Project Thesis
PRO-064

Mechanik und Meerestechnik

Implementation of an interface for the application of reinforcement learning in autonomous driving

Implementation einer Schnittstelle für die Anwendung von bestärkendem Lernen im autonomen Fahren

by
Sean Maroofi

Supervisors: Prof. Dr.-Ing. R. Seifried
Chenran Li, M.Sc.

Hamburg University of Technology (TUHH)
Institute of Mechanics and Ocean Engineering
Prof. Dr.-Ing. R. Seifried
Prof. Dr.-Ing. M. Tomizuka

Hamburg, February 2023

TUHH

Contents

Acknowledgement

I would like to thank Professor Masayoshi Tomizuka and Dr. Wei Zhan for the opportunity to work on this research topic in the *MSC*-Lab. Special thanks to Chenran Li for direct support and extensive guidance throughout the progress of this work.

Chapter 1

Introduction

The theoretical origin of artificial intelligence (AI) approaches go back to the 1950s, but it has been in recent years that technological resources reached a level in which the realization of such algorithms is possible and comfortable to handle. AI applications require a large amount of computation power and highly optimized utilization of hardware components in order to be efficiently deployed. With the increase in popularity of studying AI in various research fields several commercial and open-source libraries have been developed which allow simple application and straightforward handling of AI-algorithms and simulation environments. Autonomous intelligent systems promise to solve tasks successfully and more reliably than humans because of their ability to learn and remember, which led to an increase of their study and application for autonomous driving. But the realization of a fully self-driving vehicle is challenging because of the systems complexity. Autonomous vehicles need to drive safely in dense urban scenarios observing their surroundings and make the correct decision in any situation. On highways an autonomous vehicle needs to be able to achieve safe line changing and accelerate and break if necessary. If an animal or human crosses the street, the vehicle needs to react in time and avoid collisions while not endangering passengers. The amount of tasks and requirements are countless and make autonomous driving a complex and difficult problem to solve.
One area of AI are machine learning (ML)-applications in which an intelligent system learns to perform specific tasks. Reinforcement learning (RL) is a subfield of ML in which an intelligent agent takes actions in a provided environment. Performed actions are rewarded and the agent's goal is to achieve the highest score by maximizing the received rewards. Applying RL to autonomous driving tasks is promising as the driving process can be modeled as a sequential decision problem with probabilistic transitions and feedback returned by the environment [Forbes02]. Several work has been reported in which vehicles successfully learned to navigate through urban scenery [SavariChoe21] or execute key driving maneu-

ver such as lane switching [NaveedQiaoDolan20].
To work on topics in the field of autonomous driving, the *Mechanical Systems Control Lab* of Professor Tomizuka developed a traffic simulation over years. It includes a compact dataset containing recorded real-life data of interactive urban driving scenarios from different countries [ZhanEtAl19]. This simulation has been used to study trajectory prediction of vehicles in complex traffic scenario with close contact of traffic attendees as in [JiaEtAl21] or [ZhanEtAl21]. Additionally, the simulation was equipped with an interface to offer simple and straight forward trajectory planning algorithms.
To extend research into the field of RL-applications in traffic scenery, the developed simulation environment can be utilized for applying RL-algorithms. For this purpose a RL-framework for comfortable and straight forward implementations of RL-algorithms is necessary.

1.1 State of the Art

A popular choice for decision making in autonomous driving is to apply non-learning model-based algorithms. Usually these require to design a driving policy manually as described in [PadenEtAl16]. The procedure usually starts with a routing planning phase in which a route through the road network of an urban environment is chosen. It is followed by a behavioral layer deciding on an appropriate driving task such as lane following, lane changing, parking, etc. Once a driving behavior is selected, a motion planning module determines a dynamically feasible trajectory that is taking obstacles into account and ensures a comfortable and safe vehicle motion for the passengers. Lastly, a control system generates actuator inputs to follow the planned path with a stable closed-loop feedback controller. However, designing a driving policy by hand can be disadvantageous as the model might have to be modified depending on the scenario and task in the environment [ChenYuanTomizuka19b]. Especially in highly interactive traffic scenarios model-based approaches require either defining motion heuristics or an accurate designed cost function to specify the planning and decision making, which can be challenging. Furthermore, model-based approaches require improvement and maintenance by human engineers, which can be expensive and time consuming. [ChenYuanTomizuka19a]
One approach in autonomous driving is the incorporation of an experienced human driver and learning the driving policy by imitating human behavior. This procedure is called imitation learning and is highly studied in autonomous driving tasks [CoutoAntonelo21], [BansalKrizhevskyOgale18]. In this supervised learning approach designing a policy-model or reward (cost) function is not required and instead only expert driving data is necessary [ChenYuanTomizuka19a]. Collecting driving expert data is not difficult but can become costly and time

consuming as a large amount of data from real-world is needed. Additionally, the expert data generally does not include dangerous situations, which increases a risk of lack of safety as the vehicle wouldn't deal with such cases. Furthermore, with a human expert driver as the supervision the vehicle will not be able to be superior to a human driver [ChenYuanTomizuka19b]. An example for imitation learning approach is behavior cloning (BC) which has been applied to autonomous driving in 1989 known as the ALVINN project [Pomerleau88].
Another alternative are model-free deep RL-approaches. Model-free methods do not estimate the dynamics of the environment and instead try to find the optimal policy, but make us of a reward function. Dangerous situations and busy urban scenarios can be simulated and trained with the goal of exceeding human-level performance. The application of deep RL to autonomous driving tasks gained in popularity as they promise high performance results, adaption and generalization. In [FehérEtAl19] a trajectory is planned by determining two trajectory points between a provided start state and end state. A spline is inserted using the four points and the vehicle learns to chose the optimal intermediate states. The authors in [NaveedQiaoDolan20] study lane changing and lane following behavior for an autonomous vehicle. Based on observations taken in the environment a high-level network chooses between the options of changing the lane or staying in the current lane. The observation consists of three historical environment states. Each state includes the ego-vehicles current velocity, current lane-ID, ratio of change in distance for safety purposes as well as the velocity, lane-ID and distance to the surrounding vehicles, consisting of the obstacle-vehicle in the same lane as well as the target-vehicles in the target-lane. Once a maneuver is chosen, a low-level controller plans a trajectory fitting to the current state. For example, for a lane changing action with a slower ego-velocity a more sharp trajectory is chosen while in a higher speed situation a longer and smoother trajectory is preferred. Once a trajectory is set, a proportional integral derivative (PID)-controller is used to follow the trajectory.

1.2 Goal of this Work

To use the simulation environment for learning-based trajectory planning an additional RL-framework is necessary. The goal is to provide an interface for simple implementation and testing of RL-algorithms in the traffic-simulation environment. The structure of this framework is designed to stick close to the common structure of other RL-libraries and therefore allow straight-forward adaptation and generalization. To test the framework, a basic model-free RL-approach is implemented, applied to a chosen traffic scenario and a vehicle is trained on the chosen environment in order to prepare a suitable basic reward design in the first

place, ensuring RL is applicable when applying the interface.
First, Chap. 2 explains the theoretical background of RL and a RL-approach is selected after a detailed discussion. Chap. 3 provides an overview of the structure of the simulation environment as well as details on how the RL-interface is integrated together with implementation details of the algorithm chosen previously. In Chap. 4 the framework is tested and the results are evaluated and details on the settings and the design are presented. Finally, Chap. 5 summarizes the present work and gives an outlook on further improvements.

Chapter 2

Reinforcement Learning

Because of the continuously growing popularity in applying and improving RL-algorithms the amount of available approaches is immense. All RL-approaches have unique properties and are suitable for different tasks and environments. Choosing an appropriate algorithm is not trivial, which is why in the following a handful of common implementations are introduced and compared and an approach is chosen after a detailed discussion eventually.

2.1 Fundamentals

A key part of AI is the notion of learning by trial and error. Instead of developing and implementing an optimized solution to a problem by hand, a self operating system will learn to solve the task itself. This approach is referred to as reinforcement learning. An agent takes actions in an environment and receives a corresponding reward returned by the environment. During the training process the agent learns to determine the optimal actions by maximizing the returned rewards. Fig. 2.1 shows a simple example of a fictional game which a RL-algorithm could try to learn. An agent tries to reach a goal field by moving around in a grid-based environment. Each time the agent takes an action, it receives a corresponding reward. For example, small negative rewards are returned for less meaningful actions, such as moving to a neighboring field, and a larger reward is returned for reaching a termination state which can be the desired goal or an unfavorable state, ending the current episode. By reiterating the given problem the agent will learn to find the optimal solution.
In the following the basics of RL are explained and essentials of the mathematical theory are established. The notation is held close to [AbbeelLevine22].

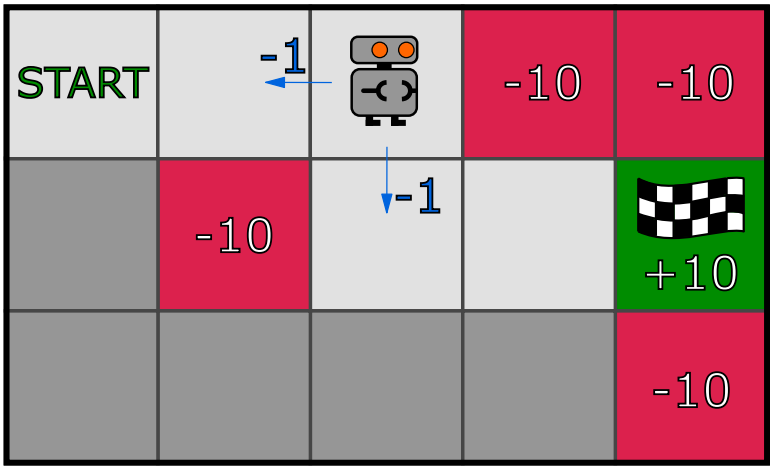


Figure 2.1: Example of a grid-based RL-problem. Unfavorable termination fields (red) return negative rewards and goal state(s) a positive (green). The agent learns to find the optimal solution (light gray)

2.1.1 Markov Decision Process

The agents current state $\boldsymbol{s}$ in the environment is represented by its state space $\mathcal{S}$. It can be discrete or continuous depending on the environment. Each state $\boldsymbol{s} \in \mathcal{S}$ contains all relevant information about the agent in the current environment step t. In some literature, as for example in [AbbeelLevine22], t is referred to the current time step as it describes the environment at time t. Because the later introduced simulation environment is stepped the terms environment step and time step are used interchangeably for describing t. The transition probability $p(\boldsymbol{s}_{t+1}|\boldsymbol{s}_t)$ between two states $\boldsymbol{s}$ is described by the transition operator $\mathcal{T}$. For a nondeterministic environment the state transition underlies uncertainty. It possesses the Markov property if the future state is conditionally independent from its predecessor $p(\boldsymbol{s}_{t+1}|\boldsymbol{s}_t, \boldsymbol{s}_{t-1}, ..., \boldsymbol{s}_0) = p(\boldsymbol{s}_{t+1}|\boldsymbol{s}_t)$. In other words, the stochastic process is memoryless and a sequence of states has the property of a Markov chain which is described by the two components $\mathcal{M} = \{\mathcal{S}, \mathcal{T}\}$. If the probability distribution over all the states at time step t is written as a vector $\boldsymbol{\mu}_t$ with entries $\mu_{t,i} = p(\boldsymbol{s}_t = i)$, then $\boldsymbol{\mathcal{T}}$ is a matrix with entries $\mathcal{T}_{i,j} = p(\boldsymbol{s}_{t+1} = i|\boldsymbol{s}_t = j)$ and the following linear equation holds

$$\boldsymbol{\mu}_{t+1} = \boldsymbol{\mathcal{T}}\boldsymbol{\mu}_t. \tag{2.1}$$

To specify a decision making problem the addition of action choices is necessary. This concept is referred to as a Markov decision process (MDP) [Bellman57]. An MDP consists of four elements $\mathcal{M} = \{\mathcal{S}, \mathcal{A}, \mathcal{T}, r\}$. In addition to the state space $\mathcal{S}$ and transition operator $\mathcal{T}$, the discrete or continuous action space $\mathcal{A}$ defines the possible actions $\boldsymbol{a} \in \mathcal{A}$ that an agent can take when being in state $\boldsymbol{s}$ while

$r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ describes the reward function. For each state transition at time t the agent receives a reward $r(\boldsymbol{s}_t, \boldsymbol{a}_t)$. The addition of decision making adds an additional probability. Again $\mu_{t,j} = p(\boldsymbol{s}_t = i)$ describes the probability of state j at time t. The probability of action k taken at time t is given by $\nu_{t,k} = p(\boldsymbol{a}_t = k)$. Hence the transition operator $\boldsymbol{\mathcal{T}}$ becomes a three dimensional tensor with entries $\mathcal{T}_{i,j,k} = p(\boldsymbol{s}_{t+1} = i | \boldsymbol{s}_t = j, \boldsymbol{a}_t = k)$, which changes Eq. (2.1) to

$$\mu_{t+1,i} = \sum_{j,k} \mathcal{T}_{i,j,k} \mu_{t,j} \nu_{t,k}. \tag{2.2}$$

2.1.2 Goal in Reinforcement Learning

The probability of taking action $\boldsymbol{a}$ in state $\boldsymbol{s}$ is represented by the policy $\pi_\phi(\boldsymbol{a}|\boldsymbol{s})$ with ϕ denoting the parameters of the policy. A sequence of states $\boldsymbol{s}$ and the actions $\boldsymbol{a}$ taken in each state $\boldsymbol{s}$ in a finite horizon problem for a fixed number of time steps $t = 1 \dots T$ can be defined as a trajectory $\tau = (\boldsymbol{s}_1, \boldsymbol{a}_1, ..., \boldsymbol{s}_T, \boldsymbol{a}_T)$. The probability distribution over a trajectory can be factorized into the probability $\pi_\phi(\boldsymbol{a}|\boldsymbol{s})$ of an action $\boldsymbol{a}$ taken in a state $\boldsymbol{s}$ and state transition probability $\mathcal{T}_{s_{t+1},s_t,a_t} = p(\boldsymbol{s}_{t+1}|\boldsymbol{s}_t, \boldsymbol{a}_t)$,

$$p_\phi(\tau) = p(\boldsymbol{s}_1) \prod_{t=1}^{T} \pi_\phi(\boldsymbol{a}_t|\boldsymbol{s}_t) p(\boldsymbol{s}_{t+1}|\boldsymbol{s}_t, \boldsymbol{a}_t). \tag{2.3}$$

This equation denotes the probability distribution for starting in $\boldsymbol{s}_1$ and following the trajectory over a time horizon $t = T$. With the probability distribution $p_\phi(\tau)$ the objective of RL can be defined as the expected value under the trajectory distribution

$$\phi^* = \arg\max_\phi \mathop{\mathbb{E}}_{\tau \sim p_\phi(\tau)} \left[\sum_t r(\boldsymbol{s}_t, \boldsymbol{a}_t) \right]. \tag{2.4}$$

During the training time the goal is to shape the parameters ϕ that define the policy in order to maximize the expected return of the sum of rewards under a trajectory τ.
The factorization in Eq. (2.3) can be interpreted as a Markov chain on an extended state defined as a tuple $(\boldsymbol{s}_t, \boldsymbol{a}_t)$ at time t. Then the Markov property is denoted as $p((\boldsymbol{s}_{t+1}, \boldsymbol{a}_{t+1})|(\boldsymbol{s}_t, \boldsymbol{a}_t)) = \pi_\phi(\boldsymbol{a}_{t+1}|\boldsymbol{s}_{t+1}) p(\boldsymbol{s}_{t+1}|\boldsymbol{s}_t, \boldsymbol{a}_t)$ for the augmented state $\boldsymbol{s}_t^{aug} = (\boldsymbol{s}_t, \boldsymbol{a}_t)$. The linearity of expectation allows to take out the summation operation of the expectation over the trajectory distribution and Eq. (2.4) can be rewritten as the expected return at time t, under the state-action marginal $\rho_\pi(\boldsymbol{s}_t, \boldsymbol{a}_t)$ summed over time

$$\phi^* = \arg\max_\phi \sum_{t=1}^{T} \mathop{\mathbb{E}}_{(\boldsymbol{s}_t, \boldsymbol{a}_t) \sim \rho_\pi(\boldsymbol{s}_t, \boldsymbol{a}_t)} \left[r(\boldsymbol{s}_t, \boldsymbol{a}_t) \right]. \tag{2.5}$$

Considering Eq. (2.5) for the case $T = \infty$, the state-action marginal $\rho_\pi(\boldsymbol{s}_t, \boldsymbol{a}_t)$ converges under the assumption of ergodicity (each state pair has a non-zero transition probability) and the Markov chain being aperiodic. Then Eq. (2.1) can be considered for k-times resulting in

$$\boldsymbol{\mu}_{t+k} = \begin{pmatrix} \boldsymbol{s}_{t+k} \\ \boldsymbol{a}_{t+k} \end{pmatrix} = \boldsymbol{\mathcal{T}}^k \begin{pmatrix} \boldsymbol{s}_t \\ \boldsymbol{a}_t \end{pmatrix}. \tag{2.6}$$

When $k \to \infty$, $\rho_\pi(\boldsymbol{s}_t, \boldsymbol{a}_t)$ converges to a stationary distribution and $\boldsymbol{\mu} = \boldsymbol{\mathcal{T}}\boldsymbol{\mu}$ yields. Then Eq. (2.6) becomes an eigenvalue problem

$$(\boldsymbol{I} - \boldsymbol{\mathcal{T}}^\infty)\boldsymbol{\mu} = 0, \tag{2.7}$$

with $\boldsymbol{\mu} = p$ being the eigenvector of $\boldsymbol{\mathcal{T}}^\infty$ with eigenvalue 1. To determine a stationary distribution, Eq. (2.7) has to be solved.

