AGH University of Krakow

FIELD OF SCIENCE: ENGINEERING AND TECHNOLOGY

SCIENTIFIC DISCIPLINE: AUTOMATION, ELECTRONICS, ELECTROTECHNICS AND SPACE TECHNOLOGIES

DOCTORAL THESIS

Prediction model of an autonomous vehicle's behavior, based on Artificial Intelligence methods

Author: Nikodem Pankiewicz

Supervisor: dr hab. inż Piotr Bania

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

Kraków, 2023

AKADEMIA GÓRNICZO-HUTNICZA IM. STANISŁAWA STASZICA W KRAKOWIE

DZIEDZINA: NAUK INŻYNIERYJNO-TECHNICZNYCH

DYSCYPLINA: AUTOMATYKA, ELEKTRONIKA, ELEKTROTECHNIKA I TECHNOLOGIE KOSMICZNE

ROZPRAWA DOKTORSKA

Model predykcji zachowania się pojazdu autonomicznego oparty na metodach Sztucznej Inteligencji

Autor: Nikodem Pankiewicz

Promotor rozprawy: dr hab. inż Piotr Bania

Praca wykonana: Akademia Górniczo-Hutnicza im. Stanisława Staszica w Krakowie, Wydział Elektrotechniki, Automatyki, Informatyki i Inżynierii Biomedycznej

Kraków, 2023

Special thanks to all who contributed to this work, especially my beloved wife for her boundless patience and to my supervisor for his invaluable guidance.

Abstract

We are witnessing rapid automotive development and an increasing demand for newer and more versatile driver assistance systems (ADAS). With the simultaneous intensive development of artificial intelligence based on deep neural networks, it is possible to replace classical solutions and extend the scope of their operation. One area in which AI may be profitable is the solution to the problem of vehicle behavior planning.

The thesis focuses on inventing a methodology for developing a vehicle behavior policy in adaptive cruise control mode. The policy plans a target acceleration value which is achieved in a designated time horizon by planning and executing a trajectory. The strategy is intended to operate on highways, and the planning takes into account the road situation, which includes the road topology and all vehicles detected by the perception sensors of the controlled car. The policy aims to be effective in a real environment and handle a broad range of road situations. For this purpose, research is being conducted on the possibility of using reinforcement learning (RL) and imitation learning methods.

Preliminary analysis showed that imitation learning, which is based on collected real data, is limited by a restricted amount and poor data quality. On the other hand, the policy optimized by reinforcement learning methods in simulation often appears to be suboptimal in real-world conditions. It is due to the deviation between simulation and reality (sim2real gap) and missing replicated real-world phenomena.

The dissertation presents methods to mitigate these problems. Firstly, it presents a method of improving the quality of real-world data using numerical optimization. Secondly, it presents a way to combine both learning methods to minimize the sim2real gap and increase the distribution of situations known to the agent.

The policies obtained using the presented solutions are assessed using the proposed evaluation methods. Additionally, the policy effectiveness is compared with the performance of a test driver and a baseline policy trained with the standard RL approach.

Furthermore, the thesis presents a methodology for evaluating the reliability of the behavior model using statistical analysis to examine the impact of input elements into the system on the selected action.

The results suggest that the proposed algorithms are promising from the perspective of developing vehicle behavior planning. The presented evaluation method allows a better understanding of the predicted behavior policy and increases its reliability.

Streszczenie

Współcześnie obserwowany jest dynamiczny rozwój branży motoryzacyjnej oraz rosnący popyt na nowoczesne i kompleksowe systemy wsparcia dla kierowców (ADAS). Wraz z intensywnym postępem możliwości sztucznej inteligencji opartej na głębokich sieciach neuronowych, możliwe jest zastąpienie tradycyjnych rozwiązań ADAS oraz rozszerzenie zakresu ich działania. Jednym z obszarów, w którym wykorzystanie sztucznej inteligencji może przynieść wymierne korzyści, jest rozwiązanie problemu planowania zachowania ruchu pojazdu.

Niniejsza praca skupia się na opracowaniu metodologii pozwalającej na stworzenie strategii zachowania pojazdu w trybie adaptacyjnego tempomatu. Strategia planuje wartość docelowego przyspieszenia pojazdu, która ma zostać osiągnięta w zdefiniowanym horyzoncie czasowym. Zadane przyspieszenie jest osiągane przez zaplanowanie oraz wykonanie trajektorii. Działanie strategii przewidziane jest na drogach szybkiego ruchu, a planowanie bierze pod uwagę sytuację drogową, w której skład wchodzi topologia drogi oraz wszystkie pojazdy wykryte przez sensory percepcji sterowanego auta. Strategia ta ma być skuteczna w rzeczywistym środowisku i radzić sobie z różnorodnymi sytuacjami. W tym celu prowadzone są badania nad możliwością wykorzystania metod uczenia ze wzmocnieniem oraz imitacji zachowań.

Wstępna analiza wykazała, że metody imitacji zachowań, które bazują na danych rzeczywistych, limituje ograniczona ilość danych oraz słaba ich jakość. Natomiast strategia pozyskana przy użyciu metod uczenia ze wzmocnieniem w symulacji, często okazuje się suboptymalna w warunkach rzeczywistych. Wynika to z odstępstw pomiędzy symulacją a rzeczywistością (sim2real gap) oraz brakiem możliwości zamodelowania naturalnych zjawisk w symulacji.

Niniejsza praca przedstawia metody niwelujące wady poszczególnych metod uczenia. Po pierwsze, przedstawia on procedurę poprawy jakości danych rzeczywistych przy użyciu optymalizacji numerycznej. Po drugie przedstawia sposób na połączenie obu metod uczenia w celu zminimalizowania sim2real gap i powiększenia zakresu scenariuszy drogowych, w procesie nauki strategii zachowania.

Strategie zachowania pozyskane przy pomocy prezentowanych rozwiązań są oceniane przy pomocy zaproponowanych metod ewaluacji. Dodatkowo ich efektywność jest porównana z wydajnością testowego kierowcy oraz bazową strategią pozyskaną standardowymi metodami RL.

Praca prezentuje także metodologię pozwalająca na ocenę wiarygodności modelu zachowania przy pomocy statystycznej analizy badającej wpływ elementów wejściowych do systemu na wybrane przez sieć neuronową zachowanie.

Wyniki pracy sugerują, że zaproponowane algorytmy są obiecujące w perspektywie rozwoju tempomatu adaptacyjnego opartego na sieciach neuronowych. Przedstawione metody ewaluacji pozwalają na lepsze zrozumienie przewidywanego zachowania strategii oraz zwiększają jej wiarygodność.

Contents

List of Figures

Glossaries

ACC	Adaptive Cruise Control mode
ADR	Automatic Domain Randomization
ANN	Artificial Neural Network
ANOVA	Analysis of Variance
API	Application Programming Interface
BC	Behavioral Cloning
CNN	Convolutional Neural Network
COM	Center of Mass
CPU	Central Processing Unit
DA	Domain Adaptation
DDPG	Deep Deterministic Policy Gradient
DQN	Deep Q Network
DR	Domain Randomization
ego/host	the object that is controlled by a presented algorithm
FL	Follow Lane maneuver
GAN	Generative Adversarial Networks
GIS	Geographic Information System
GPS	Global Positioning System
GPU	Graphics Processing Unit
HD	High Definition
IG	Ingredient Gradients
IL	Imitation Learning
IMU	Inertial Measurement Unit
KL divergence	Kullback-Leibler divergence
KPI	Key Performance Indicator
LCL	Lane Change Left
LCR	Lane Change Right
LiDAR	Light Detection and Ranging
LQR	Linear Quadratic Regulation
LSTM	Long-Short Term Memory
MARWIL	Monotonic Advantage Re-Weighted Imitation Learning

