize environmental pollution in the future. It yields valuable insights into the development of a driving agent that can maneuver through demanding tracks in varying external circumstances.

7.2. Future Work

Continuing the summary of the results, beyond the satisfying performance in most of the experiments, promising enhancements should be further investigated that exceed the limits of this thesis. This section introduces a collection of ideas that emerged during the research.

Firstly, the examined algorithms should be trained for more time. The results clearly show that the algorithms have the potential to increase their performance by training longer. Particularly, by examining the cumulative reward in the training of the FMT-approaches in Figure 6.1, it is obvious that the performance would rise with more training steps. This hypothesis is underlined by comparing the training time with the training times and used hardware in related experiments, e.g., as presented in chapter 2. For instance, the authors in [12] used a computer with 16 GPUs and 12228 CPU cores and trained their algorithms for up to 50 hours to achieve satisfying results. Certainly, the experiments differ, but this shows how little time was available for training in this thesis.

Furthermore, an idea could be to provide the agent with additional sensors or the input of multiple cameras. Thus, the agent would be able to gather a better understanding of its surroundings. It is very likely that this would lead to better results. This could also be a solution for the problem of image pre-processing in different light settings, as, with distance sensors, the agent would be less dependent on the pre-processing.

Moreover, the image pre-processing can be tremendously improved. Particularly, the results from experiment 2 in Section 6.3.2 show that the pre-processing is a crucial part, and, thus, it constitutes a bottleneck of the algorithms. A simple attempt could be to analyze the color thresholds and include the colors in the filter range that the light changes. Another promising attempt is to use deep learning approaches for object recognition and segmentation, e.g., by utilizing Convolutional Neural Networks [54, 55].

Another idea is to improve the memory of the algorithms. The experiments showed that memory additionally provides useful information to the algorithm. Furthermore, the algorithms that use memory show great potential despite the implemented memory being a simple attempt. The state-of-the-art approach of adding memory to reinforcement learning algorithms is to utilize Recurrent Neural Networks RNN, such as Long Short-Term Memory (LSTM). The authors [13, 17] presented in Section 2 utilized this idea in their approaches and achieved promising results.

Finally, an attempt to increase the stability of the algorithm against varying external influences is to include those influences in the training. Thus, the agent could learn how to deal with those variations from the beginning. Furthermore, more influences could be included in the training and the experiments. For instance, include varying friction, camera resolution, noise in the camera picture, more light settings, or different motor powers between the left and right motors.

Although the results of this thesis are satisfying, this work only constitutes fundamental research. Considering the potential of the examined algorithms and the provided development framework, this thesis should encourage further investigation in this domain.

Bibliography

[1] Jonas König. "Model training of a simulated self-driving vehicle using an evolution-based neural network approach". BA thesis. Leipzig, Germany: Universität Leipzig, 2022.

[2] Merlin Flach. "Methods to Cross the simulation-to-reality gap". BA thesis. Leipzig, Germany: Universität Leipzig, 2023.

[3] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. Second. The MIT Press, 2018. URL: http://incompleteideas.net/book/the-book-2nd.html.

[4] Volodymyr Mnih et al. *Playing Atari with Deep Reinforcement Learning*. 2013. arXiv: 1312.5602 [cs.LG].

[5] Peter Henderson et al. *Deep Reinforcement Learning that Matters*. 2019. arXiv: 1709.06560 [cs.LG].

[6] Timothy P. Lillicrap et al. *Continuous control with deep reinforcement learning*. 2019. arXiv: 1509.02971 [cs.LG].

[7] Yuhuai Wu et al. *Scalable trust-region method for deep reinforcement learning using Kronecker-factored approximation*. 2017. arXiv: 1708.05144 [cs.LG].

[8] Peter Henderson, Joshua Romoff, and Joelle Pineau. *Where Did My Optimum Go?: An Empirical Analysis of Gradient Descent Optimization in Policy Gradient Methods*. 2018. arXiv: 1810.02525 [cs.LG].

[9] Yurii Nesterov. "A method for unconstrained convex minimization problem with the rate of convergence $o(1/k^2)$". In: 1983.

[10] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. 2017. arXiv: 1412.6980 [cs.LG].

[11] A. Rupam Mahmood et al. *Benchmarking Reinforcement Learning Algorithms on Real-World Robots*. 2018. arXiv: 1809.07731 [cs.LG].

[12] OpenAI: Marcin Andrychowicz et al. "Learning dexterous in-hand manipulation". In: *The International Journal of Robotics Research* 39.1 (2020), pp. 3–20. DOI: 10.1177/0278364919887447. eprint: https://doi.org/10.1177/0278364919887447. URL: https://doi.org/10.1177/0278364919887447.