2.1.3 Value Functions

In order to design RL-algorithms, two important values are defined. The Q-value of a state action pair $(\boldsymbol{s}_t, \boldsymbol{a}_t)$ at time t describes the total reward that is achieved when starting in state $\boldsymbol{s}_t$ taking action $\boldsymbol{a}_t$ and following the policy afterwards. This is described by the following equation

$$Q^\pi(\boldsymbol{s}_t, \boldsymbol{a}_t) = \sum_{t'=t}^{T} \mathop{\mathbb{E}}_{\pi_\phi}[r(\boldsymbol{s}_{t'}, \boldsymbol{a}_{t'})|\boldsymbol{s}_t, \boldsymbol{a}_t], \tag{2.8}$$

where $\boldsymbol{s}_{t'}$ denotes the successive states of $\boldsymbol{s}_t$. Similarly the value function of a state $\boldsymbol{s}_t$ is the sum over all time steps $t \to T$ of the expected value of the reward at the successive states $\boldsymbol{s}_{t'}$ conditioned on the state $\boldsymbol{s}_t$,

$$V^\pi(\boldsymbol{s}) = \sum_{t^t=t}^{T} \mathop{\mathbb{E}}_{\pi_\phi}[r(\boldsymbol{s}_{t'}, \boldsymbol{a}_{t'})|\boldsymbol{s}_t]. \tag{2.9}$$

Both values can be used to find the optimal actions in two different ways. Possessing the current policy $\pi_\phi(\boldsymbol{a}|\boldsymbol{s})$ and knowing the Q-value $Q^\pi(\boldsymbol{s}, \boldsymbol{a})$ of a state action pair $(\boldsymbol{s}, \boldsymbol{a})$ allows to improve the policy. Choosing the best action $\boldsymbol{a}^* = \arg\max_{\boldsymbol{a}} Q^\pi(\boldsymbol{s}, \boldsymbol{a})$, an improved policy $\pi^{'}$ can be defined with $\pi^{'}_\phi(\boldsymbol{a}^*|\boldsymbol{s}) = 1$. Then the new policy $\pi^{'}$ is at least as good or better then π.
Alternatively, the probability of a good action can be increased by using gradient based updating. The definition for the value function in Eq. (2.9) can be rewritten as

$$V^\pi(\boldsymbol{s}) = \mathop{\mathbb{E}}_{\boldsymbol{a}_t \sim \pi_\phi(\boldsymbol{a}_t|\boldsymbol{s}_t)}[Q^\pi(\boldsymbol{s}_t, \boldsymbol{a}_t)]. \tag{2.10}$$

The value function is the expected value over actions of the Q-function and gives the average of the Q-values in $\boldsymbol{s}$ using the policy $\pi_\phi(\boldsymbol{a}|\boldsymbol{s})$. The current policy $\pi_\phi(\boldsymbol{a}|\boldsymbol{s})$ is modified in order to increase the probability of taking a good action $\boldsymbol{a}'$ for which $Q^\pi(\boldsymbol{s},\boldsymbol{a}) > V^\pi(\boldsymbol{s})$ applies, because then the single action $\boldsymbol{a}'$ is better than the average.
These two improvement methods are used for several different implementations of RL-algorithms.

2.2 Selection of an Algorithm

To test the RL-framework and design the reward values, an implementation of a RL-algorithm is necessary. In past works different algorithms with different settings have been applied, studying the various sub-problems arising in autonomous driving. The amount of different deep RL-algorithms is too large to cover, which is why instead a few common algorithms are presented and discussed in the following.

2.2.1 On-policy vs Off-policy

When it comes to model-free reinforcement learning there are two different types of reaching the optimal policy. In both methods a policy is applied which the agent follows, but they differ in the update procedure. On-policy methods follow a current policy to learn the state's values and update their policy after the agent reached a termination state. Afterwards, the updated policy is utilized to collect new information. In off-policy methods the policy is directly updated during learning, allowing sub-optimal actions to be taken and still converge to the optimal policy. Although on-policy methods improve stability in training, they often suffer from poor sample efficiency in continuous space. Off-policy algorithms make use of past experience, but require challenging addition of function approximators such as a neural network (NN) to approximate non linear functions for high-dimensional continuous spaces to ensure stability and convergence [HaarnojaEtAl18]. Further, on-policy variants learn directly with consecutive samples, which is inefficient due to correlations between the samples. In off-policy variants storing experience and randomizing samples from past experience can break correlations while at the same time the behavior distribution is averaged over the previous states when sampling from experience replay. This can help for smoother training and avoid converging to local minimum or even divergence in parameters of the function approximators [MnihEtAl13].

2.2.2 Deep Q-Learning

When talking about RL-algorithms, Q-Learning is a common popular off-policy applied variant. It estimates the Q-values $Q^{\pi}(\boldsymbol{s}_t, \boldsymbol{a}_t)$ and learns to find the optimal Q-value $Q^*(\boldsymbol{s}, \boldsymbol{a}) = \max Q^{\pi}(\boldsymbol{s}, \boldsymbol{a})$ together with the optimal policy $\pi^*(\boldsymbol{a}|\boldsymbol{s}) = \arg\max_{\boldsymbol{a}} Q^*(\boldsymbol{s}, \boldsymbol{a})$. For this purpose, Q-Learning makes use of the Bellman equations to update the Q-Values recursively until convergence and $Q^*(\boldsymbol{s}, \boldsymbol{a})$ is determined

$$Q^{\pi}(\boldsymbol{s}_t, \boldsymbol{a}_t) = \mathop{\mathbb{E}}_{\boldsymbol{s}_{t+1} \sim p}[r(\boldsymbol{s}_t, \boldsymbol{a}_t) + \gamma \max_{\boldsymbol{a}_{t+1}} Q^{\pi}(\boldsymbol{s}_{t+1}, \boldsymbol{a}_{t+1})]. \tag{2.11}$$

To extend the state $\boldsymbol{s}$ to a high dimensional continuous quantity, the Q-function can be approximated using a NN resulting in a parameterized Q-function $Q_{\theta}(\boldsymbol{s}_t, \boldsymbol{a}_t)$. This algorithms is referred to as deep Q network (DQN), [MnihEtAl13]. It makes use of a replay buffer $\mathcal{D}$ remembering past experience, and once a transition pair $(\boldsymbol{s}_t, \boldsymbol{a}_t, r(\boldsymbol{s}_t, \boldsymbol{a}_t), \boldsymbol{s}_{t+1})$ is present, it is stored in $\mathcal{D}$ while at the same time mini-batches are sampled at each time step. Additionally, DQN implements a target network $Q_{\bar{\theta}}(\boldsymbol{s}_t, \boldsymbol{a}_t)$ together with the online network $Q_{\theta}(\boldsymbol{s}_t, \boldsymbol{a}_t)$ [KiranEtAl20]. The goal is to minimize the loss function, being a supervised regression problem with

$$L(\theta) = \mathop{\mathbb{E}}_{\boldsymbol{s}_t, \boldsymbol{a}_t \sim \rho_{\pi}(\boldsymbol{s}_t, \boldsymbol{a}_t)}[(y_t - Q_{\theta}(\boldsymbol{s}_t, \boldsymbol{a}_t))^2] \tag{2.12}$$

and

$$y_t = \mathop{\mathbb{E}}_{\boldsymbol{s}_{t+1} \sim p}[r(\boldsymbol{s}_t, \boldsymbol{a}_t) + \gamma \max_{\boldsymbol{a}_{t+1}} Q_{\bar{\theta}}(\boldsymbol{s}_{t+1}, \boldsymbol{a}_{t+1})]. \tag{2.13}$$

Here y_t denotes the temporal difference target. It has been proven that DQN is capable of learning control policies in several Atari games as in [MnihEtAl15]. Unfortunately, DQN works with discrete action space only, which complicates the action selection for a physical control task with a continuous action space, as for autonomous driving. This is due to the fact that in DQN the maximizing action is desired, which would require an iterative optimization process at every time step in a continuous setting. This is very slow with large and unconstrained function approximators [LillicrapEtAl15]. One approach is to discretize the action space, as studied in [XuEtAl16]. In terms of steering a vehicle the range of the steering angle is close around the center and therefore a discretization might make sense. The challenge lies in finding the right amount of steps, to ensure stable and smooth control. Choosing too many steps may become computationally expensive while choosing a bigger step size may lead to jerky behavior [KiranEtAl20].

2.2.3 Deep Deterministic Policy Gradient

Because DQN is only applicable to problems with a deterministic action space, one approach is to consider a NN representing the policy $\pi_\phi(\boldsymbol{a}_t|\boldsymbol{s}_t)$. The policy is updated using deterministic policy gradient (DPG) which is an actor-critic approach [SilverEtAl14]. The combination of the actor-critic approach together with the ideas in DQN is called deep deterministic policy gradient (DDPG). It applies a parameterized actor-network representing the policy mapping states to actions deterministically while the critic is learned with the Bellman equations as in Q-Learning. The temporal difference target is given by

$$y_t = r(\boldsymbol{s}_t, \boldsymbol{a}_t) + \gamma Q_\theta(\boldsymbol{s}_{t+1}, \boldsymbol{a}_{t+1})]. \tag{2.14}$$

The actor-network is updated using the sampled policy gradient

$$\hat{\nabla}_\phi J_\pi(\phi) = \mathop{\mathbb{E}}_{\boldsymbol{s}_t \sim p}[\nabla_{\boldsymbol{a}} Q_\theta(\boldsymbol{s}_t, \pi_\phi(\boldsymbol{s}_t)) \nabla_\phi \pi_\phi(\boldsymbol{s}_t)]. \tag{2.15}$$

Similar to DQN, in DDPG target networks are implemented. These are a target $Q_{\bar{\theta}}(\boldsymbol{s}_t, \boldsymbol{a}_t)$ for the Q-network and another target $\pi_{\bar{\phi}}(\boldsymbol{s}_t)$ for the policy-network. The weights of both target networks are updated with an exponentially moving average

$$\bar{\theta} = \zeta\ \theta + (1 - \zeta)\bar{\theta}, \tag{2.16}$$

$$\bar{\phi} = \zeta\ \phi + (1 - \zeta)\bar{\phi}. \tag{2.17}$$

Continuous action spaces are difficult to explore. Off-policy algorithms allow to separate the exploration problem from the learning process, which is why the authors in [LillicrapEtAl15] implement an additional noised policy, used to increase exploration

$$\pi(\boldsymbol{s}_t) = \pi_\phi(\boldsymbol{s}_t) + \epsilon_t. \tag{2.18}$$

Although DDPG offers sample-efficient learning, the combination of the deterministic actor-network and the Q-function suffers from extreme brittleness and high sensitivity towards hyperparameters. This makes DDPG difficult to stabilize, which leads to poorer performance of tasks with a high-dimensional action space [HaarnojaEtAl18]. Additionally, DDPG is susceptible to small approximation errors in the network-functions, which can lead to overestimation bias over the update process. This occurs because the learned approximated Q-function estimates the value of a state to be greater than the true unknown state value. While updating the policy, minor overestimation errors may develop into more significant bias. The accumulating error may lead to an estimation of bad states possessing high value. This leads to suboptimal policy updates, which results in unsatisfactory actions learned by the suboptimal policy [FujimotoHoofMeger18].

2.2.4 Twin Delayed Deep Deterministic Policy Gradient

To counter the overestimation and suboptimal policies twin delayed deep deterministic policy gradient (TD3) is introduced. In [FujimotoHoofMeger18] the authors address the issues with DDPG by adding minor changes. First, an additional Q-function is added and both networks are trained simultaneously. The two Q-functions $Q_{\bar{\theta}_1}$ and $Q_{\bar{\theta}_2}$ are compared and the one returning a smaller value is included in the Bellman error loss functions

$$y_t = r(\boldsymbol{s}_t, \boldsymbol{a}_t) + \gamma \min_{i=1,2} Q_{\bar{\theta}_i}(\boldsymbol{s}_{t+1}, \pi_{\phi_1}(\boldsymbol{s}_{t+1})), \tag{2.19}$$

$$y_t = r(\boldsymbol{s}_t, \boldsymbol{a}_t) + \gamma \min_{i=1,2} Q_{\bar{\theta}_i}(\boldsymbol{s}_{t+1}, \pi_{\phi_1}(\boldsymbol{s}_{t+1})). \tag{2.20}$$

This approach is called Clipped Double Q-Learning. It counters the overestimation as the smaller Q-value of the two networks is used to regress towards the target update. Second, in TD3 the policy is updated less frequently than the Q-function approximators. This design allows minimization of the estimation error before updating the policy, which initiates higher policy updates as value estimates with lower variance are considered during the policy update [FujimotoHoofMeger18].
TD3 also uses target policy smoothing regularization. A deterministic policy, as in DDPG, can overfit to sharp peaks in the value estimate induced by the Q-function approximators. The policy exploits these peaks and returns suboptimal behavior. For this purpose, noise is added to the actions in practice resulting in similar actions having a similar value. In other words, the Q-function is smoothed out over similar actions. The noise is clipped to keep the target close to the original action [FujimotoHoofMeger18]

$$y_t = r(\boldsymbol{s}_t, \boldsymbol{a}_t) + \gamma Q_{\bar{\theta}}(\boldsymbol{s}_{t+1}, \pi_{\pi_\phi(\boldsymbol{a}|\boldsymbol{s})\bar{}param}(\boldsymbol{s}_{t+1}) + \epsilon), \tag{2.21}$$

$$\epsilon \sim \text{clip}(\mathcal{N}(0, \sigma), -c, c). \tag{2.22}$$

2.2.5 Soft Actor Critic

Soft actor critic (SAC) is an off-policy model-free implementation of deep RL applicable to continuous state and action space [HaarnojaEtAl18]. It combines an off-policy replay-buffer to use data of past experience together with an actor-critic formulation. Similar to TD3, SAC uses two networks to approximate two Q-functions and regresses to a single target with the shared target incorporating clipped double Q-Learning. But instead of using the target policy, as in TD3, the current policy is evaluated for the next state-action pairs in the target. Additionally, SAC defines a stochastic actor which makes the addition of clipped noise

for target policy smoothing redundant. Furthermore, SAC extends the main objective of maximizing the reward in RL-problems by incorporating a maximum entropy framework

$$J(\pi) = \sum_{t=0}^{T} \mathop{\mathbb{E}}_{(\boldsymbol{s}_t, \boldsymbol{a}_t) \sim \rho_\pi(\boldsymbol{s}_t, \boldsymbol{a}_t)} \left[r(\boldsymbol{s}_t, \boldsymbol{a}_t) + \alpha_{temp} \mathcal{H}(\pi(\cdot|\boldsymbol{s}_t)) \right], \tag{2.23}$$

with α_{temp} denoting the weight of the entropy term. This extension allows the agent to change its behavior to be more varying by not only maximizing expected reward but increase exploration in the environment at the same time.