MDP	Markov Decision Process
MPC	Model Predictive Control
NGSIM	Next Generation Simulation dataset
object	regards any vehicle on the road
OEM	Original Equipment Manufacturer
OSM	Open Street Map
PLCL	Prepare for Lane Change Left
PLCR	Prepare for Lane Change Right
POMDP	Partially Observed Markov Decision Process
PPO	Proximal Policy Optimization
Radar	Radio Detecting and Ranging
RBF	Radial Basis Function
ReLU	Rectified Linear Unit
RL	Reinforcement Learning
RNN	Recurrent Neural Network
RPM	Rotation per Minute
RSS	Responsible Sensitive Safety
SAC	Soft Actor-Critic
SAE	Society of Automotive Engineers
SARS'	SARS' tuple (s_t, u_t, r_t, s_{t+1})
SGD	Stochastic Gradient Descent
sim2real gap	the difference between simulation and real environment
SLSQP	Sequential Least Squares Programming
SVM	Support Vector Machine
target	a vehicle perceived by the host's sensors
TD	Temporal Difference
TTC	Time To Collision
UAV	Unmanned Air Vehicle
VCS	Vehicle Coordinate System - originates from the center of the rectangle that circumscribes the vehicle, x-axis points forward, y-axis points left
VRAM	Video Random Access Memory
WCS	World Coordinate System

Notations

In Reinforcement Learning

t	discrete time step $t \geq 0$
T	time horizon, number of time steps,
S	set of environment states of the environment
s_t	state in time t
s_d	terminal state at time T
A	action space
u_t	policy action in time t
$u_{e,t}$	expert policy action in time t,
$\Omega \in \mathbb{R}$	observation space
$O(o_{s,t}\|s_t, u_{t-1})$	probabilistic observation function
o_t	state observation in time t
$P(s_{t+1}\|s_t, u_t)$	probabilistic transition model in MDP
$\Psi(s_{t+1}\|s_t, u_t)$	probabilistic transition model in POMDP
$R(s_t, u_t, s_{t+1})$	reward function
G_t	discounted cumulative reward for episode
τ	episode trajectory, consists of T SARS' tuples $\tau = \{(s_0, u_0, r_0, s_1), (s_1, u_1, r_1, s_2), \ldots, (s_{T-1}, u_{T-1}, r_{T-1}, s_d)\}$
G_τ	return of the trajectory τ
$\pi(u_t\|o_t, \theta)$	parameterized policy which determines the next control signal
π_e	expert policy, often performed by human
π_θ^*	optimal policy
θ	parameters vector of behavior policy π_θ
γ	discount-rate parameter $\gamma \in [0, 1]$
$h(s, u, \theta)$	parameterized function that returns numerical preferences for each state-action pair
$f^{\text{ANN}}(s, \theta)$	ANN function
D_{train}	training dataset
D_{opt}	Dataset with optimized samples
$V(s)$	state value
$Q^\pi(s, u)$	q-value; action value

μ	mean of normal distribution
σ	standard deviation of normal distribution
$f_{\mathrm{PDF}}(x\|\mu,\sigma)$	probability density function
$U(\theta,\tau)$	expected return of trajectory in REINFORCE algorithm
$b(s_t)$	baseline for calculating td-error
ς	td-error
β	MARWIL scaling parameter in MARWIL
$\mathrm{clip}(r_t(\theta),\epsilon)$	clipping function
$\hat{g}$	PPO gradient estimator
$\hat{A}_t$	advantage estimation of action
λ	PPO hyper parameter
ϵ	clipping parameter in PPO and in ϵ-greedy policy
$\mathbb{R}$	set of real numbers
$\mathbb{R}^N$	N-dimensional set of real numbers
$\mathbb{E}$	expected value
In control/physics	
x, y	coordinates in WCS
s, d	coordinates in Frenet Coordinate System [1] defined along the center of the lane, s - longitudinal, d - lateral (perpendicular) coordinate
v_s, v_d	velocity in Frenet Coordinate System
$v_{s,\max}$	maximal longitudinal speed
a_s, a_d	acceleration in Frenet Coordinate System
ζ_{CS}	rotation in specified in subscript coordinate system
$x^{\mathrm{vcs}}, y^{\mathrm{vcs}}$	coordinates in VCS, axis x points forward, y to the left
$v_x^{\mathrm{vcs}}, v_y^{\mathrm{vcs}}$	velocity in VCS in axis x,y
v^{vcs}	absolute in VCS
$a_x^{\mathrm{vcs}}, a_y^{\mathrm{vcs}}$	acceleration in VCS in axis x,y
a^{vcs}	absolute acceleration in VCS
δ	steering angle
ϑ_t	throttle control value $\vartheta_t \in [0,1]$
b_t	brake control value $b_t \in [0,1]$
$a_{s,\mathrm{target}}$	planned longitudinal acceleration
β	behavior state vector
s_i^{v}	state vector of i$^{\mathrm{th}}$ vehicle
$\psi(s_t^{\mathrm{v}},\vartheta_t,\delta_t,b_t) \rightarrow s_{t+1}^{\mathrm{v}}$	function that transforms vehicle state s_t^{v}
F	Newtonian force
ρ	state of road model
$k(s_{i,t}^{\mathrm{v}}, a_{s,\mathrm{target}}, \rho)$	trajectory function based on vehicle state, control, and road model state
w	width

l	length
h	height
m	mass
$r_{w,i}$	wheel radius for i $^{\text{th}}$ wheel
c_d	aerodynamic draw coefficient
ι	air mass density
g_r	gear ratios
l_f	distance between COM and front axis
l_r	distance between COM and rear axis
ω	angular velocity
I	moment of inertia
F_{aero}	aerodynamic resistance force
ι	mass density of air
a_{f}	front projected area
v_{wind}	wind velocity
r_w	tire radius
τ	torque
$v_{\text{s\_limit\_exec}} = \frac{v_s}{v_{s,\max}}$	level of preset speed limit execution
J	objective function
c_i	cost term of objective
g_i	constrain term of objective
w_i	weight of i$^{\text{th}}$ cost term c_i
$f_{\text{bsf}}(x)$	B-Spline function
κ	road curvature
Δ_r	sensing range
α	step-size coefficient in SGD
In Agent Explainability	
$g_i(o)$	attributation of i^{th} feature of observation
o^{base}	baseline observation for Integrated Gradients algorithm
$f^{\text{ANN}}(o)$	ANN function
$\frac{\partial f^{\text{ANN}}(o)}{\partial o_i}$	gradient of $f^{\text{ANN}}(o)$ along the i$^{\text{th}}$ dimension
$u_{\text{left avail}}, u_{\text{right avail}}$	Boolean value that informs whether changing lane to the left or right is safe according to RSS
ρ	population Pearson correlation coefficient
$\text{cov}(R(X), R(Y))$	are the covariance of the rank variables,
$\sigma_{R(X)}\sigma_{R(Y)}$	are the standard deviations of the rank variables.
r_s	Spearman correlation coefficient
d_i	the difference between the two ranks of each observation,
n	number of observations.

α	step-size parameter in IG
$u_{\mathrm{LC(R/L)\ avail}}$	boolean value which represents whether LCL or LCR maneuver is safe - calculated based on RSS rules
H_0	null hypothesis
H_1	alternative hypothesis

Chapter 1

Introduction and Motivation

1.1 Introduction

Recently the automotive industry has set itself the goal of minimizing the number of car accidents, especially fatal ones. Most of these accidents are caused by human error. Therefore, automakers are trying to support drivers by implementing newer and more advanced assistance systems, which prevent common errors, control the driver's attention, and support decision-making. The ultimate goal is to replace the human driver with fully automated vehicles and eliminate the human factor. Society of Automotive Engineers (SAE) describes [2] the highest level 5 of vehicle autonomy as a mode that can drive under all possible conditions, and the vehicle featured with it may not be equipped with a pedal or steering angle. To achieve such automation, one needs to develop a holistic system that consists of perception, behavior planning, and vehicle control modules. These subsystems must be integrated to cooperate with each other in order to guarantee safe and efficient driving.