[13] Qi Zhang, Tao Du, and Changzheng Tian. *Self-driving scale car trained by Deep reinforcement learning*. 2019. arXiv: 1909.03467 [cs.LG].

[14] Yusef Savid et al. "Simulated Autonomous Driving Using Reinforcement Learning: A Comparative Study on Unity and ML-Agents Framework". In: *Information* 14.5 (2023). ISSN: 2078-2489. DOI: 10.3390/info14050290. URL: https://www.mdpi.com/2078-2489/14/5/290.

[15] Abu Jafar Md Muzahid et al. "Deep Reinforcement Learning-Based Driving Strategy for Avoidance of Chain Collisions and Its Safety Efficiency Analysis in Autonomous Vehicles". In: *IEEE Access* 10 (2022), pp. 43303–43319. DOI: 10.1109/ACCESS.2022.3167812.

[16] Andreas Folkers, Matthias Rick, and Christof Büskens. "Controlling an Autonomous Vehicle with Deep Reinforcement Learning". In: *2019 IEEE Intelligent Vehicles Symposium (IV)*. 2019, pp. 2025–2031. DOI: 10.1109/IVS.2019.8814124.

[17] Dianzhao Li and Ostap Okhrin. *A Platform-Agnostic Deep Reinforcement Learning Framework for Effective Sim2Real Transfer in Autonomous Driving*. 2023. arXiv: 2304.08235 [cs.LG].

[18] Tuomas Haarnoja et al. *Soft Actor-Critic Algorithms and Applications*. 2019. arXiv: 1812.05905 [cs.LG].

[19] Scott Fujimoto, Herke van Hoof, and David Meger. *Addressing Function Approximation Error in Actor-Critic Methods*. 2018. arXiv: 1802.09477 [cs.AI].

[20] Lingheng Meng, Rob Gorbet, and Dana Kulić. *Memory-based Deep Reinforcement Learning for POMDPs*. 2021. arXiv: 2102.12344 [cs.LG].

[21] Martijn Otterlo and Marco Wiering. "Reinforcement Learning and Markov Decision Processes". In: *Reinforcement Learning: State of the Art* (Jan. 2012), pp. 3–42. DOI: 10.1007/978-3-642-27645-3_1.

[22] Richard Bellman. "A Markovian Decision Process". In: *Indiana University Mathematics Journal* 6 (1957), pp. 679–684.

[23] Csaba Szepesvári. *Algorithms for Reinforcement Learning*. Vol. 4. Jan. 2010. DOI: 10.2200/S00268ED1V01Y201005AIM009.

[24] Maximilian Schaller. "Weak Supervision Strategies Using Reinforcement Learning". BA thesis. Dresden, Germany: TU Dresden, 2019.

[25] Xintian Han. "A mathematical introduction to reinforcement learning". In: *Semantic Scholar* (2018), pp. 1–4.

[26] John Schulman. "Optimizing Expectations: From Deep Reinforcement Learning to Stochastic Computation Graphs". PhD thesis. EECS Department, University of California, Berkeley, Dec. 2016. URL: http://www2.eecs.berkeley.edu/Pubs/TechRpts/2016/EECS-2016-217.html.

[27] Joshua Achiam. "Spinning Up in Deep Reinforcement Learning". In: (2018). Accessed: 2023-05-10. URL: https://spinningup.openai.com/en/latest/index.html.

[28] Yu Tianyang Zhang Hongming. "Taxonomy of Reinforcement Learning Algorithms". In: *Deep Reinforcement Learning: Fundamentals, Research and Applications*. Singapore: Springer Singapore, 2020, pp. 125–133. ISBN: 978-981-15-4095-0. URL: https://doi.org/10.1007/978-981-15-4095-0_3.

[29] Théophane Weber et al. *Imagination-Augmented Agents for Deep Reinforcement Learning*. 2018. arXiv: 1707.06203 [cs.LG].

[30] Tingwu Wang et al. *Benchmarking Model-Based Reinforcement Learning*. 2019. arXiv: 1907.02057 [cs.LG].

[31] David Silver et al. *Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm*. 2017. arXiv: 1712.01815 [cs.AI].

[32] Christopher JCH Watkins and Peter Dayan. *Q-learning*. Vol. 8. 1992, pp. 279–292.

[33] Dongbin Zhao et al. "Deep reinforcement learning with experience replay based on SARSA". In: *2016 IEEE Symposium Series on Computational Intelligence (SSCI)*. 2016, pp. 1–6. DOI: 10.1109/SSCI.2016.7849837.