2.2.6 Choosing an Algorithm

In this work, the goal is to chose a RL-algorithm and test it on the simulation in order to examine if RL is possible with the simulation and to find appropriate basic necessities for the framework, such as implementing necessary helper functions and shape rewards.
All of the algorithms mentioned above are applicable to continuous state space. A single or multiple NN's are implemented in all of these approaches to approximate the Q-function(s). However, in DQN the action selection is deterministic and therefore DQN is inapplicable to continuous action spaces. Even though the action space can be discretized, if desired, it is unfavorable to choose a discretization method for examining the framework because this would complicate the development unnecessarily.
DDPG performs well for example in control tasks in common environments, such as the *cartpole-swing-up* task or the *cheetah-locomotion* problem [BrockmanEtAl16]. It offers great sample efficiency but shows instability [DuanEtAl16]. Small approximation errors can lead to overestimation bias and poor policy updates. TD3 is proposed and addresses the problems with DDPG. It outperforms DDPG and other RL-algorithms adding the clipped Double Q-function trick. Around the same time SAC was proposed and showed stability and sample efficiency. Although TD3 was shown to be superior to an earlier version of SAC, showing stronger performance across common RL-problems, such as *HalfCheetah-v1* and the *Walker2d-v1* [BrockmanEtAl16], in [HaarnojaEtAl18], the authors mention the addition of another Q-function as in TD3. The second Q-function network shows improved results in their experiments and outperforms TD3 in common RL-training environments [HaarnojaEtAl18], as well as custom environments, such as for autonomous driving [ChenYuanTomizuka19b].
The choice for a fitting algorithm is difficult as there are a variety of implementations with custom adjustments. Successful improvements, such as additional networks and smoothing operations, are adapted and customized further in order to improve results. For this work SAC was chosen. Recent publications have

shown its strong performance in learning problems [JesusEtAl21], [DuanEtAl21]. It is stable and less sensitive to hyperparameters compared to other algorithms [HaarnojaEtAl18]. The RL-framework added to the simulation needs to be examined in its capability of training and the reward function needs to be tuned. Being less sensitive to hyperparameter tuning promises to lay the focus on shaping rewards in order to complete the traffic simulation RL-training framework.

2.3 Components of Soft Actor Critic

The SAC approach is characterized by three key components. These are an entropy maximization framework, to ensure exploration in the environment and stability of learning, an actor-critic network consisting of two NN's for the policy function and value function, and an off-policy replay buffer which offers the ability to use previously collected data for efficiency purposes.
Again $\boldsymbol{s}_t$ and $\boldsymbol{a}_t$ denote the state and action at the current time step t while $p : \mathcal{S} \times \mathcal{S} \times \mathcal{A} \rightarrow [0, \infty]$ is the unknown transition probability, representing the probability density of the state $\boldsymbol{s}_{t+1}$ at the next time step. The reward the agent receives is represented by $r : \mathcal{S} \times \mathcal{A} \rightarrow [r_{min}, r_{max}]$ on each state transition. Additionally, the probability distribution of a state-action tuple $(\boldsymbol{s}_t, \boldsymbol{a}_t)$ is represented by ρ_π under the policy $\pi(\boldsymbol{a}_t|\boldsymbol{s}_t)$.
As proposed by [HaarnojaEtAl18], two parameterized Q-functions are trained independently to optimize $J_Q(\theta_i)$ to improve the performance by countering potential positive bias in the improvement step. The option of using a replay buffer allows to train the value estimators and policy on off-policy data.

2.3.1 Maximum Entropy Augmentation

As described by Eq. (2.5) the objective in RL is to maximize the expected sum of rewards

$$\sum_t \mathop{\mathbb{E}}_{(\boldsymbol{s},\boldsymbol{a})\sim\rho_\pi} \left[r(\boldsymbol{s}_t, \boldsymbol{a}_t) \right]. \tag{2.24}$$

In SAC the reward is extended with the expected entropy of the policy over $\rho_\pi(\boldsymbol{s}_t)$. This means that the optimal policy does not only maximize the reward but also maximizes the entropy at each visited state,

$$\pi^*(\boldsymbol{a}|\boldsymbol{s}) = \arg\max_\pi \sum_t \mathop{\mathbb{E}}_{(\boldsymbol{s}_t,\boldsymbol{a}_t)\sim\rho_\pi(\boldsymbol{s}_t,\boldsymbol{a}_t)} \left[r(\boldsymbol{s}_t, \boldsymbol{a}_t) + \alpha_{temp} \mathcal{H}(\pi(\cdot|\boldsymbol{s}_t)) \right]. \tag{2.25}$$

To regulate the effect of the entropy $\mathcal{H}$ in relation to the reward and control the stochasticity of the optimal policy, a temperature parameter α_{temp} is introduced.

This adjustment of the objective motivates the agent to increase exploration while dropping unpromising paths and additionally distribute probability uniformly among actions with equal attractiveness.
In case of infinite horizon problems the sum of the expected rewards and entropies can become infinite. Therefore the objective of the maximum entropy framework can be extended with a discount factor to ensure a finite sum and therefore convergence of the algorithm. Furthermore, the addition of a discount factor will encourage the agent to reach the goal state faster as early rewards will be favored. Then the objective in Eq. (2.25) becomes

$$J(\pi) = \sum_{t=0}^{\infty} \mathop{\mathbb{E}}_{(\boldsymbol{s}_t, \boldsymbol{a}_t) \sim \rho_\pi} \left[\sum_{l=t}^{\infty} \gamma^{l-t} \mathop{\mathbb{E}}_{\boldsymbol{s}_l \sim p, \boldsymbol{a}_l \sim \pi} [r(\boldsymbol{s}_t, \boldsymbol{a}_t) + \alpha_{temp} \mathcal{H}(\pi(\cdot|\boldsymbol{s}_t))|\boldsymbol{s}_t, \boldsymbol{a}_t] \right] \quad (2.26)$$

2.3.2 Soft Policy Iteration

A common approach in RL-training is to derive actor-critic algorithms from simple policy iteration. It consists of two components, the policy evaluation step and the policy improvement step. In the case of SAC, first a formulation of soft policy iteration is necessary. Similar to policy iteration, soft policy iteration alternates between policy evaluation and policy improvement to maximize entropy.
In the evaluation step of policy iteration with unknown state transition the Q-value $Q^\pi(\boldsymbol{s}_t, \boldsymbol{a}_t)$ of a state is to be evaluated using a fixed policy π. Following the set policy, the Q-values of each state-action pair are updated with

$$Q^\pi(\boldsymbol{s}_t, \boldsymbol{a}_t) \leftarrow r(\boldsymbol{s}_t, \boldsymbol{a}_t) + \gamma \mathop{\mathbb{E}}_{\boldsymbol{s}_{t+1} \sim p, \boldsymbol{a}_{t+1} \sim \pi} \left[Q^\pi(\boldsymbol{s}_{t+1}, \boldsymbol{a}_{t+1}) \right]. \quad (2.27)$$

In the policy improvement step the old policy π is replaced by a new policy π' with

$$\pi'(\boldsymbol{a}_t|\boldsymbol{s}_t) = \begin{cases} 1 & \text{if } \boldsymbol{a}_t = \arg\max_{\boldsymbol{a}_t} Q^\pi(\boldsymbol{s}_t, \boldsymbol{a}_t) \\ 0 & \text{otherwise} \end{cases}. \quad (2.28)$$

In the evaluation step of policy iteration the policy π is determined by trying to maximize the reward only. In soft policy evaluation the policy is optimized in terms of the maximum entropy formulation. As described by Eq. (2.25), the reward is augmented with the entropy term and the Q-value of a state is computed by repeatedly applying a modified Bellman backup transition operator $\mathcal{T}^\pi$ as in Eq. (2.1) with

$$\mathcal{T}^\pi Q(\boldsymbol{s}_t, \boldsymbol{a}_t) = r(\boldsymbol{s}_t, \boldsymbol{a}_t) + \gamma \mathop{\mathbb{E}}_{\boldsymbol{s}_{t+1} \sim p} [V(\boldsymbol{s}_{t+1})] \quad (2.29)$$

and

$$V(\boldsymbol{s}_t) = \mathop{\mathbb{E}}_{\boldsymbol{a}_t \sim \pi} Q(\boldsymbol{s}_t, \boldsymbol{a}_t) - \alpha_{temp} \log \pi(\boldsymbol{a}_t|\boldsymbol{s}_t), \quad (2.30)$$

with the term $\alpha_{temp} \log \pi(\boldsymbol{a}_t|\boldsymbol{s}_t)$ accounting for the entropy $\mathcal{H}$.
In the policy improvement step of soft policy iteration the policy is updated

towards the exponential of the new Q-function that has been computed in the evaluation step. Additionally, the choice of an updated policy is restricted by a set of policies Π. This set can be a parameterized by a family of distributions for example. For this purpose, the updated policy is projected into the desired set of policies, making use of the Kullback-Leibler divergence which gives a statistical distance measurement between two probability distributions. An improved policy is chosen with the goal to minimize the divergence as described by

$$\pi_{new} = \arg\min_{\pi' \in \Pi} \mathrm{D}_{KL}\left(\pi'(\cdot|\boldsymbol{s}_t) \middle\| \frac{\exp(Q^{\pi_{old}}(\boldsymbol{s}_t, \cdot))}{Z^{\pi_{old}}(\boldsymbol{s}_t)}\right), \tag{2.31}$$

with the partition function $Z^{\pi_{old}}(\boldsymbol{s}_t)$ normalizing the distribution.

2.3.3 A Soft Variant of Actor Critic

For a continuous setting the Q-function and policy are estimated with function approximators instead of iteratively running the evaluation and improvement step until convergence. Both networks are optimized alternatively using stochastic gradient descent.
In [HaarnojaEtAl18] the authors define a state value function $V_\psi(\boldsymbol{s}_t)$, a soft Q-function $Q_\theta(\boldsymbol{s}_t, \boldsymbol{a}_t)$ and tractable policy $\pi_\phi(\boldsymbol{a}_t|\boldsymbol{s}_t)$. These functions are parameterized by ψ, θ and ϕ. Both $V_\psi(\boldsymbol{s}_t)$ and $Q_\theta(\boldsymbol{s}_t, \boldsymbol{a}_t)$ are modeled as NN's while $\pi_\phi(\boldsymbol{a}_t|\boldsymbol{s}_t)$ can be computed as a Gaussian with mean and covariance computed with a NN, as proposed by the authors.

Update Rules

For the soft value function $V_\psi(\boldsymbol{s}_t)$ the objective is to minimize the squared residual error

$$J_V(\psi) = \mathop{\mathbb{E}}_{\boldsymbol{s}_t \sim \mathcal{D}}\left[\frac{1}{2}\left(V_\psi(\boldsymbol{s}_t) - \mathop{\mathbb{E}}_{\boldsymbol{a}_t \sim \pi_\phi}\left[Q_\theta(\boldsymbol{s}_t, \boldsymbol{a}_t) - \alpha_{temp} \log \pi_\phi(\boldsymbol{a}_t|\boldsymbol{s}_t)\right]\right)^2\right], \tag{2.32}$$

with $\mathcal{D}$ denoting the distribution of previously state-action samples or, in other words, the replay buffer containing previous experience. Eq. (2.32) can be optimized with stochastic gradient descent

$$\hat{\nabla}_\psi J_V(\psi) = \nabla_\psi V_\psi(\boldsymbol{s}_t)\left(V_\psi(\boldsymbol{s}_t) - Q_\theta(\boldsymbol{s}_t, \boldsymbol{a}_t) + \alpha_{temp} \log \pi_\phi(\boldsymbol{a}_t|\boldsymbol{s}_t)\right). \tag{2.33}$$

In contrast to Eq. (2.32), the actions $\boldsymbol{a}_t$ are sampled using the current policy instead of the replay buffer. Eq. (2.33) uses the minimum of the two Q-functions.

To find the optimal parameters of the soft Q-function the objective is to minimize the soft Bellman residual error

$$J_Q(\theta) = \mathop{\mathbb{E}}_{(\boldsymbol{s}_t, \boldsymbol{a}_t) \sim \mathcal{D}} \left[\frac{1}{2} \left(Q_\theta(\boldsymbol{s}_t, \boldsymbol{a}_t) - \hat{Q}(\boldsymbol{s}_t, \boldsymbol{a}_t) \right)^2 \right], \tag{2.34}$$

with $\hat{Q}(\boldsymbol{s}_t, \boldsymbol{a}_t)$ being the expected soft Q-value for the state-action tuple $(\boldsymbol{s}_t, \boldsymbol{a}_t)$ which is given by

$$\hat{Q}(\boldsymbol{s}_t, \boldsymbol{a}_t) = r(\boldsymbol{s}_t, \boldsymbol{a}_t) + \gamma \mathop{\mathbb{E}}_{\boldsymbol{s}_{t+1} \sim p} \left[V_{\bar{\psi}}(\boldsymbol{s}_{t+1}) \right]. \tag{2.35}$$

Again stochastic gradient descent can be used to optimize Eq. (2.34)

$$\hat{\nabla}_\theta J_Q(\theta) = \nabla_\theta Q_\theta(\boldsymbol{s}_t, \boldsymbol{a}_t) \left(Q_\theta(\boldsymbol{s}_t, \boldsymbol{a}_t) - r(\boldsymbol{s}_t, \boldsymbol{a}_t) - \gamma V_{\bar{\psi}}(\boldsymbol{s}_{t+1}) \right). \tag{2.36}$$

In this update the additional target value network $V_{\bar{\psi}}(\boldsymbol{s}_{t+1})$ is used with parameters $\bar{\psi}$.
At last, the parameters ϕ for the policy are obtained by minimizing the objective

$$J_\pi(\phi) = \mathop{\mathbb{E}}_{\boldsymbol{s}_t \sim \mathcal{D}, \epsilon_t \sim \mathcal{N}} \left[\log \pi_\phi(f_\phi(\epsilon_t; \boldsymbol{s}_t) | \boldsymbol{s}_t) - Q_\theta(\boldsymbol{s}_t, f_\phi(\epsilon_t; \boldsymbol{s}_t)) \right]. \tag{2.37}$$

Here the policy is reparameterized as the action $\boldsymbol{a}_t$ is replaced by

$$\boldsymbol{a}_t = f_\phi(\epsilon_t; \boldsymbol{s}_t), \tag{2.38}$$

in which f_ϕ represents a function with parameters $\boldsymbol{s}_t$ that takes in a noise vector ϵ_t as input, sampled from a fixed distribution. The policy π_ϕ is now depending on f_ϕ implicitly. Eq. (2.37) is optimized by approximating the gradient with

$$\hat{\nabla}_\phi J_\pi(\phi) = \nabla_\phi \log \pi_\phi(\boldsymbol{a}_t | \boldsymbol{s}_t) + \left(\nabla_{\boldsymbol{a}_t} \pi_\phi(\boldsymbol{a}_t | \boldsymbol{s}_t) - \nabla_{\boldsymbol{a}_t} Q^\pi(\boldsymbol{s}_t, \boldsymbol{a}_t) \right) \nabla_\phi f_\phi(\epsilon_t; \boldsymbol{s}_t), \tag{2.39}$$

with $\boldsymbol{a}_t$ being evaluated at $f_\phi(\epsilon_t; \boldsymbol{s}_t)$. Again the minimum of the two Q-functions is used in Eq. (2.39).