Over the years, remarkable progress has been made in all of these areas. Perception collects relevant information about the position and state of a road and objects. The development of machine learning has led to significant progress and growth in this field. The control theory has been constantly developed since the middle of the last century. Conversely, behavior planning, which considers traffic situations and outputs directives to the control modules, appears to be the least researched, albeit very interesting and challenging. The difficulty of this area is to create a solution that will ultimately handle all situations encountered on the road in an effective or at least safe manner.

Until now, the most frequent attempts to solve this problem were to code the responses to possible situations in the form of a decision tree or a procedure algorithm. However, hand-coding the rules that apply to all situations seems to be a tremendous and error-prone solution. Moreover, it does not assure the optimality of the final solution, however, it enables complete control and transparency of the system. More promising might be the data-based methods, which provide behavior rules based on analyzing the spectrum of traffic situations and optimizing behavior to be the most efficient.

One such method is Reinforcement Learning (RL) which trains the behavior policy to be optimized according to some reward function. During the training, behavior policy is optimized by an agent which explores a training environment by selecting available actions. The environment transits its current state to

the next state according to the selected action. Additionally, it provides the agent with a reward that estimates how good the selected action was in a given state. Initially, the agent randomly explores the environment because it does not know the effect of the selected action. Over time, it learns how to act to get higher rewards and utilizes that knowledge to discover more profitable states. The training aims to learn how to behave in a given state to maximize the sum of future rewards.

In the case of driving, it is not practical to use RL algorithms to learn how to drive the vehicle on the real road. To increase training's safety, speed, and cost-effectiveness, it is more judicious to take advantage of computer traffic simulations. However, the policy can learn only as much as it has managed to explore the environment. After transferring the policy to the real environment, it may be only as effective as of close the simulation was to the real world. The policy may encounter the states that were not present during training, the perception of states may vary, the transition between states may differ, or the behavior of traffic participants may be unnatural. The difference between simulation and real-world is called the sim2real gap and is one of the significant issues of applying RL in real-world applications [3].

1.2 Motivation

The dissertation was completed as a part of an industrial Ph.D. program and carried out in an automotive company focusing on evolving advanced driver assistance systems (ADAS). Research regards developing the behavior policy responsible for planning future vehicle behavior with the utilization of Reinforcement Learning and Imitation Learning methods. Among various features of automated driving, the thesis focuses on developing a policy for Adaptive Cruise Control mode. The policy is developed in the interests of its application in customers' vehicles.

The thesis primarily concentrates on understanding the problem of transferring developed policies from simulated environments into real ones. It focuses on developing methods that alleviate the sim2real gap and distributional shift issues. The proposed methods integrate simulated data with test drive data in the learning process. The approach derives inspiration from offline and online Reinforcement Learning (Sec. 3.1, 3.2). Offline RL is used for learning only the fixed dataset of experience collected in advance, usually by some human expert. This type of data harvesting provides close to optimal real-world experience, facilitating learning and eliminating exploration problems. However, the limited scope of the dataset restricts policy to handle a wide range of situations. On the other hand, online RL allows unrestricted exploration of the simulated environment, therefore, learning all possible state transitions and action consequences. It leverages offline learning in terms of explored situations, however, simulated data is affected by the aforementioned sim2real gap.

The advantages of these two approaches seem to mitigate each other's drawbacks. Therefore the methods proposed in this thesis concentrate on the development of techniques that enable applying data from various sources in order to train behavior-driving policy which is efficient in a real environment. At the same time, the work studies the disadvantages of offline learning and proposes a method for increasing the quality of real data in order to support the learning process.

1.3 Thesis

The objective of this research is to demonstrate the efficacy of utilizing real-world data during the training process to enhance the performance of outcome policies in natural circumstances. The training regards the optimization of behavior policy which is responsible for planning the directives for the control module of a vehicle in Adaptive Cruise Control mode. It is hypothesized that policy training that uses both real-world and simulated data in presented convention significantly improves performance in the natural environment compared to using only simulated data. It is also expected that a presented combination of offline and online learning methods alleviates the limitation of both training methods.

1.4 Work content

Section 1.5 provides a literature review of automated car movement planning. Section 1.5.1 demonstrates a general approach to path planning using Machine Learning methods. Section 1.5.2 outlines the evolution of Adaptive Cruise Control technology from classical algorithms to machine learning-based approaches. The following Section 1.5.3 is devoted to the issues involved in transferring the functionalities developed in the simulation to natural conditions. The last part 1.6 highlights the elements presented in the reviewed articles, which an attempt to improve is the subject of this dissertation. It precedes Section 1.7, which clarifies the thesis's contribution.

The following Chapter 2 presents the objective of ACC controller 2.1 and project requirements. Requirements involve integration with the motion stack of the automated vehicle (Sec. 2.2) and utilizing the vehicle sensors stack (Sec. 2.3). The last Section 2.4 describes the simulation tool used for experiments and Section 2.5 clarifies applied safety rules.

The next Chapter 3 explains Reinforcement Learning methodology (Sec. 3.1.1) and state-of-the-art RL algorithms (Sec. 3.1.2-3.1.14). It also includes a description of the Imitation Learning approach and its methods (Sec. 5.4-3.2.3). Finally, the chapter describes the problem of divergence between simulation and reality (Sec.3.3) and the standard techniques used to alleviate consequential issues (Sec.3.3.1-3.3.3).

Following Chapter 4 consists of a description of an experimental setup developed for achieving the project objective according to project specification (Sec. 2). Successive parts describe the experimental simulated environment (Sec. 4.1), collected dataset (Sec. 4.2), Neural Network architecture (Sec. 4.3), its inputs (Sec. 4.1.2), outputs (Sec. 4.1.3) and reward function which drives training process (Sec. 4.1.4). It also presents additional assumptions and limitations of the considered solution (Sec. 4.4).

Chapter 5 explains invented methods of developing ACC RL agents capable of driving in natural conditions. Firstly it presents the general idea, which is then elaborated in the following sections: Subsections 5.2 and 5.3 present the method of improving the dataset concerning action feasibility; Section 5.4 shows the courses of Offline RL training and their comparison; Section 5.5 presents the methodology of combining online training in simulation with collected data.

The next Chapter 6.2 firstly presents the evaluation criteria and methodology developed for comparison of policies trained with different approaches (Sec. 6.1). Then it presents the evaluation process of trained be-

havior policies according to the presented method. The last Section 6.5 summarizes experiments, interprets results, and shows the limitations and applicability of the proposed solution.

The penultimate Chapter 7 introduces the novel algorithm for explainability of ANN-based RL policies, which allows the verification of agents in terms of compliance with human intuition. The method presents an approach for agents that operate either in continuous or discrete action space.

1.5 Literature Review

1.5.1 Machine Learning for Path Planning

In the literature, it could be found a number of works that focus on the application of Reinforcement Learning in the domain of vehicle control either in behavior or path planning part ([4], [5], [6], [7],[8]). However, only a few of them have considered evaluating a trained policy in an actual vehicle, from which the most important are presented below.

The seminal work ALVINN [9] which used behavior cloning for steering a car, was created in 1989. The authors proposed an end-to-end approach that is based on training a neural network responsible for controlling a car. The ANN took as input the raw camera image and range sensor measurements. It outputs the steering angle that the vehicle should follow to stay on the road. For training, to preserve a sufficient number of samples, the author used simulated road images and calculated corresponding distances. To limit necessary computational power and to alleviate the sim2real gap, images were created in very low resolutions to be indistinguishable from the real ones, which also were captured in low definition.