[34] Marc Bellemare, Will Dabney, and Remi Munos. "A Distributional Perspective on Reinforcement Learning". In: (July 2017).

[35] J.N. Tsitsiklis and B. Van Roy. "An analysis of temporal-difference learning with function approximation". In: *IEEE Transactions on Automatic Control* 42.5 (1997), pp. 674–690. DOI: 10.1109/9.580874.

[36] Richard S Sutton et al. "Policy Gradient Methods for Reinforcement Learning with Function Approximation". In: *Advances in Neural Information Processing Systems*. Ed. by S. Solla, T. Leen, and K. Müller. Vol. 12. MIT Press, 1999. URL: https://proceedings.neurips.cc/paper_files/paper/1999/file/464d828b85b0bed98e80ade0a5c43b0Paper.pdf.

[37] Ofir Nachum et al. *Bridging the Gap Between Value and Policy Based Reinforcement Learning*. 2017. arXiv: 1702.08892 [cs.AI].

[38] Volodymyr Mnih et al. *Asynchronous Methods for Deep Reinforcement Learning*. 2016. arXiv: 1602.01783 [cs.LG].

[39] John Schulman et al. *Trust Region Policy Optimization*. 2017. arXiv: 1502.05477 [cs.LG].

[40] John Schulman et al. *Proximal Policy Optimization Algorithms*. 2017. arXiv: 1707.06347 [cs.LG].

[41] John Schulman et al. *High-Dimensional Continuous Control Using Generalized Advantage Estimation*. 2018. arXiv: 1506.02438 [cs.LG].

[42] Arthur Juliani et al. *Unity: A General Platform for Intelligent Agents*. 2020. arXiv: 1809.02627 [cs.LG].

[43] John K Haas. "A history of the unity game engine". In: (2014).

[44] Marat Urmanov, Madina Alimanova, and Askar Nurkey. "Training Unity Machine Learning Agents using reinforcement learning method". In: *2019 15th International Conference on Electronics, Computer and Computation (ICECCO)*. 2019, pp. 1–4. DOI: 10.1109/ICECCO48375.2019.9043194.

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

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

[47] Xiaodong wu, Min Xu, and Lei Wang. "Differential Speed Steering Control for Four-Wheel Independent Driving Electric Vehicle". In: May 2013, pp. 1–6. ISBN: 978-1-4673-5194-2. DOI: 10.1109/ISIE.2013.6563667.

[48] G. Bradski. "The OpenCV Library". In: *Dr. Dobb's Journal of Software Tools* (2000).

[49] Aisha Ajmal et al. "A Comparison of RGB and HSV Colour Spaces for Visual Attention Models". In: *2018 International Conference on Image and Vision Computing New Zealand (IVCNZ)*. 2018, pp. 1–6. DOI: 10.1109/IVCNZ.2018.8634752.

[50] J. D. Hunter. "Matplotlib: A 2D graphics environment". In: *Computing in Science & Engineering* 9.3 (2007), pp. 90–95. DOI: 10.1109/MCSE.2007.55.

[51] Michael L. Waskom. "seaborn: statistical data visualization". In: *Journal of Open Source Software* 6.60 (2021), p. 3021. DOI: 10.21105/joss.03021. URL: https://doi.org/10.21105/joss.03021.

[52] Wes McKinney et al. "Data structures for statistical computing in python". In: *Proceedings of the 9th Python in Science Conference*. Vol. 445. Austin, TX. 2010, pp. 51–56.

[53] Emilio Jorge et al. "Reinforcement learning in real-time geometry assurance". In: *Procedia CIRP* 72 (Jan. 2018), pp. 1073–1078. DOI: 10.1016/j.procir.2018.03.168.

[54] Daniel Maturana and Sebastian Scherer. "Voxnet: A 3d convolutional neural network for real-time object recognition". In: *2015 IEEE/RSJ international conference on intelligent robots and systems (IROS)*. IEEE. 2015, pp. 922–928.

[55] Paul Voigtlaender and Bastian Leibe. *Online Adaptation of Convolutional Neural Networks for Video Object Segmentation*. 2017. arXiv: 1706.09364 [cs.CV].

Declaration of Authorship

I do solemnly declare that I have written the presented research thesis:

„ Train an Agent to Drive a Vehicle in a Simulated Environment Using Reinforcement Learning“

by myself without undue help from a second person others and without using such tools other than that specified. Where I have used thoughts from external sources, directly or indirectly, published or unpublished, this is always clearly attributed. I am aware that infringement can also subsequently lead to the cancellation of the degree.

Leipzig, the 21.09.2023

_____

Maximilian Schaller