2.3.4 Updating the Target Network

In order to improve stability, [HaarnojaEtAl18] proposes adding an additional target network to track the actual value function simultaneously, as already mentioned in Sect. 2.3.3. The weights of the network are updated by an exponentially moving average between the value and target-value network's weights including an additional smoothing factor ζ

$$\bar{\psi} = \zeta \, \psi + (1 - \zeta) \bar{\psi}. \tag{2.40}$$

2.3.5 Reparameterization

For the reparameterization trick in Eq. (2.38) a squashing function tanh is applied elementwise to sample from a bounded Gaussian distribution

$$f_\phi(\epsilon_t; \boldsymbol{s}_t) = \tanh(\mu_\phi(\boldsymbol{s}_t) + \sigma_\phi(\boldsymbol{s}_t) \odot \epsilon_t), \tag{2.41}$$

with $\mu_\phi(\boldsymbol{s}_t)$ and $\sigma_\phi(\boldsymbol{s}_t)$ denoting the mean and standard deviation of the actor network.

2.3.6 Likelihood of Bounded Actions

The log-likelihood is necessary for updating the policy network and is computed with a change of variables in the policy, which results in

$$\pi_\phi(\boldsymbol{a}_t|\boldsymbol{s}_t) = f_\phi(\mathbf{u}_t|\boldsymbol{s}_t)\left|\det\left(\frac{\mathrm{d}\boldsymbol{a}_t}{\mathrm{d}\mathbf{u}_t}\right)\right|^{-1}, \tag{2.42}$$

with $\mathbf{u}_t \in \mathbb{R}^D$ being a random variable with mean $\mu_\phi(\boldsymbol{s}_t)$, standard deviation $\sigma_\phi(\boldsymbol{s}_t)$ and corresponding probability density f_ϕ and D the action dimension. The Jacobian is a diagonal matrix

$$\frac{\mathrm{d}\boldsymbol{a}_t}{\mathrm{d}\mathbf{u}_t} = \mathrm{diag}(1 - \tanh^2(\mathbf{u}_t)) \tag{2.43}$$

and the log-likelihood of the bounded action is computed by summing along the diagonal of the Jacobian

$$\log \pi_\phi(\boldsymbol{a}_t|\boldsymbol{s}_t) = \log f_\phi(\mathbf{u}_t|\boldsymbol{s}_t) - \sum_{i=1}^{D}(1 - \tanh^2(u_t^i)), \tag{2.44}$$

with u_t^i being the i^{th} element of $\mathbf{u}_t$.

Chapter 3

Application to the Simulation

The main contribution of this work was the establishment of the RL-interface by creating necessary sub-modules, integrating helper functions and changing parts of the logical structure behind the simulation environment. In the following an overview of the simulation structure is provided together with implementation details.

3.1 The General Structure

The simulation offers the ability to study behavior-related research in autonomous driving. It makes use of the INTERACTION dataset [ZhanEtAl19] which contains a variety of traffic scenarios including motion of traffic attendees from different countries. This dataset can be used to study a variety of topics relevant for autonomous driving, such as behavior and motion prediction, BC and RL, decision-making and planning algorithm development and more.
The structure of the simulation consists of two main threads, a main simulation pipeline and an external predictor side as depicted in Fig. 3.1. The simulation loop manages the communication and timings, ensuring sub-processes, such as state updates of every vehicle, are finished before a new iteration starts. On the predictor side the user can choose an algorithm which predicts the trajectory of the ego-vehicle(s) for future time steps.

The user has the option to chose a traffic scenario, for example a roundabout in a certain country including culture typical driving behavior in that area, the duration of the scenario as well as the number of vehicles attending the scenario. If desirable, the user can customize the initial state and design for each vehicle with the option to provide a spawning position, orientation and velocity as well as the size of the vehicle to represent vehicles of different types.
A single or multiple vehicles are chosen as ego-vehicles. This term refers to the

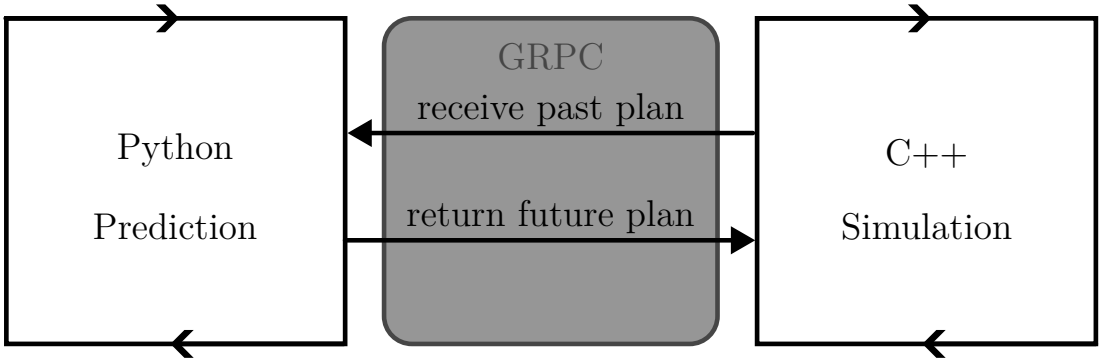


Figure 3.1: Server-Client connection: The prediction and simulation loop communicate with the Google remote procedure call (GRPC)-interface.

agent that is controlled in its environment. Each ego-vehicle receives the external prediction and is controlled by an internal default motion planning algorithm. Other vehicles are tracking the trajectories in the dataset, own responsive behavior to their surrounding vehicles and are driven by internal algorithms. For each surrounding vehicle the user decides if the ego-vehicle is aware of its current state and prediction or not. If the ego vehicle does not have knowledge about a vehicle, it is still incorporated by all the other vehicles in their prediction.
For the RL-setup a single ego-vehicle is chosen to receive a plan from the python side while the other vehicles continue to use internal classical planning algorithms on the C++ side. Exactly as in the prediction setup an urban scenario is chosen and the vehicles present during the setting together with their individual settings, as described in Sect. 3.1. Once the simulation is started, the RL-planner can be started. If the simulation reaches a termination state, the simulation is restarted and the initial state of the scenario is reloaded, Fig. 3.2. The RL-planner module is reset and information about the past episode is saved.

3.1.1 Server-Client Communication

The simulation pipeline and prediction loop are connected using the GRPC library. This extension allows the implementation of a two sided server-client communication offering the possibility of synchronous or asynchronous messages. To enable fast data transmission, GRPC makes use of a language and platform neutral binary data format called Protocol-buffer. Additionally, it allows communication to take place between a server and a client that both are implemented in different languages. For this use case the simulation side is written in C++ to offer fast computation performance. The client side uses python to offer the ability for fast and simple implementation of prediction and plan-

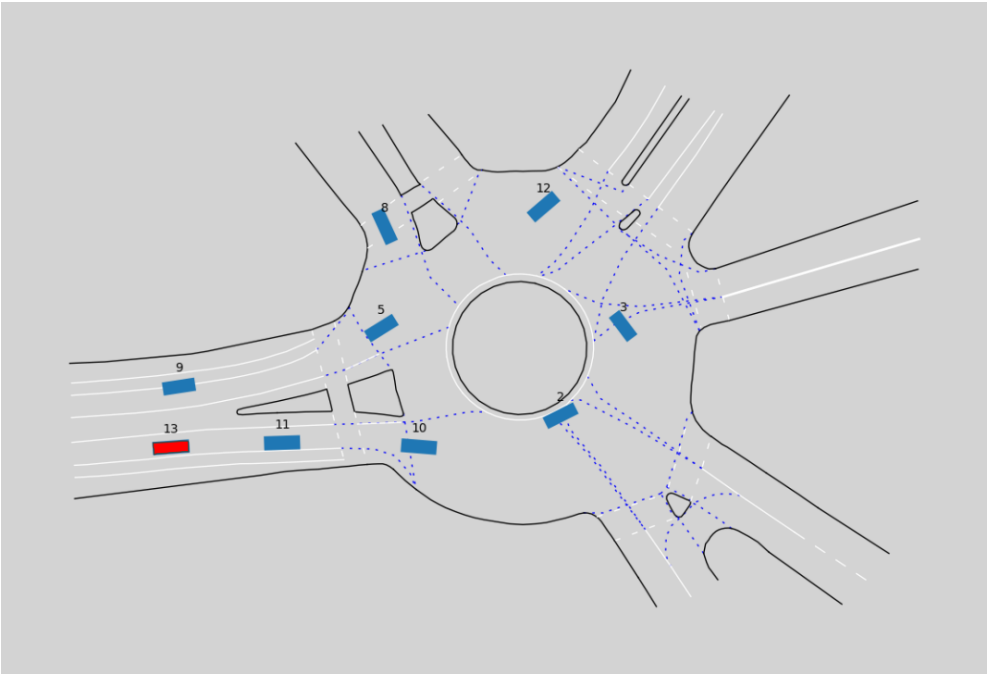


Figure 3.2: Simulation Environment: The initial state of a roundabout located in the United States is displayed. In this case vehicle number 13 is chosen as the agent/ego-vehicle.

ning algorithms and the usage of common PYTHON libraries such as PYTORCH [PaszkeEtAl19].

3.1.2 External Prediction and Planning

The original interface for the client side includes the option to develop a prediction algorithm of other vehicles future trajectories. For this purpose, the client receives historical trajectories, including its own individual trajectory as well as past states of other surrounding vehicles if desired. Each past trajectory contains information about the vehicles position, velocity and orientation from the last $n_{hist} = 10$ (1 s) time steps. The client predicts the trajectory of the ego vehicle for the next $n_{fut} = 30$ (3 s) time steps. It returns a set of predicted trajectories all equipped with a probability, denoting how likely this trajectory is to be taken by the vehicle.
In addition to the prediction interface, a client side planning interface was developed on an older state of the simulation. This interface is implemented to offer the possibility of simple and fast-forward PYTHON-based application of trajectory planning algorithms. Instead of studying trajectory prediction and using it for planning purposes directly, trajectory planning can be applied replacing

the internal planning algorithms on the simulation side. Prior to the addition of a RL-option, the planning interface needed to be transferred and adjusted to the newest version of the simulation. The user now has the choice to either run prediction or planning algorithms for the ego-vehicles.

3.2 Adding a Reinforcement Learning Framework

The main contribution of this work is to build a training loop for the updated planner interface. While previously there was two loops, the prediction/planner side and the simulation itself, for the RL-framework a third separate training loop was added which would only feature RL relevant information. The training loop is designed to stick close to the structure of the RL-pipeline implemented commonly, such as the OpenAI-gym library [BrockmanEtAl16] does, to ensure simple application of algorithms.
The RL-process is visualized in Fig. 3.3. The RL-framework consists of four main modules, the afore mentioned training loop, custom-agent-module, planning-module and connection-module. The training loop manages the learning procedure and the RL-module implements the custom chosen algorithm, in this work the SAC-algorithm. The planning-module is responsible for trajectory management and the connection-module communicates with the simulation which itself serves as the environment.
Each environment step the connection loop receives the past trajectory information of the simulation environment's state. The information is then transformed into an observation object suitable for the RL-algorithm which then chooses an action based on the observation. The action is further transformed by the planner module into a future plan and returned back to the simulation environment by the connection module. The connection loop returns the outcome, including the reward and current state of the simulation back to the training loop. Afterwards, the RL-module takes all the current step information, covering the observation, taken action, received reward and contiguous state observation, and saves it in the replay buffer. Finally the networks are trained according to their update rules described in Sect. 2.3.3.

3.2.1 Observation and Action Choice

In the previously applied implementations the prediction algorithms received the $n_{hist} = 10$ historical states to return a new planned trajectory consisting of $n_{fut} = 30$ states. Because the simulation is designed to receive $n_{fut} = 30$

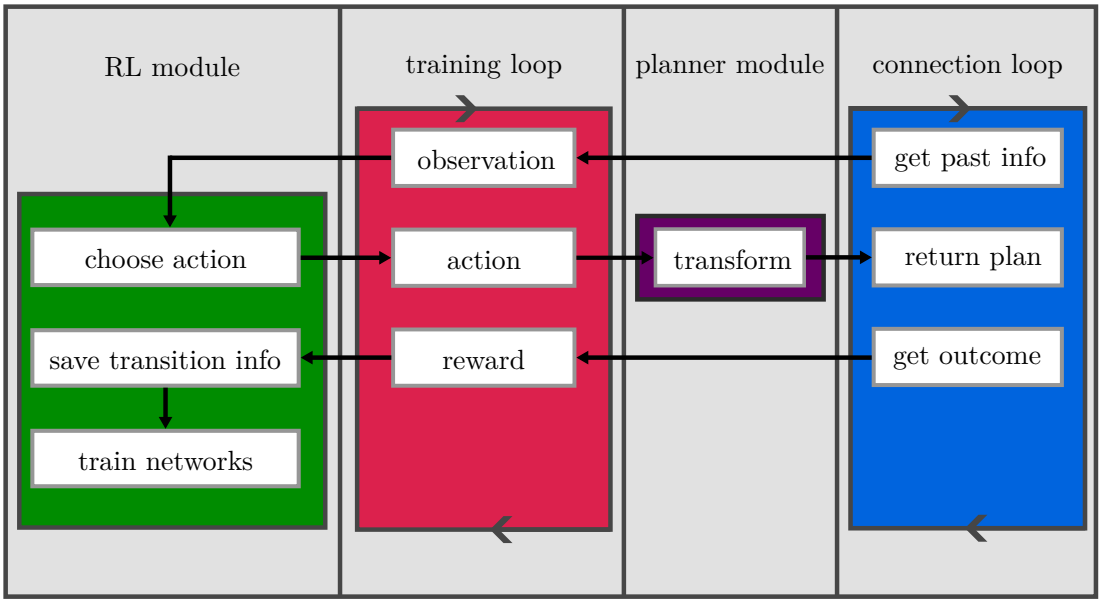


Figure 3.3: Visualization of the training loop on the PYTHON side.

states a similar choice is made for the RL-planner.
While in the prediction setting the input would include additional past trajectories of other vehicles the state $\boldsymbol{s}_t$ is reduced to only feature the ego vehicles individual information in the RL-planner setting. The state $\boldsymbol{s}_t$ of each vehicle at each environment step t consists of the x_t and y_t position together with the velocities in each direction $v_{x,t}$ and $v_{y,t}$ and the orientation of the vehicle ξ_t

$$\boldsymbol{s}_t = \begin{bmatrix} x_t \\ y_t \\ v_{x,t} \\ v_{y,t} \\ \xi_t \end{bmatrix} . \tag{3.1}$$

In contrast to previous prediction settings, not all of the $n_{hist} = 10$ trajectory points are used by the learning planner but rather only the current state at environment step t. The remaining state trajectory $t-1$ until $t-9$ is available in the implementation at environment step t but not considered by the RL-module, as those state's information is processed previously at environment steps $t-9$ until $t-1$ and therefore are redundant at environment step t. While in later applications other additional environment information is necessary to train the agent for the current setting, the ego vehicle's current state $\boldsymbol{s}_t^{ego}$ is sufficient because the RL-framework needs to be tested and evaluated as well as basic rewards have to be designed and tuned.
The output of the RL-module is a vector with the size of 1×2 consisting of values for the acceleration and orientation angle of the vehicle for the next environment

step

$$\boldsymbol{a}_t = \begin{bmatrix} a_{t+1} \\ \xi_{t+1} \end{bmatrix}, \tag{3.2}$$

with a_t denoting the acceleration of the vehicle at environment step t. Using these values a plan is generated by the planner module. The planned trajectory consists of the $n_{fut} = 30$ future states for the vehicles under the assumption of a constant velocity and orientation for these environment steps and the plan is returned to the simulation

$$\boldsymbol{s}_{t+i} = \begin{bmatrix} x_{t+i} \\ y_{t+i} \\ v_{x,t+i} \\ v_{y,t+i} \\ \xi_{t+i} \end{bmatrix} = \begin{bmatrix} \frac{1}{2} a_{t+1} \cos \xi_{t+1} \Delta t^2 + v_{x,t} \Delta t + x_t \\ \frac{1}{2} a_{t+1} \sin \xi_{t+1} \Delta t^2 + v_{y,t} \Delta t + y_t \\ a_{t+1} \cos \xi_{t+1} \Delta t + v_{x,t} \\ a_{t+1} \sin \xi_{t+1} \Delta t + v_{y,t} \\ \xi_{t+i} \end{bmatrix}, \tag{3.3}$$

with $i \in [1, 30]$. In the RL setting only the next environment step is evaluated and the remaining planned states are neglected. The remaining states can be used by other vehicles planning and prediction modules in order to plan their trajectory if desired. In this work the other vehicles do not incorporate the agents planned trajectory because early experiments showed unfavorable behavior learned by the agent. In a situation, where the agent is followed by another vehicle, the agent learned to decrease its current velocity, although it was supposed to drive at higher speed. This behavior did occur because the vehicle behind the agent reacted to the agents behavior and slowed down too, waiting for the agent to accelerate again. But because of a poor reward design during that training time, the agent would chose to not keep on driving.