Similarly to [9], another works [10] and [11] used the raw images as input data and trained a classifier based on the human demonstrations to indicate the required steering angle. In these cases, images had higher resolution. Therefore, the authors decided to use Convolutional Neural Networks (CNN) as a backbone of ANN. Work [11] proposed to reduce Imitation Learning's drawbacks resulting from a limited number of samples by additional data augmentation. This approach enriched the training dataset by adding additional images showing the vehicle fluctuating from the center of the lane and rotating relative to the route's direction.

While the methods discussed above have proven to be effective to some extent, it's worth noting that they can be limited by some factors. First of all, the end-to-end approach limits the end-users in the matter of interfering with the entire system. Moreover, it becomes impossible to debug that kind of system, understand decisions taken by an agent, or prove its reliability. This stays in contradiction to the automakers' attitude which prioritizes modularization of the entire software stack to preserve system transparency.

According to a study [12], trained agents are usually only able to perform the specific actions they were designed for, which is a major issue. The actions that the agents were trained to perform in the cited works ([9], [10], [11]) were limited to following a lane, making a turn, and avoiding collisions. However, this restricted set of functions cannot be considered a complete driving system, as users need to be able to configure the system to meet their needs. This includes specifying driving destinations and performing maneuvers that are described in higher-level abstractions than control signals.

A more recent and more advanced approach, which is also based on Imitation Learning, was proposed in a work that introduced ANN called ChauffeurNet [13]. In contradiction to previous works, it proposed to use pre-processed raw sensor data as an input to the system. The study utilized a road model, traffic data, and road users' states as inputs. These inputs were transformed into a spatial representation as multiple images from a top-down perspective. The images were fed into a Convolutional Recurrent Neural Network to predict the future path of the vehicle. The path included the vehicle's position, orientation, and speed at various time intervals. The ChauffeurNet was trained on a dataset of 26 million examples, which is equivalent to two months of continuous driving experience.

While the generated model was effective in controlling an actual vehicle, it was found to be less effective than the classical motion planning methods. This study highlights the difficulty of establishing the superiority of machine learning over classical approaches in the complex field of motion planning. The author suggests that in complex multi-stage processes like driving, machine learning could be employed to assist classical algorithms in specific modules rather than replacing the entire process.

Similar approach to the ChaufferNet was presented by the authors of SafetyNet [14] who adopted pre-processed objects as input to the ANN. However, these objects were represented as polygons and demonstrated to the SafetyNet in the form of a graph. Using Graph Neural Networks permitted the representation of complicated data structures that are not limited by coarse grid format. For training SafetyNet, the authors used Imitation learning with the additional loss function which preserved the safety of the trajectory. They used logs from 380 hours of real driving and additionally introduced perturbations into presented observations to extend the state distribution. The model was evaluated by driving 150 miles on roads in San Francisco. The authors admitted that, even though the model had managed to perform various challenging maneuvers successfully, it still needed to be wrapped by some deterministic safety layer to guarantee safety.

1.5.2 Adaptive Cruise Control

Conventional Cruise Control was the first system on the market to control the vehicle's speed to drive at preset velocity by regulating the throttle [15]. The next iteration of such a controller introduced Adaptive Cruise Control (ACC) which may adjust the hosts' longitudinal velocity to the vehicle detected by sensors. To preserve a preset range to the vehicle in front, it could brake and change throttle level [16, 17]. Most of the deployed strategies for ACC in customer vehicles are private, constituting the intellectual property of OEMs. However, in the literature, we can find multiple implementations of such a controller [18, 19, 20, 21, 22].

For example, [18] proposed using Linear Quadratic Regulation (LQR) to achieve optimal trajectory regarding fuel consumption. However, [19] pointed out that the time of response of the LQR controller was longer than a system that utilizes Model Predictive Control (MPC). Moreover [22] indicated that LQR may result in uncomfortable high jerk trajectory. As regards MPC, it is commonly utilized and tested for ACC implementation (ex. [20, 21, 22]). Work [22] reported that their solution was acceptable, particularly at closer distances between the host and the vehicle being followed. Numerous studies have indicated that the accuracy of the used model substantially impacts MPC's control performance.

Besides classical approaches, researchers incorporated Reinforcement Learning algorithms into ACC controllers to overcome issues reported by previous works. Work [23] used Deep Deterministic Policy Gra-

dient method (DDPG) [24] to train policy which maximizes multi-objective reward function. Agents got positive rewards when they were within a specified distance from the leading target, taking into account the host vehicle's velocity, acceleration, and jerk. Work reported that RL policy was able to balance between followability and comfortability in contrast to a control system based on the LQR.

Another work [25] proved its performance against the MPC approach in terms of safety, comfort, and inference time. Work utilized the DDPG algorithm [24] to train continuous control policy optimized with respect to safety, efficiency, and comfort. They applied data from the NGSIM dataset [26] to train and test agents, designed a new reward function, and incorporated safety checks to avoid collisions.

Next paper [27] introduced a system named SAINT-ACC which also trained an agent to optimize traffic efficiency, safety, and driving comfort based on training two RL agents. The first one, the 'TTC agent' used discrete action space to define the optimal time to collision (TTC) value and was trained with the DQN algorithm. The second one, 'ACC agent' was trained to specify headway to the leading target. During the training of the subsequent agent, the TTC value predicted by the first agent was utilized as an input to the reward function. The agents were trained in two basic highway scenarios: entering and exiting the highway. The work achieved superior results compared to the approach mentioned in [25].

In addition to the control part, the ACC module is intended to select a target vehicle to which it should adjust velocity. It requires a deeper understanding of traffic situations than presented in the above works. Target selection should be able to predict the movement of adjacent vehicles and understand the interaction between objects. It should predict situations such as a vehicle cutting in between the host and the lead vehicle or a sudden lane change by the followed target.

The standard approach for the ACC task with multiple targets may be found in [28, 29]. The problem was divided into three major tasks - multi-target tracking objects, primary target selection, and ACC controller integrated with the collision avoidance system. The target selection module intended to choose a vehicle that most affects the host vehicle among all neighbors. The authors proposed to assign targets to lanes and choose the nearest one in the host lane. In order to predict target lane changes, the researchers developed a fuzzy logic estimator that delivers a probability indicator for host lane occupancy. It considered the lateral position and velocity of the vehicle with respect to the host car. Another approach for target selection was presented in [30] where authors proposed to predict the lane change intention based on a sliding window support vector machine (SVM) [31]. They proposed a feature vector that consists of relative position, velocity, and acceleration between the host and target. They tried to train SVM with kernel functions such as linear, quadratic, cubic, and radial basis functions (RBF). The authors conducted tests with sliding windows ranging from 0 to 5 seconds, with an interval between frames of 0.2 seconds. Based on the training results on the NGSIM dataset [26], the RBF kernel with a 2.2s sliding window was found to have the best performance.

The most recent approach for predicting the intentions of nearby vehicles focuses on using ANNs [32, 33, 34, 35]. The widely used method relies on presenting the road situation from a bird's eye view perspective as an image. And using it to train the Convolutional Neural Network (CNN) to forecast whether the target vehicle will change lanes to the left, right, or stay in its current lane.

1.5.3 Sim2Real Transfer

Transferring the policy trained in simulation to a real environment requires applying special techniques to close the gap between domains (Sec. 3.3). Usually, there are applied methods such as Domain Adaptation (DA, Sec. 3.3.2), Domain Randomization (DR, Sec. 3.3.1), or real data are utilized (Sec. 3.3.3). In the domain of automated driving, most works ([36],[37]) focused on enhancing simulated sensor data to be more realistic. On the other hand, some researchers [38, 39] concentrated on refining simulation by diversifying training scenarios and exploiting corner cases present during real drivings.