In later applications the output can be changed to return steering values instead of direct orientation actions, as this is closer to the control of an actual vehicle. However, this requires only a conversion from steering action output to orientation information for the simulation, incorporating the vehicles dimension and is therefore omitted for this work.

In general two main approaches in vehicle control show popularity. The first implements direct control of the car with steering, breaking and acceleration actions. Alternatively the agent learns actions defined on a behavior level, such as lane changing, lane keeping , etc. These commands are passed on to a low level controller which returns the actual trajectory [Aradi20]. For example, [FehérEtAl19] makes use of the first approach defining an initial state and a desired end state with a DDPG implementation, returning two intermediate points for the trajectory. The end state is calculated with an empirical formula, ensuring to compute a feasible final state which can be reached by the vehicle under constant speed assumptions. The four points serve as basics for an inserted spline which is taking the gradients of the initial and end states into account. Another approach is a combination of direct control and behavior based action choices, as done

by [NageshraoTsengFilev19]. It separates longitudinal and lateral tasks. Longitudinal actions such as acceleration and braking are returned as direct control commands while lateral tasks contain *stay in lane* or *change in lane* commands. The steering command is then obtained using a feedback controller.

Chapter 4

Evaluation Of The Framework

In order to ensure the capability of learning with the implemented framework, basic SAC was implemented and applied to a chosen traffic-scenario. A key part of the framework is a suited reward function which provides information to the agent about its decision making. Finding an appropriate reward function can be challenging as different types of criteria need to be considered to rate an agents action. Valuing criteria have to be chosen and their values have to be tuned relatively in order to teach the agent its mistakes.

4.1 Parameter Settings

One main reason for the choice of SAC was its stability towards hyperparameter tuning. The majority of the parameters wouldn't effect the training process itself. The training settings are adopted from the original paper itself [HaarnojaEtAl18]. Tab. 4.1 shows the settings for the networks. The Adam-optimizer is chosen to update the networks weights and ReLU functions were used for non-linearity operations. All networks use the same values for their learning rates $\alpha_{act} = \alpha_{cri} = \alpha_{val}$. Two hidden layers where chosen for all networks, both consisting of 256 units per layer.
Tab. 4.2 lists all parameter settings for the SAC algorithm. Similar to the NN parameters, the values are chosen to match the paper [HaarnojaEtAl18]. A value of 0.99 was chosen for the discount factor. The replay buffer has space for storing $1 \cdot 10^6$ samples and each minibatch contains 256 samples. Each environment step is followed by a gradient step. The target smoothing coefficient needs to have a small value $\zeta = 0.005$ because the target weights should be updated slowly. The reward scale is a main tuning parameter in SAC as it represents the temperature coefficient of the energy-based policy and determines it stochasticity. A smaller reward scale results in a uniform policy generation and shows decrease in perfor-

mance as a consequence. A larger reward scale leads to an almost deterministic policy and results in less exploration. Its choice was found to be dependent to the dimension of the action selection [HaarnojaEtAl18]. For a larger action space a larger magnitude is necessary in general. Hence a value of $\alpha_{temp} = 1/5$ was chosen for the reward scale as the number of actions was $n_{act} = 2$.

4.2 Lacking Knowledge Of Traffic Rules

Early on during the tuning process it was noticeable that the agent's missing knowledge about traffic rules and behavior hampers the learning effect. Because no prior traffic rules were defined and no expert data is included during non supervised methods, the agent showed early on uncharacteristic behavior for a vehicle in a urban traffic scene. For example, in the beginning of the training the vehicle would learn to enter the roundabout but turn left instead of right for some cases, Fig. 4.1. After crashes with another vehicle or lane collisions, the vehicle would learn to turn right eventually as all choices to turn left result in penalizing rewards only. But without a supporting vehicle or the lane marking providing the correct direction in an intersection, the agent would need more time to figure out the right direction or, even worse, falsely understand driving counterclockwise to

Neural-Network-parameter	Value
Optimizer	Adam Optimizer
non-Linearity	ReLU
Actor network learning rate α_{act}	$3 \cdot 10^{-4}$
Critic network learning rate α_{cri}	$3 \cdot 10^{-4}$
Value network learning rate α_{val}	$3 \cdot 10^{-4}$
Number of hidden layers	2
Number of hidden units per layer	256

Table 4.1: Settings for the actor-, critic- and value-networks. If not explicitly mentioned the values are shared for all networks.

Soft Actor Critic parameter	Value
Discount factor γ	0.99
Replay buffer size	$1 \cdot 10^{6}$
Number of samples per minibatch	256
gradient steps	1
Target smoothing coefficient ζ	$5 \cdot 10^{-3}$
Reward scale	5

Table 4.2: Parameter settings for the SAC algorithm

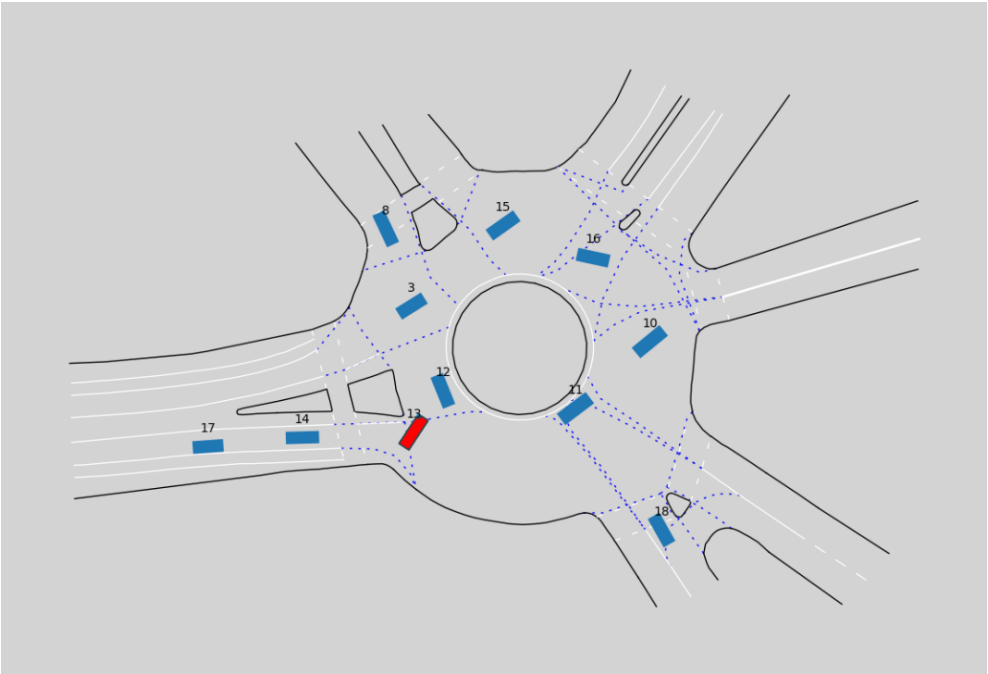


Figure 4.1: An example of the vehicle choosing to turn left in a round about and the upcoming collision.

be the correct choice. There are different approaches to counter this issue. One option is to penalize the agent with a negative reward for choosing to drive counterclockwise when entering a roundabout. However this would require to equip the roundabout with a corresponding defined reward. Moreover, once the agent learned to enter the roundabout it does not have information on when to leave the roundabout and could chose to stay in the roundabout for eternity. Additional information would be required for leaving the roundabout. Another option would be to implement a similar idea as done in [NaveedQiaoDolan20] where a high-level controller chose the agents behavior and a low-level system plans a trajectory. Similar for the roundabout, a high-level decision making network would detect the upcoming roundabout and chose to turn right with a low-level layer taking over the control of the vehicle. However, for this work a rather simple solution is chosen to test the framework and design reward values. Because the INTERACTION-dataset recorded the actual trajectory of each vehicle in real-time, the simulation is able to provide this information as a reference trajectory. Using this reference, a reward is designed punishing the agent the more it diverges from the actual trajectory. This counters the problem of potential counterclockwise directions and provides a reference on when to leave the roundabout again. Furthermore, the inclusion of a punishment for the deviation to the reference trajectory is adaptable and applicable to other other traffic scenarios, such as a merging scene.

4.3 Reward Shaping

Rewarding the actions, an agent takes in a custom environment, can be difficult to design. Complex dynamic environments as well as a continuous action space impede the reward shaping, resulting in intense study of reward choices, their tuning and chosen values. In terms of autonomous driving tasks, [PadenEtAl16] gives a short overview of typical applied reward strategies.

- Returning only rewards at the end of an episode.
- Returning rewards only for specific situations.
- Returning on-going immediate rewards at each environment step.

If the agent only receives a reward at the end of an episode, which is then discounted back to previously visited state-action pairs $(\boldsymbol{s}, \boldsymbol{a})$, longer learning times could result for the agent but reduce the human influence on the policy shaping. Immediate step rewards, evaluated at the current time step in the environment, reduce training time, but hamper the agents capability of finding an overall better solution to its task because of the intentionally designed reward. Last, an intermediate solution can be considered in which rewards are returned in predefined periods or for outstanding decisions chosen. It is up the the designer of the RL-approach to find a valuable mixture.

4.3.1 Termination States

Rewarding states in continuous environments can be challenging as there is no discrete state definition. In [FehérEtAl19] four different termination cases are considered. As the vehicle's goal is to follow the determined trajectory, the environment is reset once it's lateral distance exceeds 10 m, a lateral or longitudinal slip is higher than 0.1, the yaw angle error surpasses 0.2 rad or the maximum time limit is reached. If any of these cases occurs, the agent is punished with a negative reward and the environment is reset to its initial state. The termination cases for this simulation are listed in Tab. 4.3. Two termination cases are considered that are attached to a negative reward. The first one is the punishment the agent receives when the ego-vehicle passes over the road reference. The second reward is returned when the ego-vehicle crashes into another surrounding vehicle.
The three later termination cases were implemented as goal states. The first goal state returns a positive reward once the time limit is reached. Each traffic situation comes with a time constraint which in the previous prediction studies marked the end of the current scene. This time limit reward is included as

a reward in order to motivate the agent to stay alive. The second reward is introduced as an alternative to the previous time limit reward. It is returned once the agent is close enough to the final position of the reference trajectory. The purpose of this reward is to reward the agent for reaching the goal position in the environment instead of staying alive. It comes together with the third goal reward which penalizes the agent for taking the wrong exit or missing the goal position when taking the correct exit.

4.3.2 Situational Rewards

The task to be learned by the RL-algorithm decides whether situational rewards can be useful in the learning process. In an ongoing control task, without a specific goal state as in [ChenYuanTomizuka19b], ongoing rewards might have a better effect on the learning process than a spare reward only returned once the agents action lead to a specific outcome. A good example for sparse rewards is given in [FehérEtAl19] with the goal of trajectory planning. Once the agent determines a trajectory a fixed number of checkpoints is equally distributed along the trajectory. The agent receives a slip reward at each time step (an example for an ongoing reward) and additionally two rewards according to the agents driven distance and angle are calculated at the chosen checkpoints. This is an example of a combination of ongoing/step rewards and situational rewards.
In this work two situational rewards have been considered listed in Tab. 4.4. The first one is a total distance reward returned at the end of an episode. It is equivalent to the path the ego-vehicle drove in the environment, determined by calculating the sum of the distances between the single (x, y) positions of the driven trajectory. While this reward could count as a termination reward, it is separated from the termination rewards in Sect. 4.3.1 because the aforementioned rewards are afflicted with an individual termination case.
The second situational reward serves as an alternative for the "time limit reached" termination reward in Tab. 4.3. Instead of resetting the environment once the

Step reward	Reward
Lane Collision	- constant
Vehicle Collision	- constant
Time limit reached	+ constant
End reached	+ constant
Wrong exit/End missed	- constant

Table 4.3: A list of all the termination states together with their returned reward. All feature a constant reward that can be specified by the user.

time limit is reached, the agent receives a punishment for staying alive when the time limit is exceeded.

4.3.3 Ongoing Rewards

During the testing of the RL-framework, the on-going rewards were found to influence the performance of the agent significantly. In each environment step the agent benefits from receiving information about his last action. Simple tasks as the grid-based problem in Fig. 2.1 do not require a complex step reward. An example could be a simple punishing reward for staying alive, returned after each step taken in order to encourage the agent to move on to reach the goal state. But continuous environments together with a continuous action space require a more complex reward function.
Depending on the problem setting and the action choice, the reward design can vary in an autonomous driving task. In [ChenYuanTomizuka19b] the overall reward consists of five sub-rewards. The ego vehicle is encouraged to move forward by receiving a reward equivalent to its speed, but is reduced once a speed limit is exceeded $r_v \leftarrow 10 - r_v$ if $r_v = v_{vel} > 5\,\mathrm{m/s}$. In order to improve smooth driving steering is punished with $r_{\alpha_{steer}} = 0.5 \cdot \alpha^2_{steer}$. Vehicle collisions and lane exceeding are punished with $r_{col} = -10$ and $r_{exc} = -1$, respectively. Finally, a constant reward $r_c = -0.1$ is added to punish the ego-vehicle for standing still, similar to the simple grid-based example.
For this work several on-going rewards were implemented and evaluated. A list of all intermediate rewards are listed in Tab. 4.5. The left side shows a description of each action rewarded while the left side lists the formula to calculate the rewarded value. At each environment step the agent receives at least a constant negative reward for staying alive. This is equivalent to the reward r_c in [ChenYuanTomizuka19b] and the simple grid-based example. The second and third entry in Tab. 4.5 list negative rewards for a change in velocity and orientation. In some cases slowing down or changing the orientation is necessary, but these rewards main purpose is to counter non smooth trajectory development that emerges from drastic velocity and orientation jumps. A deviation to a reference velocity is introduced to keep the vehicle at a constant velocity. In a breaking process or accelerating situation the agent is forced to increase its velocity or decrease it, but adding this reward should stop the agent from accelerating to

Situation	Reward
Total traveled distance	$+\sum_{t=1}^{T} \sqrt{(x_t - x_{t-1})^2 + (y_t - y_{t-1})^2}$
Time limit exceeded	$-(t - T_{max}) \leftrightarrow t > T_{max}$

Table 4.4: Rewards returned only in chosen situations.

much and exceeding the constant target speed v^{ref}. To motivate the agent to move forward, each environment step a traveled distance reward is considered. It is calculated by measuring the distance between the position in the previous and the current environment step.
The last two rewards describe rewards determined when considering a reference trajectory for the ego-vehicle and comparing its current positioning in the environment to the reference. The reference trajectory contains the optimal x_t - y_t coordinates at each environment step t. Two different approaches are examined to include a reference reward. In earlier attempts the distance between the current position (x_t, y_t) of the ego-vehicle and the desired position (x_t^{ref}, y_t^{ref}) in the reference at environment step t was considered. The idea in this approach is to teach the agent to end up in the desired position at the according environment step t. As an alternative a slightly different approach is introduced to include the reference trajectory information. This reward is equal to the distance between the agents current position (x_t, y_t) and the nearest point (x_i^{ref}, y_i^{ref}) in the reference trajectory. With this design the agent is supposed to stick close to the reference trajectory in general, excluding the time constraint.

4.3.4 Reward Function Evaluation

Shaping the reward function for a RL-application can be a time consuming and challenging part. When exploring the action space RL-algorithms rely heavily on the reward function. For example, a sparse reward function can hamper the training process as RL-algorithms struggle to determine a successful policy. Because of this well thought out, hand-crafted reward functions are necessary to achieve satisfying training results, especially in the case of applications designed for real-world scenarios [RengarajanEtAl22].
While some of the introduced rewards look to be promising in theory their inclusion would hamper the training process. Looking at the goal termination rewards the "Time-limit-reached" reward showed unfavorable results when used

Action	Reward calculation
Staying alive	$-$ constant
Change in orientation	$-\lvert\xi_t - \xi_{t+1}\rvert$
Change in velocity	$-\lvert v_t - v_{t-1}\rvert$
Deviation to reference velocity	$-(v_t - v^{ref})^2$
Traveled step distance	$+\sqrt{(x_t - x_{t-1})^2 + (y_t - y_{t-1})^2}$
Deviation to reference trajectory	$-\sqrt{(x_t - x_t^{ref})^2 + (y_t - y_t^{ref})^2}$
Deviation to closest reference point	$-\min_i (x_t - x_i^{ref})^2 + (y_t - y_i^{ref})^2$

Table 4.5: Rewards returned at each environment step.

as the only goal reward. Instead of following the reference trajectory and trying to reach the goal in time, the agent would chose to slow down in the roundabout and find a collision-free spot to receive the positive reward as shown in Fig. 4.2.

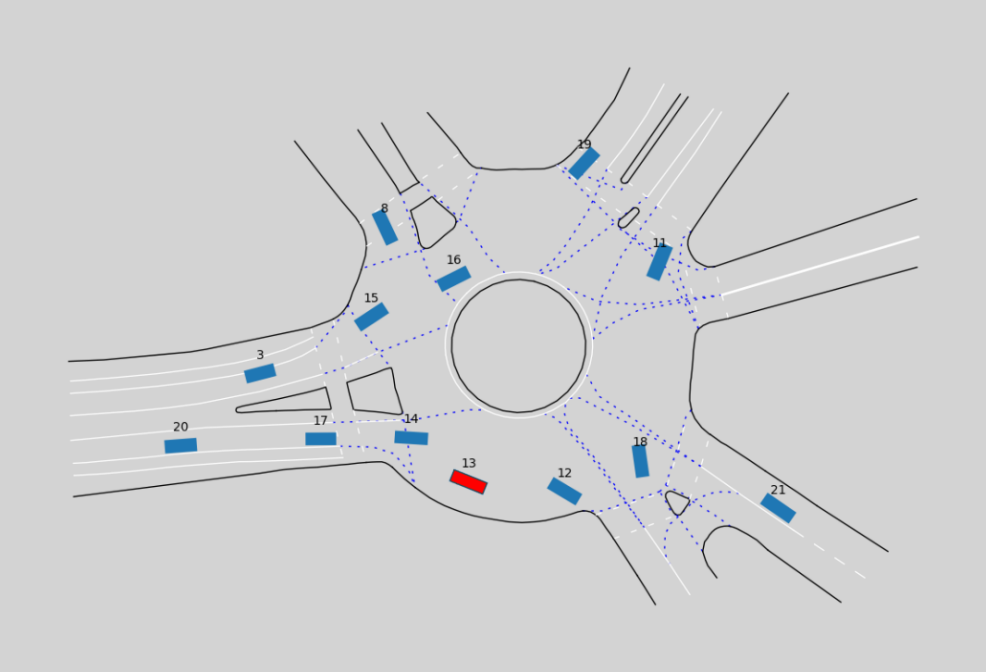


Figure 4.2: The vehicle waits at a collision-save location to receive the goal termination reward for surviving. Note that vehicle 11 was right in front of vehicle 13 at the beginning.

The alternative ”End-reached” termination reward improved the results a lot and the vehicle would be able to reach its final position. Additionally the ”Wrong-exit/End-missed” termination reward was added because in some cases the vehicle would miss the final position and despawn. Similar results could occur if the vehicle takes the wrong exit.
Two situational rewards were introduced. Both of them did not show significant influences when they were applied. Their removal did not influence the results in a noticeable way as the vehicle would show similar behaviour when the two rewards were not considered. The ”Total-traveled-distance” reward is only reasonable to add if a reference trajectory reward is omitted to motivate the agent to drive as far as possible. But it also comes with the risk of driving as long as possible instead of reaching the goal state if scaled wrong. The ”Time-limit-exceeded” reward is an alternative to the ”Time-limit-reached” termination reward and would counter staying alive as long as possible. But similar to the distance rewarding it could lead to a behavior where the agent would not explore all possibilities when applied wrongly if for example the agent would be punished before it could explore the goal state in a difficult traffic situation with a larger exploration phase. Their

influence therefore needs more examination, but both look promising for certain applications.
The ongoing rewards were the most influential rewards as they would shape the agents behavior each environment step. A "Staying-alive" reward did show least influence. Penalizing the agent for changing its velocity or orientation would lead to more smooth trajectories as seen in Fig. 4.5. The agent's trajectory becomes smoother in later time steps while it is still very curvy in the beginning. A direct reference velocity did not show favorable results and instead a range was defined which the velocity should not exceed. It is referred to as the reference velocity $v^{ref} = [v^{min}, v^{max}]$. Choosing a too large constant value for the reference velocity v^{ref} resulted in behavior in which the agent would drive to fast and crash into the vehicle in front, as shown in Fig. 3.2. Instead of slowing down earlier the vehicle would get very close the vehicle 11 and try to pass it once there is enough space. Reducing the constant value v^{ref} would help the ego-vehicle to not crash into the vehicle in front but drive too slow potentially slowing down traffic and letting other vehicles crash into it. With a defined velocity range and the penalization of large changes in the action space the agent is free to slow down and accelerate both to chosen limits, resulting in a more smooth velocity change over the trajectory. The ego-vehicle would learn to break early instead of abruptly slowing down while also maintaining to never stop and accelerate once it has the possibility to do so. When using the velocity range as a reward the results showed how the agent would almost keep a constant distance to the vehicle in front while previously it attempted to close the gap to keep the desired velocity. Similar to the "Total-traveled-distance" reward, the "Traveled-step-distance" reward did not show any big influences and again more time is necessary to investigate if it results in any additional benefits.
The largest influence on successful results together with the termination rewards was the inclusion of a reward connected to the reference trajectory. While in theory the design of the first reference-trajectory-reward makes sense it, comes with a crucial side effect. If the agent does not reach the desired position at the environment step t, it is unlikely to be at the target position at the next environment step $t + 1$. This means a single mistake at environment step t will influence all the upcoming environment steps following after and the negative reward will begin to increase. The vehicle is punished for the mistake it made at the environment step t every following step over and over again. This design will end up in a cumulative reward, which can hamper the training process as the agent will have difficulties to understand which decision was disadvantageous. The second design does not produce a cumulative reward as the time constraint is eliminated and therefore it is a better choice for the training process. However, during the training process it was noticeable that the agent would learn to slow down and stick to the minimum reference velocity v^{min} to receive the positive reward for consecutive environment steps and stick close to the closest reference point as long as possible. To counter this behavior, a counter mechanism was

implemented which would return a positive reward only once for each reference point. Once the agent was close enough to a reference point, it would have to continue driving to earn the next positive reward of a reference point as visualized in Fig. 4.3.

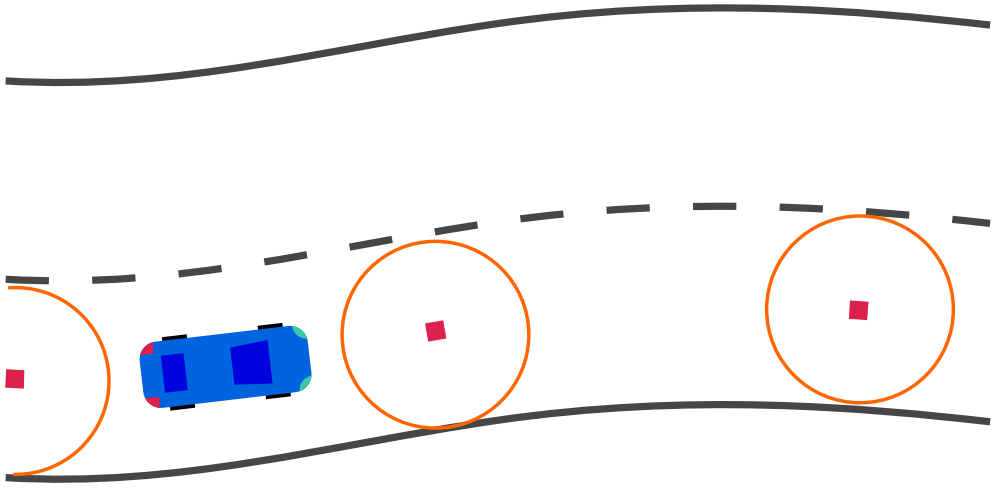


Figure 4.3: Visualization of the reference reward. If the agent is within a defined radius (orange circle) around a reference point (red square), it would receive a positive reward.

During the training process different implementations of the rewards were tested. The ongoing and situational rewards were first applied as dynamic rewards only with the reward value being equivalent to the calculated values. For example the reward for a change in velocity would be the result of $-|\xi_t - \xi_{t+1}|$. The agent would receive a larger punishment the more it would accelerate. However, this would lead to situations in which the agent explores dead ends of a policy as the rewards will always be slightly different causing the agent to get stuck searching for minor reward increases. In Fig. 4.4 a situation is shown where the vehicle would learn to crash into the line instead of continue driving as the vehicle 11 would be in front of it when approaching the roundabout entry, Fig. 3.2. Instead of following the vehicle with a slower velocity, the agent would choose to crash and maximize its reward beforehand as there were small changes in the rewards each environment step. Therefore some rewards were chosen to be constant with potential defined ranges, which showed improvements in the training results.

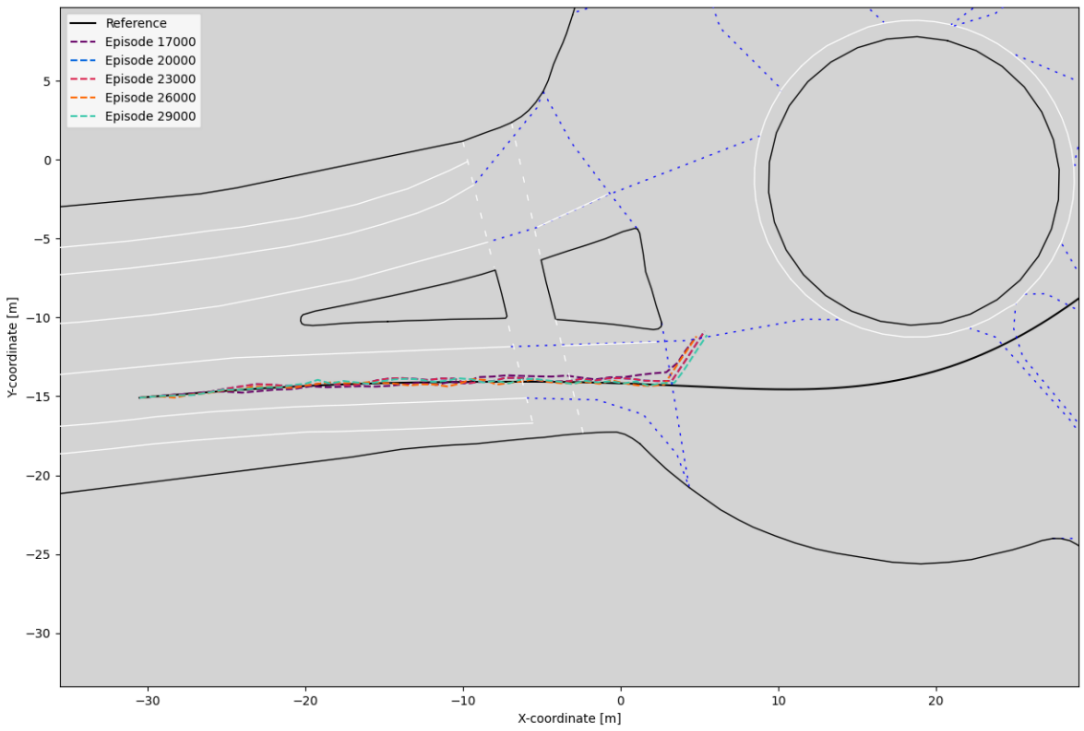


Figure 4.4: When using dynamic rewards the policy might converge to a local point where the vehicle explores less because of dynamic rewards.

4.4 Mastering Navigation Through A Roundabout

In order to find suitable values for all chosen rewards, a part of this work was to train an agent to learn a chosen traffic scenario and teach the vehicle to successfully master the given scenery. For this purpose a roundabout scenery originally located in the US was chosen Fig. 3.2. Tab. 4.6 lists all the previously mentioned and explained rewards and termination states, used during the successful training session. It should be noted that the case of exceeding the time limit did not occur in this case.
Fig. 4.5 shows the trajectories the vehicle learned. It is clearly visible how the agent is able replicate the reference trajectory after enough training time. Episode 1000 shows the still non-smooth unrealistic trajectory which smooths out over the training process. Here the agent is still exploring the beginning of the scenario when there is only a narrow lane to drive on and a vehicle in front and another vehicle in the back Fig. 4.8(a). Episode 10000 shows how the agent still did not fully explore the action space in the beginning as it would crash into the lane. This is a common difficulty with continuous action spaces which can be

Reward	Calculation	Reward value
Lane Collision	-	-20
Vehicle Collision	-	-20
End reached	$\sqrt{(x_t - x_{i=N}^{ref})^2 + (y_t - y_{i=N}^{ref})^2} < 2\,\mathrm{m}$	$+100$
Wrong exit/End missed	vehicle despawned	-5
Time limit exceeded	$t > T_{max}$	-4
Change in orientation	$\lvert\xi_t - \xi_{t+1}\rvert > 0.2\,\mathrm{rad}$	-0.25
Change in velocity	$\lvert v_t - v_{t-1}\rvert > 0.5\,\mathrm{m/s}$	-0.35
Exceeding max. velocity	$v_t > v^{max} = 10\,\mathrm{m/s}$	$-(v_t - 10)^2$
Fall below min. velocity	$v_t < v^{min} = 4\,\mathrm{m/s}$	$-(v_t - 4)^2$
Velocity in def. range	$v^{max} > v_t > v^{min}$	$+1$
Agent close to reference	$\Delta^{ref} < 0.1\,\mathrm{m}^2$	$+5$
Agent lost reference	$\Delta^{ref} > 0.5\,\mathrm{m}^2$	-5

Table 4.6: Rewards which resulted in successful mastering the roundabout hurdle. Here $\Delta^{ref} = \min_i(x_t - x_i^{ref})^2 + (y_t - y_i^{ref})^2$ is the squared distance to closest reference point and only rewarded if the agent hasn't been at that reference point before.

countered with well implemented reward functions or limitations in actions. It should be noted that the reward function was adjusted between episodes 20000 and 30000 when the vehicle got stuck exploring small changes in the returned rewards, as described in Sect. 4.3.4. For this adjustment the change in velocity and orientation rewards were modified to return constant negative rewards instead of punishing the agent with the calculated value directly. Even though it would be favorable to restart the training process from the beginning, this small modification would have not influenced the agents ability to master the beginning of the roundabout as at the time of mastering to enter the roundabout successfully other rewards were the dominating factors. After around 33000 episodes the agent would repeatedly drive through the roundabout successfully.