The example of enhancing sensor data by introducing GANs is presented in [36] and [37]. The work [36] introduced SurfelGAN aimed at generating realistic images from simulated data. The inputs to the GAN, which is the root of the proposed network, are the textured voxels (surfels) which were formed based on LiDAR points and camera images. Additionally, the simulation needs to provide semantic and instance segmentation masks. These data, which are actually easily accessible from simulation, are converted to realistic camera images for training the modules of the automated vehicles such as object detectors, behavior predictors, or motion planners.

The second work [37] used conditional GAN to create two pairs of the encoder-decoder system and one discriminator. One encoder-decoder was trained to parse virtual images from the driving simulation to the segmentation masks. In parallel, the second encoder-decoder learned to translate segmentation masks to realistic images. Accordingly, the discriminator was trained to recognize matching pairs of virtual and real images. For evaluation, the authors conducted two training sessions with different methods of closing the sim2real gap. The first was supplied with the presented approach, and another utilized the domain randomization technique. Both agents were trained in the same environment and evaluated in a different second one. The evaluation demonstrated the superiority of the authors' method over domain randomization.

Other examples of closing the sim2real gap in the automated driving application are the works of [38] and [39], which considered the approach of DR (Sec. 3.3.1). The work [38] focused on enhancing the simulation to make the generated scenarios even more challenging for the agent. By increasing the set of corner cases, an agent was being adapted to real-world incidences.

The work regarded parameterization of the default scenario with a set of adjustable values. The parameters are optimized with respect to the reward that the agent collected facing a given scenario. The parameters were adjusted when the agent barely exceeded the performance expectation. Every time adjusted parameters created a slightly different scenario which is supposed to be more challenging for the agent, and therefore the agent is supposed to be more robust. Such guided domain randomization provided a curriculum of environments from the easiest to the most challenging. The evaluation of training showed that the agent is more efficient and preserved more safety than while trained in an environment with fixed parameters.

However, another work [39] proposed an entire framework for learning driving automatically with RL. It is worth emphasizing the techniques the authors proposed to alleviate the problem of the sim2real gap. First of all, for each training episode, they randomized the parameters of the simulation to adapt agents to diverse environments and avoid overfitting. To reflect the problems with extrinsic sensor calibration, the position of the camera sensor was randomized. Also, to be robust to variation in vehicle dynamics, the physical properties of the vehicle were also sampled from distribution. Additionally, the agent robustness

was enhanced with data augmentation which was based on rotating the camera images. The augmentation influenced generalization and decreased the demand for training data.

1.6 Missing Factors

The aforementioned works contributed to developing vehicle motion planning based on machine learning methods. Starting with [9], most authors chose supervised learning or imitation learning technique which requires training data collected in advance. However, [13] has already stated that using even a large dataset is not enough to cover the distribution of all possible real-world scenarios. Therefore limited to fixed dataset training is not sufficient to solve a real-world problem; therefore, the training dataset needs to be augmented somehow.

The most popular approach for state augmentation is using simulation, which allows for creating an infinite number of traffic situations. To fully exploit simulation capabilities, training should follow the Reinforcement Learning methodology [40]. However, the utilization of simulation in policy optimization results in a policy that tends to be adjusted to the simulated environment which may diverge from the natural one due to the sim2real gap.

One possible method to overcome this limitation is a combination of different sources of data. Therefore learning methods may provide a solution that is optimized for adequate scenario distribution and is optimized with regard to real data.

However, mixing the data sources or training methods may introduce the issue known in the literature as catastrophic forgetting, to which ANNs are extremely susceptible. Catastrophic forgetting occurs when training data is not distributed equally during training. ANN tends to forget what it learned previously when data that was presented in the early stages of training is not repeated later. In the considered case, the catastrophic forgetting may be noticeable when the agent is first trained on real-world data and then improved in the simulation. By keeping this training order, the agent can better fit the simulated data, even if it was first trained on the dataset.

It is worth mentioning that the data-based method has another disadvantage: the possibility of corruption in state-action samples. Corruption may occur when the actions are not suitable for the assigned states. The suboptimal samples may be inferred from faulty actions performed by a human expert or from improper labeling. Poor quality of the dataset may result in a derated final quality of the developed agent.

The other not thoroughly addressed in the literature problem results from the fact that most of the work designed artificial agents that consider only sensor data for selecting an action. Typically, the agent maps the current state, which is observed by the sensors, with the action that the human driver performed. That approach misses the intention of the end-user of the driving system. Because the optimal action heavily depends not only on the road situations but also on the goal of driving. For example, an agent, which is driving through an intersection, needs to know the passenger's destination to select an appropriate control decision. The same regards the velocity at which the user wants to drive. The absence of such information in the dataset and the lack of methods to provide it to the system excludes the agent's training with intentional control.

One more thing that is required from considered driving systems is providing transparency. Automotive OEMs prefer motion modularized system architecture in which each module is transparent and explainable. Such an approach allows for debugging, helps to understand the made decisions and to foresee its strengths and weaknesses. The above-described works show that some researchers consider such an approach, dividing motion architecture into route planning, behavior planning, trajectory generation, and control modules. While classical algorithms satisfy transparency requirements, the ANN-based modules still lack explainability. They remain considered a black box that does not know what they pay attention to and why they make particular decisions.

1.7 Work Contribution

The thesis aims to propose a comprehensive method of developing a driving behavior policy that is robust under real-world circumstances using RL methods (Sec. 5). The behavior policy is responsible for predicting the next acceleration value, which should be performed by a vehicle within a defined time horizon in order to fulfill the objectives of the adaptive cruise controller.

The proposed method comprises several solutions that comply with the issues presented in previous works, summarized in Section 1.6. The thesis focuses on overcoming the problem of data limitation in Imitation Learning algorithms. Therefore it incorporates the utilization of simulation within Reinforcement Learning training alongside real-world data. Such an enhancement provides better coverage over state distribution due to the exploitation of simulation and, at the same time, minimizes the sim2real gap due to the utilization of the real data. The work also addresses the issue of catastrophic forgetting by suitable incorporation of real data during the RL training process. Additionally, RL training is supported by a curriculum learning method that contains a Domain Randomization mechanism (3.3.1) which supports a broad range of environmental characteristics.

Another contribution of the thesis is the proposition of solving the issue related to data imperfection (Sec. 1.6), which appears in offline reinforcement learning algorithms (5.4). The work proposes alleviating the problem with a proprietary algorithm called Optimization-based Imitation Learning (Sec. 5.2). The approach involves the numeric optimization of the actions included in the dataset. The optimization leads to the higher quality of the outcome policy trained with Imitation Learning methods.

Additionally, responding to market requirements regarding the system transparency 1.6, the thesis shows the integration of developed policy into the modular hierarchical architecture of the motion system 2.2. In this architecture, only one part is based on ANN, and the rest are derived from legacy solutions which actually are not a matter of the thesis. However, the ANN-driven module is the core of the driving agent and remains an integral part of a holistic solution. Besides, the system is suitable for receiving the control signals from a vehicle user and adapting behavior accordingly. Moreover, to increase system transparency, the work proposes a methodology that enables understanding the agent's decisions and deciphering the black box ANN. The novel method of explainability of the RL agent is presented in Section 7 [41].

As regards the evaluation of the proposed method, the thesis suggests the appropriate to the presented problem evaluation process (6.2). The process is divided into simulation-aided and real-data parts. The simulation part aims to pre-select the most promising agent candidates. Whereas the latter tests the agent in

naturalistic circumstances and returns actual values of Key Performance Indicators (KPIs), which score the agent's efficiency in several different aspects.

Chapter 2

Project Foundations

2.1 Control Objectives

The proposed methods have been implemented and evaluated on the Adaptive Cruise Control (ACC) driving mode, which fulfills a multi-factor objective. First of all, it should ensure a safe and comfortable driving experience. In that mode, users can set the desired maximum velocity at which the vehicle should drive. In order to achieve this, the controller takes into account various factors, including the road conditions and the state of other vehicles on the road. Therefore, the vehicle only drives at the set speed when the road ahead is empty, or there are no expected obstructions. If it is foreseeable that some other vehicle will cut into the host's lane soon, increasing acceleration to reach the set speed would be suboptimal. If there is a vehicle in front of the host, the controller adjusts velocity to maintain a safe distance. The same applies if some vehicle is preparing to cut in front of the host. In that case, the controller should predict its future movement and adjust its own velocity accordingly.

Adapting speed to the vehicles, particularly those whose speed fluctuates due to disrupted traffic or faulty perception, directly impacts passenger comfort. To guarantee the highest level of comfort for passengers, it is essential for the ACC system to adhere to specific acceleration and deceleration thresholds. By doing so, the system can minimize abrupt movements, ensuring a smoother and more comfortable journey for passengers.

Moreover, to maintain velocity, the controller should select the target vehicle, which may impact the host's driving course. Most of the time, the target vehicle is a car that drives in front of the host preceding it. However, in more complex scenarios, for example, when the adjacent car is preparing to make a cut in between the host and the vehicle ahead, also this car should be considered a target.

To summarize, ACC aims to control the host vehicle in such a way as to maximize the configured speed and preserve safety. Besides that, it should increase driving comfort by suppressing the oscillations resulting from the movement of the target vehicles. The adequate selection of targets and prediction of their future states would eliminate rapid speed changes. Moreover, it should minimize heavy braking events, avoid dangerous cut-ins, and suppress the speed oscillation concerning the target vehicle.

parameter	**description**
w	width
l	length
h	height
m	mass
$r_{w,i}$	wheel radius for i $^{\text{th}}$ wheel
c_d	aerodynamic draw coefficient
a_f	vehicle front projected area
ι	air mass density
g_r	gear ratios
l_f	distance between COM and front axis
l_r	distance between COM and rear axis

Table 2.1: static parameters of vehicle that are contained in vector $\boldsymbol{p}$

The conducted experiments assume that driving takes place on the highway (as Section 4.1 describes) at speeds ranging from 0 to 35 m/s, therefore incorporating both fast driving and traffic congestion.

The problem is to control the vehicle object to fulfill the ACC objective. First of all, the vehicle state is represented as a vector:

$$s^v = \begin{bmatrix} x, \\ y, \\ v_x, \\ v_y, \\ \zeta_{WCS} \end{bmatrix} \tag{2.1}$$

where ζ_{WCS} is rotation. The parameters of the vehicle are shown in Table 2.1,

The function

$$\psi(s_t^v, \vartheta_t, \delta_t, b_t) \rightarrow s_{t+1}^v \tag{2.2}$$

transforms vehicle state in discrete time steps based on control inputs: ϑ_t is throttle input to the engine, δ_t is a steering angle, b_t is a brake input.

The body dynamics are defined as:

$$m\ddot{x} = (F_{x,fl} + F_{x,fr})\cos(\delta) - (F_{y,fl} + F_{y,fr})sin(\delta) + F_{x,rl} + F_{x,rr} - F_{\boldsymbol{aero}} \tag{2.3}$$

$$m\ddot{y} = (F_{x,fl} + F_{x,fr})sin(\delta) + (F_{y,fl} + F_{y,fr})cos(\delta) + F_{y,rl} + F_{y,rr} \tag{2.4}$$

$$I\omega = l_f((F_{x,fl} + F_{x,fr})sin(\delta) + (F_{y,fl} + F_{y,fr})cos(\delta)) - l_r(F_{y,rl} + F_{y,rr}) \tag{2.5}$$

where x denotes that that forces F acts longitudinally and y laterally. Subscripts fl, fr, rl, rr relate accordingly to the front left, front right, rear left and rear right wheel as presented in Figure 2.1. Whereas ω is the angular velocity, δ denotes the steering angle, l_f, l_r are the distance from the center of mass (COM) to the front and rear axis respectively, and I is a moment of inertia $I = ((0.5(l_f + l_r))^2 m)^{-1}$.

Whereas F_{aero} is an aerodynamic resistance force and may be modeled as

$$F_{\text{aero}} = 0.5 \iota c_d a_{\text{f}} |v_x + v_{\text{wind}}|(v_x + v_{\text{wind}}) \tag{2.6}$$

where ι is the mass density of air, c_d is an aerodynamic draw coefficient, a_{f} is the vehicle front projected area, v_{wind} is the wind velocity.

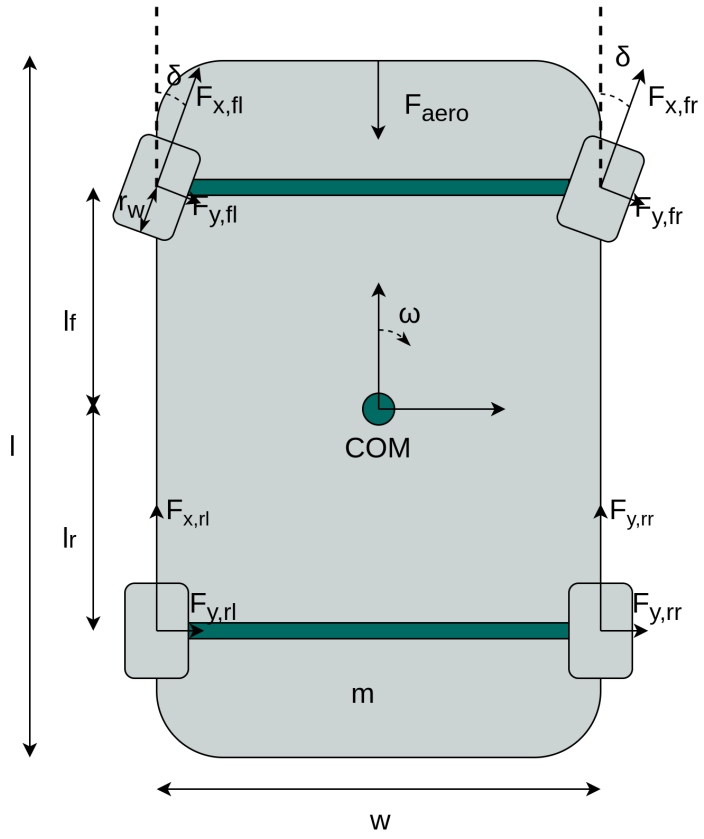


Figure 2.1: Schematic view of a vehicle dynamics system.

The tire force $F_p = \{F_{x,fl}, F_{x,fr}, F_{x,rl}, F_{x,rr}\}$ is related to wheel torque τ_w and the relationship is given by:

$$I_w \dot{\omega} = \tau_w - r_w F_p \tag{2.7}$$

where I_w is wheel inertia, r_w is effective radius of the tire and ω is the tire angular velocity.

τ_w results from throttle input to the engine, torque converter, and transmission as presented in Figure 2.2. The engine torque τ_e is determined by throttle input ϑ and engine angular velocity as defined in [42, 43]. Then τ_e is converted by torque converter τ_c and passed to the wheels by transmission τ_w. The thesis uses three different models of engines - gas, diesel, or electric.

The transmission is parameterized by set of gear ratios $g_r \in \{g_{-1}, g_0, g_1, \ldots, g_N\}$ where N is typically limited to 6.

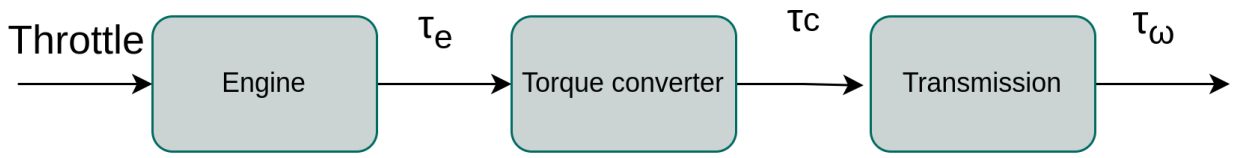


Figure 2.2: Torque transmission

$$\tau_w = \frac{\tau_c}{g_r} \tag{2.8}$$

where g_r is gear ratio and

$$\omega_t = \frac{\omega_w}{g_r} \tag{2.9}$$

The throttle input ϑ, steering angle δ, and brake input b constitute the direct way of controlling the vehicle object.