The upper half of Fig. 4.6 shows the averaged score in this training session over the total number of episodes, divided into smaller sets. The score increases over the number of episodes and in the last episode sets the agent is reaching the end more often than colliding or missing the final state. Additionally, the averaged number of environment steps taken per step and the averaged driven distance are displayed on the lower half of Fig. 4.6. Note how the agent manages to stay alive longer and drive farther at the same time with more experience.

In Fig. 4.7 the distribution of occurring termination cases for each episode set is visualized. Lane collisions were the majority of termination reasons. But in the beginning of the training process small red percentages are noticeable. The first larger red bar is attributable to the vehicle number 11 driving in front of the ego-

Figure 4.5: Selected trajectories from the training cycle with the settings in Tab. 4.6. The according episode can be seen in the legend.

vehicle before entering the roundabout and the later appearing vehicle number 14 following the agent, as seen in Fig. 4.8(a). The second small red bar shows the collision with the vehicle number 12 which is already present in the roundabout as seen in Fig. 4.1. After enough training time the agent knows how to react in that situation, Fig. 4.8(b). The later appearing red parts in the distribution show the collisions with vehicle number 11 and 10 when the vehicle wants to leave the roundabout again, as depicted in Fig. 4.8(c). In the later episodes the agent manages to reach the final state repeatedly, shown in Fig. 4.8(d). Sometimes it would miss the end state, which happened when the vehicle was not less than 2 m away from the final position and left the roundabout causing to despawn, as described in Tab. 4.6. With more training episodes the purple bar would decrease further. It should be noted that the larger blue part between approximately 15000 and 27000 episodes is due to the earlier described phenomenon in which the agent got stuck due to minor reward changes Sect. 4.3.4. After the change the vehicle managed to master the scenario.

Fig. 4.9 shows the average deviation in each episode. The few points with a larger deviation happened due to a minor bug in the code, where the vehicle would leave the roundabout, but the environment was not reset and the simulation continued to run until the computer crashed. Only the later peak is attributable to the

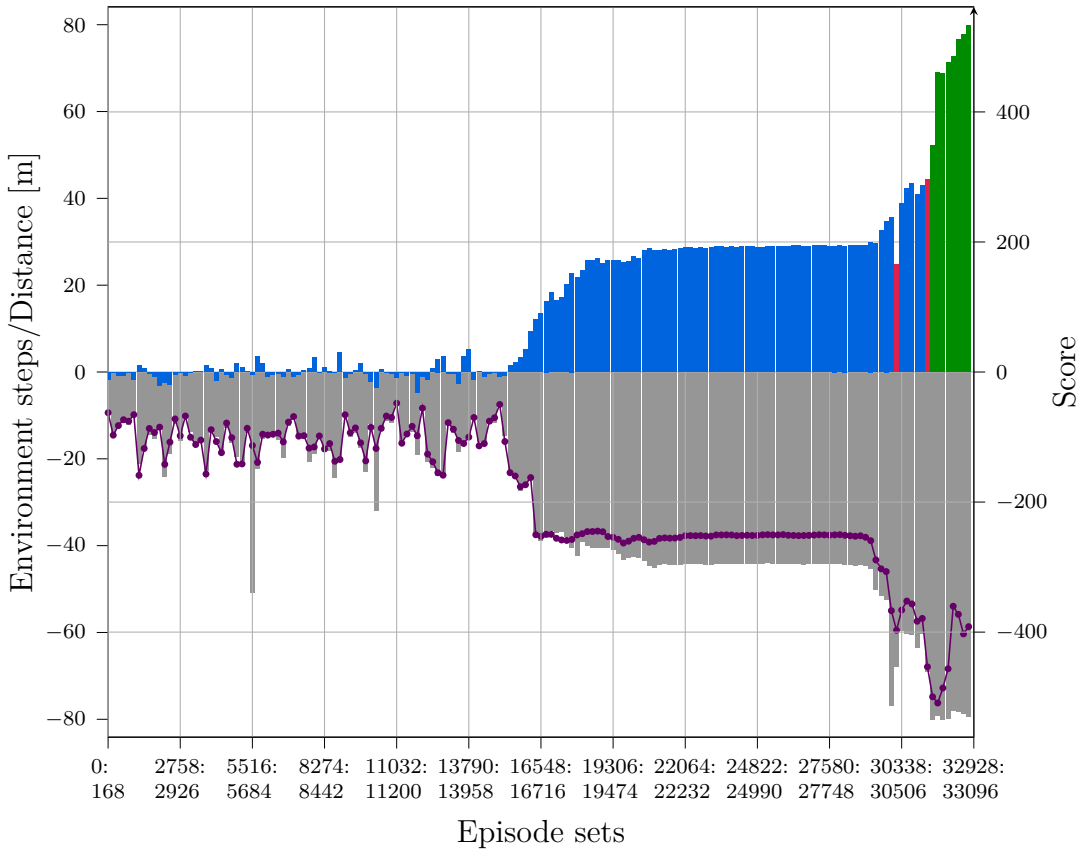


Figure 4.6: Averaged score and averaged episode steps as well as averaged distance over set of episodes from the training with the settings in Tab. 4.6. The upper half shows the score with the bars colored in the most frequent termination case. The lower half shows the averaged number of episodes together with the averaged distance the agent drove for each set of episodes.

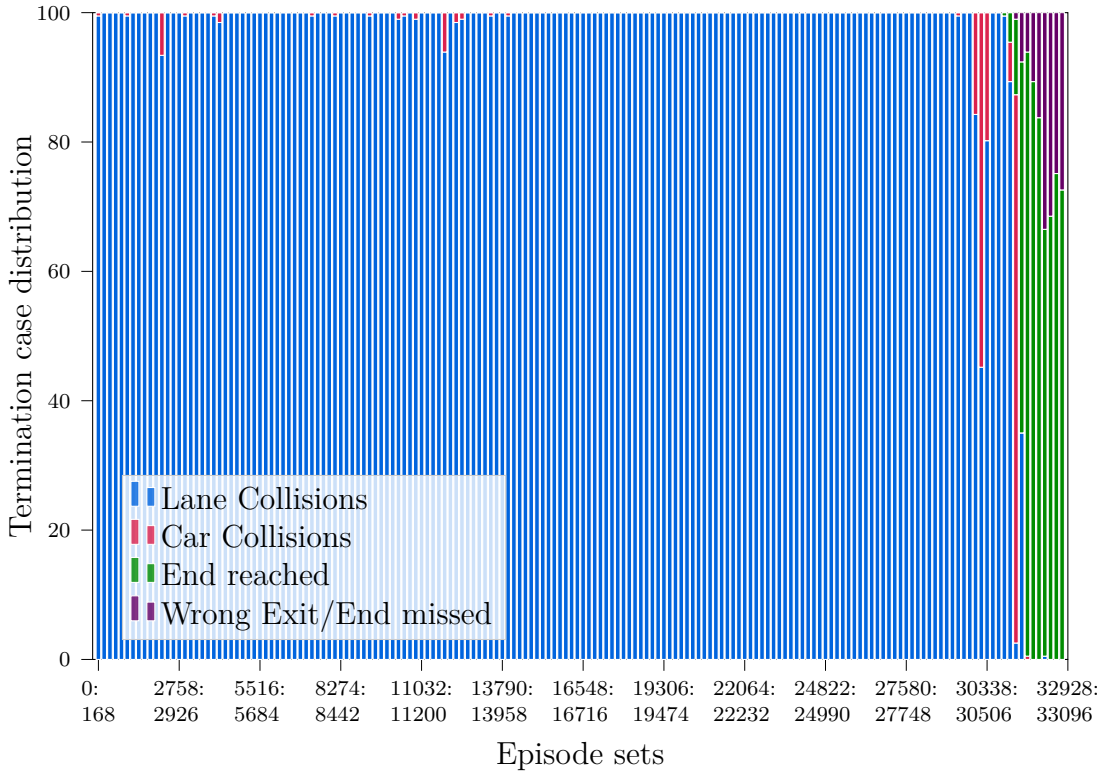
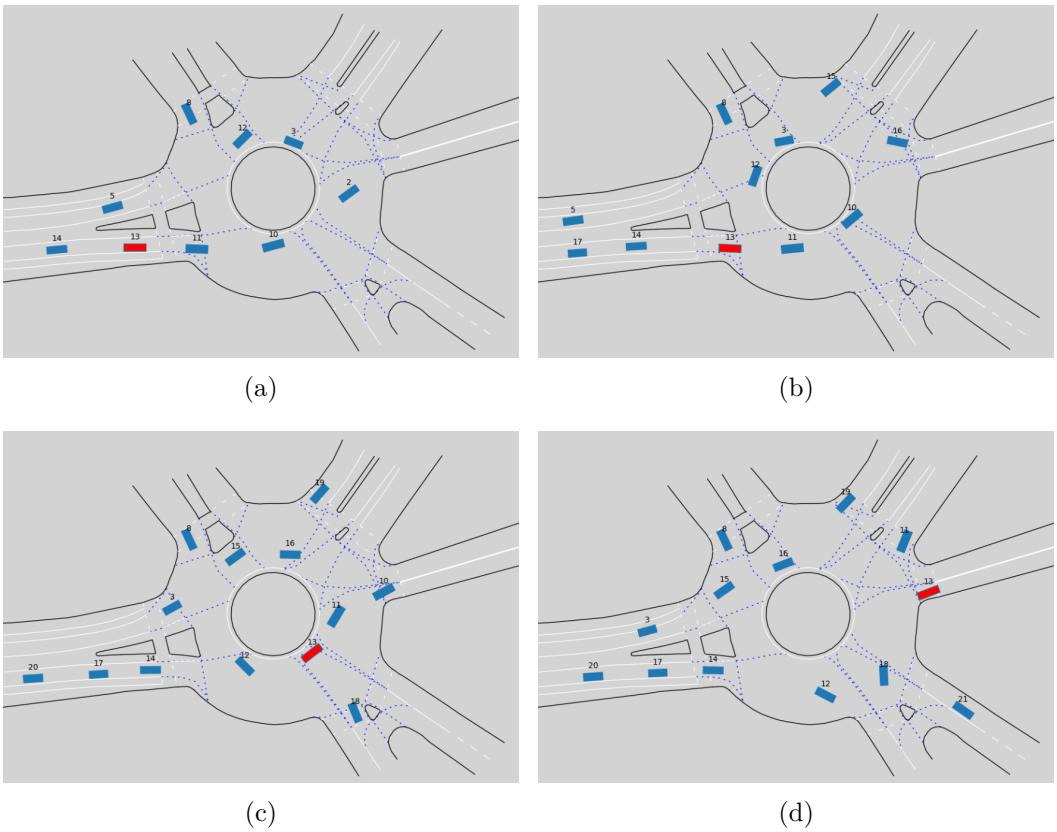


Figure 4.7: Case distribution over episode sets: Lane collision (blue), Car collision (red), reached end position (green), agent missed position barely (purple). Taking an actual wrong exit did not occur.

agent missing the exit and instead keep on driving in the roundabout, following vehicle 11 and sometimes crashing into it. Overall the deviation jumps as the vehicle learns to correct its mistake, but then takes sub-optimal decisions at later environment steps. The almost constant deviation in the middle is the agent getting stuck in the earlier described dead end, Sect. 4.3.4. Only at the end of the training session the overall deviation decreases as the agent finds the optimal policy.

4.4.1 Training Duration

Because the simulation environment was originally designed to study multi-agent prediction behavior, each individual vehicle runs its own sub-thread over the runtime period. Each sub-thread takes care of updating the vehicles individual state, which does not only require a large amount computation power, but also influences the training time when applying a RL-application. Fig. 4.10 shows the



(a) (b)

(c) (d)

Figure 4.8: Images of the agent successfully mastering the roundabout at environment steps (a) 2100 ms, (b) 3700 ms, (c) 6400 ms and (d) 8000 ms

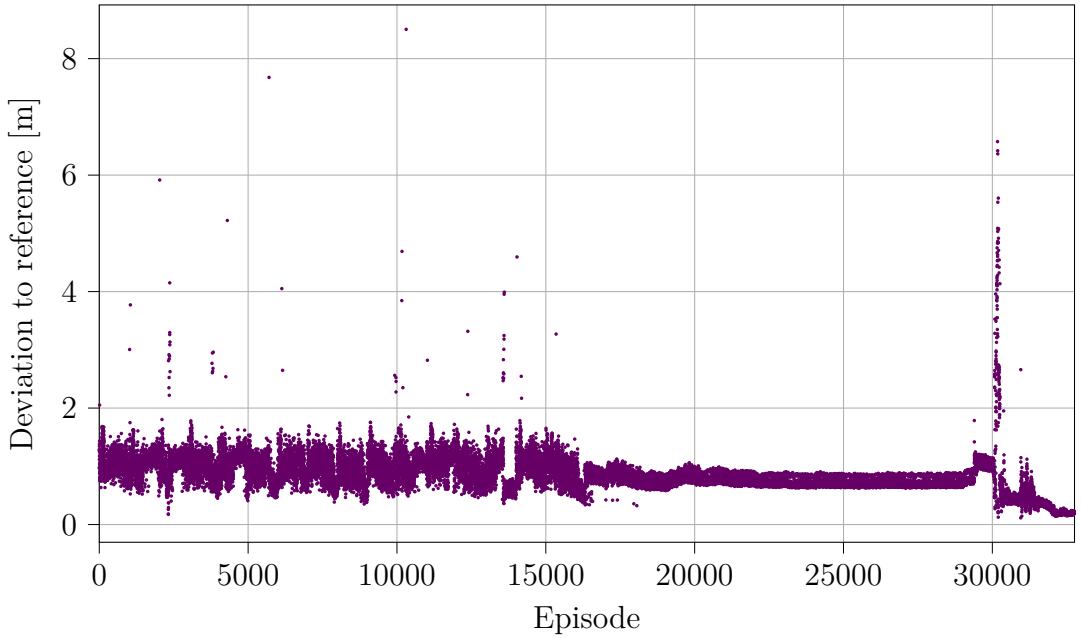


Figure 4.9: Average Deviation in each episode. The agent manages to decrease the deviation repeatedly but also increase when exploring unknown new states.

training duration for chosen number of episodes. As shown, running the simulation for only 1000 episodes takes approximately an hour. This is considerably high compared to other training environments. For around 30000 episodes the simulation trains more than 2 days. The increase in time results due to the agents successful learning in the environment as it manages to stay alive for a longer period of time.