However, in the specified problem, the control input is $u \in A \in [-1, 1]$. The u is a normalized value of the longitudinal acceleration $a_{s,\text{target}}$ to be achieved in the next 0.5s $(t + 1)$. The value of u is then mapped into acceleration range $a_{s,\text{target}} \in [-3.5, 1.5] m/s^2$ as defined in Equation (4.1).

The target acceleration $a_{s,target}$ is achieved by calculating, updating, and executing future trajectories. The proprietary trajectory function

$$k(s_t^v, a_{s,\text{target}}, \rho) \rightarrow \mathbb{R}^{12} \tag{2.10}$$

takes as an input the actual state of the vehicle s_t^v, target acceleration $a_{s,\text{target}}$, and road state ρ. Based on that it plans the continuous trajectory of control inputs to achieve target acceleration in 0.5s and position the vehicle in the center of the lane. The continuous trajectory is represented by three cubic polynomials, each for one control input ϑ, δ, b.

The control takes place in a system defined as a Partially Observed Markov Decision Process (POMDP):

$$P = (S, A, \Psi, R, \Omega, O, \gamma) \tag{2.11}$$

where S is set of all possible states of the environment $s_t \in S \in \mathbb{R}^M$, where s_t is an environment state in time t; A is an action space, and $u \in A \in [-1, 1]$; $\Psi(s_{t+1}|s_t, u_t)$ is a probabilistic transition function that describes the probability of transitioning from state s_t to next state s_{t+1} when action u_t is selected; $R(s_t, u_t, s_{t+1}) \rightarrow r_t \in \mathbb{R}$ assigns reward for executing selected control u_t in state s_t to next state s_{t+1}; $\Omega \in \mathbb{R}$ is an observation space and $O(s_t, u_{t-1}) \rightarrow o_{s,t}$ is a function calculates observation $o_{s,t}$ of the current state s_t. The last $\gamma \in [0, 1]$ is a discount factor.

The transition model Ψ defines model and probababilites of reaching state s_{t+1} from state s_t and control signal u_t:

$$\Psi(s_{t+1}|s_t, u_t) \tag{2.12}$$

where u_t is an action provided by some stationary policy π parameterized by vector θ based on the observation $o_{s,t}$ of current state in time t:

$$\pi(u_t|\theta, o_{s,t}), \quad u_t \in [-1, 1] \tag{2.13}$$

The control objective is to maximize the sum of discounted rewards:

$$J = \left(\sum_{t=0}^{T-1} \gamma^t R(s_t, u_t, s_{t+1}) \right) \tag{2.14}$$

where T is a finit planning horizon and reward function $R(s_t, u_t, s_{t+1})$ is defined as:

$$R(s_t, u_t, s_{t+1}) = -(v_{s,\max} - v_{t+1}^{\text{vcs}})^2 - (a_{s,t+1}^{\text{vcs}})^2 - c_0(\Delta s) - c_1(\Delta s) \tag{2.15}$$

where v_{t+1}^{vcs} is the absolute velocity of the controlled object in the Vehicle Coordinate System in state s_{t+1}; $v_{s,\max}$ is targetted preset velocity; a_{t+1}^{vcs} is absolute acceleration in VCS achieved by executing u_t in s_t.

$$c_0(\Delta s) = \begin{cases} 1, & \text{if } \Delta s \leq d_{\text{lon\_min}} \\ 0, & \text{if } \Delta s \geq d_{\text{lon\_min}} \end{cases} \tag{2.16}$$

where $d_{\text{lon\_min}}$ is defined as (2.19) and Δs is the distance alongside lane centerline between the front bumper of the controlled vehicle and the rear bumper of the vehicle in front.

The term c_1 determines the collision event of host and front vehicles

$$c_1(\Delta s) = \begin{cases} 1, & \text{if } \Delta s \leq 0 \\ 0, & \text{if otherwise,} \end{cases} \tag{2.17}$$

The state $s_t \in \mathbb{R}^M$ is a vector that describes the environment state and consists of states of the controlled vehicle ($i = 0$), all other N vehicles $s_1^v, \ldots, s_N^v$, and the state of the road model ρ.

$$s_t = \begin{bmatrix} s_i^v \\ s_{i+1}^v \\ \vdots \\ s_N^v \\ \rho \end{bmatrix} \tag{2.18}$$

The subject vehicle is controlled by $\pi(u_t|\theta, o_{s,t}), u_t \in [-1, 1]$ and the rest of the vehicles $\{1, 2, \ldots, N\}$ is controlled by some heuristic policy $\pi_i(s_t) \rightarrow \{\vartheta_t, \delta_t, b_t\}$. The control policy does not contribute to the control of the other vehicles however they may react on actions performed by the policy.

The observation space Ω consists of observation of the host object, adjacent vehicles, and road. $\Omega = \Omega_{\text{host}} \times \Omega_{\text{objects}} \times \Omega_{\text{road}}$

$\Omega_{\text{host}} \in \mathbb{R}^4$ consists of information about absolute velocity in VCS (v^{vcs}), level of preset speed limit execution ($v_{\text{s\_limit\_exec}} = v^{\text{vcs}}/v_{s,\max}$), absolute acceleration (a^{vcs}) and selected acceleration in last time step ($a_{s,target,t-1}$).

$\Omega_{\text{objects}} \in \mathbb{R}^{10\times 11}$ describes up to 10 vehicles closest to the host object. It includes relative to the host features such as longitudinal and lateral distance$(x^{\text{vcs}}, y^{\text{vcs}})$, velocity $(v_x^{\text{vcs}}, v_y^{\text{vcs}})$), and acceleration $(a_x^{\text{vcs}}, a_y^{\text{vcs}})$. It also consists data about target's rotation (ζ_{vcs}) and its dimensions (w, l).

$\Omega_{\text{road}} \in \mathbb{R}^{6\times 10\times 3}$ represent the physical lane markers in front of the host vehicle. Ω_{road} consists of 10 points on each lane marker and their positions $x^{\text{vcs}}, y^{\text{vcs}}$ in VCS and encoded lane marker type. Section 4.1.2 provides a detailed description of the observation space.

2.2 Motion Stack

Behavior planning is a part of the holistic architecture of automated motion systems (Fig. 2.3). The motion system consists of several parts beginning from the perception of the vehicle's surroundings through the planning route, behavior, and future trajectory until to update of the actuators in the control module.

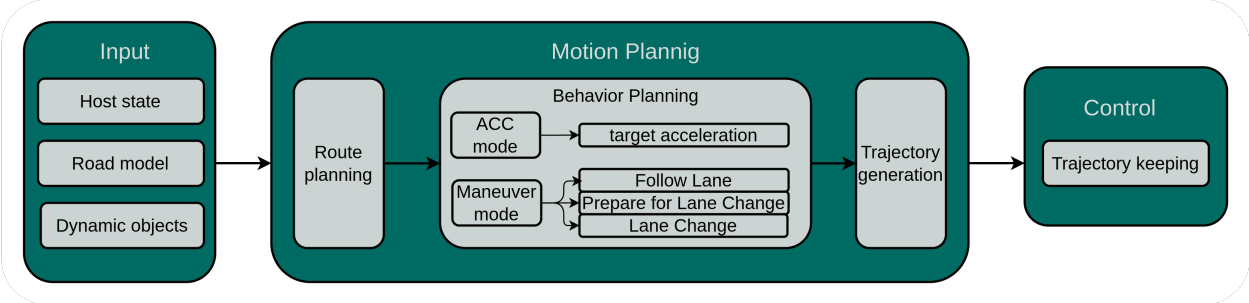


Figure 2.3: Motion planning architecture consists of 3 major modules. Perception provides data about traffic participants, a detailed road model, and any static object within the range of the host sensors. Relying on that information, planned route, and user preferences, the behavior planning module selects an appropriate control signal which supplements the trajectory generator. The generator calculates the continuous reference trajectory executed by a control module.