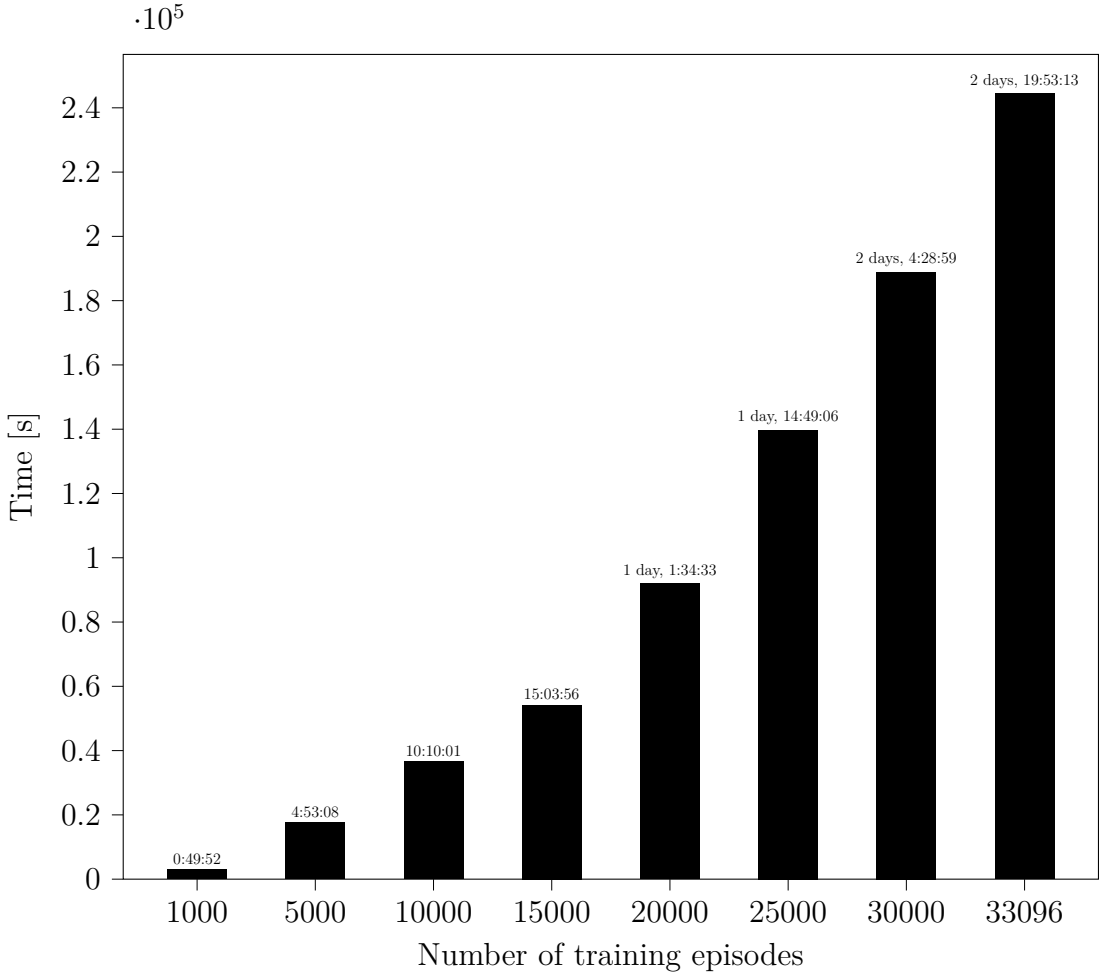


Figure 4.10: Duration of training time for certain number of episodes.

Chapter 5

Conclusion and Outlook

In this work a RL-interface was established, making use of an already preexisting traffic simulation designed for studying behavior prediction in autonomous driving in highly dense traffic environments. After a detailed background about the fundamentals of RL different RL-algorithms were introduced and compared. SAC was selected and implemented. A roundabout scenario was chosen and an agent was trained to master the traffic scenario. For this purpose, the simulation environment was adjusted for a RL training loop and a custom reward function was designed. Different reward strategies were established and evaluated.
The framework was successfully utilized to train an agent in a chosen urban environment. However, the long training times, visualized in Fig. 4.10, raise the question if the framework should be utilized for further application of RL-algorithms to the current state of the simulation.
As explained in Sect. 4.4.1, the simulation starts a single sub-thread for each vehicle which accounts for the vehicle's individual behavior, such as planning a trajectory as well as predicting other vehicles behavior. But for single agent RL-applications only a maximum of two threads is necessary. The first sub-thread runs the simulation environment while the second one takes care of the RL-tasks, such as remembering observations or choosing the action for next environment step. In terms of multi-agent RL additional sub-threads can be started, according to the number of agents acting in the environment. Every other vehicle would continue to follow its recorded trajectory in the dataset only without running a sub-thread. This would reduce the training times and allow faster development.
Further time optimization can be achieved by adjusting the current logging process. For the behavior prediction setting the simulation features an already inbuilt logging system. States of each vehicle at every environment step as well as planned and predicted trajectories are saved in text files. An external visualization module can be used to display the simulation environment Fig. 3.2.

An additional logging module was implemented to save the results of the RL-algorithm, such as the score of each episode, the duration and termination cause. This data is stored in separate text files because the visualization tool follows an established structure when reading the simulation data files. To improve the evaluation process of the results the logging module could be adjusted to feature all data at the same location. Additionally, a faster and less power consuming method can be considered to save the produced data. This could speed up the training duration further.
In terms of visualizing the environment an additional real-time visualization tool is present in the current state of the simulation. Because the inclusion of visual observations of the scenery as an input for the agent is desirable, the present real time visualization tool could be adjusted to serve as an image-generation module which produces pictures adjusted for feature extraction and pattern recognition by neural networks, as done by [ChenYuanTomizuka19b] for example. These images could then be read by the PYTHON side and used as input information/observation. For this purpose, the already existing visualization tools (real-time and replay) could be considered. Furthermore, if the INTERACTION dataset possesses information for spatial visualization image generation, a three-dimensional environment can be considered, which would increase the possibilities for studying RL in autonomous driving.
Finally, training a RL-agent requires generalization across different environmental settings. In terms of autonomous driving tasks the ego-vehicle should train in multiple traffic situations. This includes different urban settings, such as several unique roundabouts with varying sizes and number of exits , intersections of two or more roads, single or multiple lane merging etc. together with variable number of surround vehicles. Needles to say, the user of the present RL-framework can rerun the training loop over and over again, and additionally load a different configuration file, but it would be convenient to start an automated process. An initialization module has been implemented with respect to this work, but it still needs a manual definition of the scenery for the PYTHON client. Therefore this module could be extended to receive the information by the C++ side. Even further, currently the simulation features a configuration file assistance tool which generates an initialization file depending on the desired settings for the prediction tasks (traffic scenario, number of vehicles, time constraints etc.). A similar generation tool could be implemented taking care of generating randomized training situations (changing traffic scenario, determining the number of vehicles, time constraints etc.) and additionally control the training process by starting the server and client itself.

Overall, the already preexisting features for behavior prediction and trajectory planning lead to the simulation being comprehensive, including a lot of different sub-modules and classes. Henceforth, working with the existing code was very

challenging and time consuming. Especially the multi-threading design consisting of a main loop, sub-threads for vehicles and update processes lead to overhauls of the implementations for just small additions and changes. Even though for now the RL-framework is present, larger updates and extensive changes might require large modifications again. To that end, maintenance will be time-consuming and extending the simulation will increase the already present complexity, especially since the previous prediction structures do not require the RL-additions. In terms of that adjusting the simulation environment further might be less favorable and instead creating and developing an additional separate simulation environment from scratch might be preferential. This would come of the advantage that the prediction and RL-tasks would be separated and the current simulation would be less voluminous. Already implemented parts and sub-modules of the current simulation could be copied, reused and adjusted while an RL-optimized simulation environment could be build. Alternatively, countless already existing commercial RL-environments exist which are optimized and maintained by professionals. Making use of an external library, which would optimally allow the integration of the already existing INTERACTION dataset, would save time consuming maintenance and the users of the simulation could focus their work on the development of their algorithms.

Bibliography

[AbbeelLevine22] Abbeel, P.; Levine, S.: Lecture notes of cs285 deep reinforcement learning, 2022.

[Aradi20] Aradi, S.: Survey of deep reinforcement learning for motion planning of autonomous vehicles. CoRR, Vol. abs/2001.11231, 2020.

[BansalKrizhevskyOgale18] Bansal, M.; Krizhevsky, A.; Ogale, A.S.: Chauffeurnet: Learning to drive by imitating the best and synthesizing the worst. CoRR, Vol. abs/1812.03079, 2018.

[Bellman57] Bellman, R.: A markovian decision process. Indiana Univ. Math. J., Vol. 6, pp. 679–684, 1957.

[BrockmanEtAl16] Brockman, G.; Cheung, V.; Pettersson, L.; Schneider, J.; Schulman, J.; Tang, J.; Zaremba, W.: Openai gym. arXiv preprint arXiv:1606.01540, 2016.

[ChenYuanTomizuka19a] Chen, J.; Yuan, B.; Tomizuka, M.: Deep imitation learning for autonomous driving in generic urban scenarios with enhanced safety, 2019.

[ChenYuanTomizuka19b] Chen, J.; Yuan, B.; Tomizuka, M.: Model-free deep reinforcement learning for urban autonomous driving, 2019.

[CoutoAntonelo21] Couto, G.C.K.; Antonelo, E.A.: Generative adversarial imitation learning for end-to-end autonomous driving on urban environments. CoRR, Vol. abs/2110.08586, 2021.

[DuanEtAl16] Duan, Y.; Chen, X.; Houthooft, R.; Schulman, J.; Abbeel, P.: Benchmarking deep reinforcement learning for continuous control. CoRR, Vol. abs/1604.06778, 2016.

[DuanEtAl21] Duan, J.; Ren, Y.; Zhang, F.; Guan, Y.; Yu, D.; Li, S.E.; Cheng, B.; Zhao, L.: Encoding distributional soft actor-critic for autonomous driving in multi-lane scenarios. CoRR, Vol. abs/2109.05540, 2021.

[FehérEtAl19] Fehér, Á.; Aradi, S.; Hegedü s, F.; Bécsi, T.; Gáspár, P.: Hybrid ddpg approach for vehicle motion planning. pp. 422–429, 2019.

[Forbes02] Forbes, J.R.N.: Reinforcement Learning for Autonomous Vehicles. dissertation, University of California at Berkeley, 2002.

[FujimotoHoofMeger18] Fujimoto, S.; van Hoof, H.; Meger, D.: Addressing function approximation error in actor-critic methods. CoRR, Vol. abs/1802.09477, 2018.

[HaarnojaEtAl18] Haarnoja, T.; Zhou, A.; Abbeel, P.; Levine, S.: Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. CoRR, Vol. abs/1801.01290, 2018.

[JesusEtAl21] Costa de Jesus, J.; Kich, V.; Kolling, A.; Grando, R.; Cuadros, M.; Gamarra, D.F.: Soft actor-critic for navigation of mobile robots. Journal of Intelligent & Robotic Systems, Vol. 102, 2021.

[JiaEtAl21] Jia, X.; Sun, L.; Zhao, H.; Tomizuka, M.; Zhan, W.: Multi-agent trajectory prediction by combining egocentric and allocentric views. In 5th Annual Conference on Robot Learning, 2021.

[KiranEtAl20] Kiran, B.R.; Sobh, I.; Talpaert, V.; Mannion, P.; Sallab, A.A.A.; Yogamani, S.K.; Pérez, P.: Deep reinforcement learning for autonomous driving: A survey. CoRR, Vol. abs/2002.00444, 2020.

[LillicrapEtAl15] Lillicrap, T.; Hunt, J.; Pritzel, A.; Heess, N.; Erez, T.; Tassa, Y.; Silver, D.; Wierstra, D.: Continuous control with deep reinforcement learning. CoRR, 2015.

[MnihEtAl13] Mnih, V.; Kavukcuoglu, K.; Silver, D.; Graves, A.; Antonoglou, I.; Wierstra, D.; Riedmiller, M.A.: Playing atari with deep reinforcement learning. CoRR, Vol. abs/1312.5602, 2013.

[MnihEtAl15] Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A.A.; Veness, J.; Bellemare, M.G.; Graves, A.; Riedmiller, M.A.; Fidjeland, A.; Ostrovski, G.; Petersen, S.; Beattie, C.; Sadik, A.; Antonoglou, I.; King, H.; Kumaran, D.; Wierstra, D.; Legg, S.; Hassabis, D.: Human-level control through deep reinforcement learning. Nature, Vol. 518, pp. 529–533, 2015.

[NageshraoTsengFilev19] Nageshrao, S.; Tseng, E.; Filev, D.: Autonomous highway driving using deep reinforcement learning, 2019.

[NaveedQiaoDolan20] Naveed, K.B.; Qiao, Z.; Dolan, J.M.: Trajectory planning for autonomous vehicles using hierarchical reinforcement learning. CoRR, Vol. abs/2011.04752, 2020.

[PadenEtAl16] Paden, B.; Cap, M.; Yong, S.Z.; Yershov, D.; Frazzoli, E.: A survey of motion planning and control techniques for self-driving urban vehicles, 2016.

[PaszkeEtAl19] Paszke, A.; Gross, S.; Massa, F.; Lerer, A.; Bradbury, J.; Chanan, G.; Killeen, T.; Lin, Z.; Gimelshein, N.; Antiga, L.; Desmaison, A.; Kopf, A.; Yang, E.; DeVito, Z.; Raison, M.; Tejani, A.; Chilamkurthy, S.; Steiner, B.; Fang, L.; Bai, J.; Chintala, S.: Pytorch: An imperative style, high-performance deep learning library. In Advances in Neural Information Processing Systems 32, pp. 8024–8035. Curran Associates, Inc., 2019.

[Pomerleau88] Pomerleau, D.A.: Alvinn: An autonomous land vehicle in a neural network. In D. Touretzky (Ed.) Advances in Neural Information Processing Systems, Vol. 1, Morgan-Kaufmann, 1988.

[RengarajanEtAl22] Rengarajan, D.; Vaidya, G.; Sarvesh, A.; Kalathil, D.; Shakkottai, S.: Reinforcement learning with sparse rewards using guidance from offline demonstration, 2022.

[SavariChoe21] Savari, M.; Choe, Y.: Online virtual training in soft actor-critic for autonomous driving. In 2021 International Joint Conference on Neural Networks (IJCNN), pp. 1–8, 2021.

[SilverEtAl14] Silver, D.; Lever, G.; Heess, N.; Degris, T.; Wierstra, D.; Riedmiller, M.: Deterministic policy gradient algorithms. In Proceedings of the 31st International Conference on International Conference on Machine Learning - Volume 32, ICML'14, p. I387I395, JMLR.org, 2014.

[XuEtAl16] Xu, H.; Gao, Y.; Yu, F.; Darrell, T.: End-to-end learning of driving models from large-scale video datasets. CoRR, Vol. abs/1612.01079, 2016.

[ZhanEtAl19] Zhan, W.; Sun, L.; Wang, D.; Shi, H.; Clausse, A.; Naumann, M.; Kümmerle, J.; Königshof, H.; Stiller, C.; de La Fortelle, A.; Tomizuka, M.: INTERACTION Dataset: An INTERnational, Adversarial and Cooperative moTION Dataset in Interactive Driving Scenarios with Semantic Maps. arXiv:1910.03088 [cs, eess], 2019.

[ZhanEtAl21] Zhan, W.; Sun, L.; Ma, H.; Li, C.; Jia, X.; Wang, D.; Shi, H.; Clausse, A.; Naumann, M.; Kümmerle, J.; Königshof, H.; Stiller, C.; de La Fortelle, A.; Tomizuka, M.: Interpret: Interaction-dataset-based prediction challenge iccv2021 competition, 2021.

Appendix

A.1 Contents Archive

There is a folder **PRO_064_Maroofi/** in the archive. The main folder contains the entries

- **PRO_064_Maroofi.pdf**: the pdf-file of the thesis PRO-064.
- **Data/**: a folder with all the relevant data, programs, scripts and simulation environments.
- **Latex/**: a folder with the *.tex documents of the thesis PRO-064 written in Latex and all figures (also in *.svg data format if available).
- **Presentation/**: a folder with the relevant data for the presentation including the presentation itself, figures and videos.

Erklärung

Ich, Sean Maroofi (Student des Maschinenbaus an der Technischen Universität Hamburg, Matrikelnummer 51334), versichere, dass ich die vorliegende Projektarbeit selbstständig verfasst und keine anderen als die angegebenen Hilfsmittel verwendet habe. Die Arbeit wurde in dieser oder ähnlicher Form noch keiner Prüfungskommission vorgelegt.

Unterschrift	Datum