All modules work and cooperate as follows:

- **Perception** module uses all sensors available on the vehicle to represent the situation on the road and vehicle state. The sensor stack (Figure 2.4) consists of all-around radars, a front-facing camera, GPS, and IMU. The information from GPS, and IMU, with additional readings from wheel encoders and platform components, enables estimating vehicle state as its velocity, acceleration, position in the World Coordinate System (WCS), and yaw rate. Radars provide information about objects around the vehicle, their positions in VCS, relative speed, dimensions, and type. Front front-facing camera detects lane markers on the road and objects that remain redundant information to the radar output. We assume that the maximum detection range is 150m. However, it heavily depends on the traffic situation, present objects, and weather. The performance of perception directly impacts the steering effectiveness.

- **Routing** module is in charge of finding the route between the actual position and the driving destination and providing a sequence of lane-level goals that should be followed. This module requires HD maps that specify road geometry, road lane arrangement, lane geometry, driving direction, and connections to each other. Additionally, they consist of all meaningful information for driving decisions, such as traffic signs, lights, and road surface markings.

- **Behavior planning** module chooses the parameters of the driving trajectory according to an available interface that the trajectory planning module provides. The most common decision is to select the maneuver that should be performed (e.g. follow the lane, change lane, abort maneuver). However, it could also be reference signals such as desired speed, acceleration, offset from lane center, distance

to the following car, or its combination [44]. The behavior planning module makes decisions based on the features returned by the perception module, given goal by routing, or user preferences (for example max speed, min distance to the leading vehicle). In the presented Adaptive Cruise Control mode, this module selects the target acceleration signal to be reached within a specified time range.

- **Trajectory planning** generates a continuous reference trajectory taking into consideration the output from the behavior planning module (for example the targeted acceleration in ACC mode). This module should consider the vehicle's dynamic state and model, localization within the lane, and proximity to other vehicles to produce a feasible, safe, and efficient trajectory.Trajectory can be defined as a polynomial that represents the control input value over time.

- **Control** module takes as input vehicle state and reference trajectory and executes it setting control signals directly to throttle, steering angle, and braking actuators.

2.3 Sensor Stack

In order to conduct experiments, we utilized a vehicle equipped with sensors such as a front-facing camera, all-around radars and IMU. Perception software stack employs data from all radars and processes it into a meaningful representation of surrounding objects. First, a point cloud is generated from radar detections or range-doppler maps. Then, the Neural Network detects and classifies objects based on that [45, 46, 47]. In applied stack, ANN can distinguish between classes such as a vehicle, a truck, a pedestrian, and others. Detections are returned 20 times per second and contain information about objects' position, rotation, velocity, and dimension.

Meanwhile, the front-facing camera monitors the road, capturing images 36 times per second. On each image, the embedded software detects lane markers and constructs meaningful road representations. Lane markers are described by parameters of a cubic polynomial function, its width, visible range, and quality of detection. Additionally, the software provides the marking type of the lines in order to inquire information about the possibility of lane changing. The limitation of vision data is that it only detects lane markers on the host lane and on the nearest adjacent lanes. The maximal range of detection is 150m; however, in practice is limited by vehicles that occlude lane markers, resulting in a mean range of 60m.

A vision system could also detect visible vehicles and may be used alternatively as a source of object data. However, the radar system has better accuracy in determining vehicles' speed. At the same time, the vision outperforms the radars in the precision of objects' dimensions and classification. Optionally these two sources may be fused and processed to provide the most accurate result [48, 49].

Additionally, internal vehicle sensors such as IMU, wheel encoders, and GPS provide information about the state of the host vehicle. The state contains actual velocity, acceleration, steering angle, yaw rate, and rest information about steering mechanisms such as braking pedal, blinkers, throttle, etc. It may also include information about the targeted speed that the driver specified.

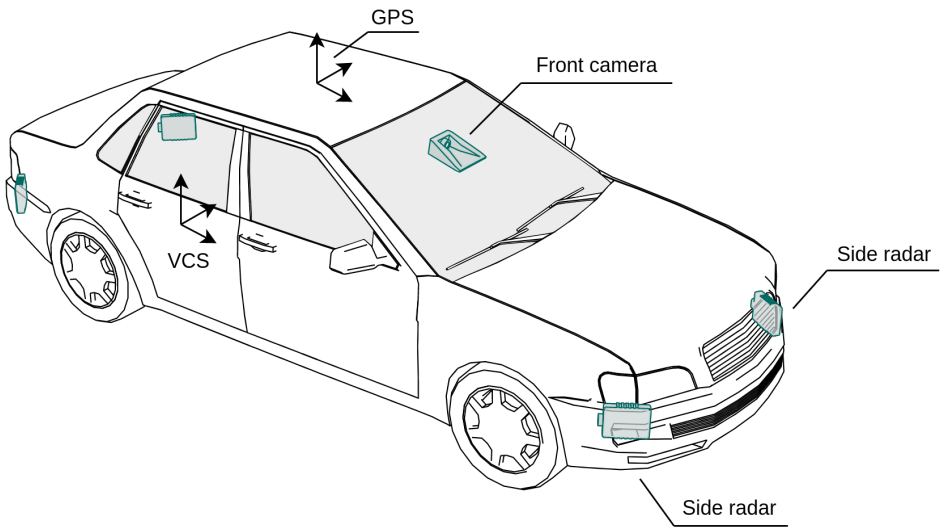


Figure 2.4: Sensor stack consists of all-around radars, front-facing camera, and GPS antenna for surround perception. Additionally, wheel encoders, IMU, and other internal sensors are used for estimating vehicle state.

2.4 Simulation

Simulation software was created by the [50] and further was developed internally. A package was created to simulate various situations on the highways based on the randomization of starting parameters and stochastic behavior of simulated drivers. As regards the definition of road map it might be imported from real-world maps represented in e.g. OpenStreetMap [51] format. The simulated traffic participants are controlled on the basis of predefined heuristics, which are supposed to resemble real drivers' behavior. They may proceed to the target destination, change lanes, overtake, avoid collisions, merge in and out of the highway, or their behavior may be randomized to the extent of traffic law.

In order to simulate a broad range of scenarios, the software allows for the creation of different behavior models of driving actors by specifying heuristics parameters. The parameters refer to values such as speed deviation from speed limit, accelerating pattern, behavior changing interval, distance to the followed vehicle, offset from lane center, craving for changing lanes, and a chance for letting in other vehicles in front of them. The actors may be assigned different types of vehicles, which are specified by the following parameters: dimensions (width, length, height), axis distances from COM, range of speed and acceleration, dynamic parameters such as engine parameters (min/max RPM, torque), gearbox ratios, and wheel dimensions.

These parameters are utilized to determine the vehicle's next position, rotation, velocity, and acceleration accordingly to 2.1. The simulation provides two control models: bicycle kinematic model [52, 53], and dynamic with powertrain model [54].

Dynamic powertrain motion model [54] is defined for middle-size vehicles with front-wheel drive, equipped with an automatic transmission, four-wheel independent suspension, and open differential. It supports three standard engine models - gas, diesel and electric.

The software package is designed for high-speed inference to simulate scenarios faster than in real-time. Therefore it is implemented in low-level Rust programming language while providing APIs to Python programming language. To limit computational load, it supports only primary object perception, providing ground truth data about objects' physical properties. Perception may be visualized in the form of bounding boxes for vehicles and lines for road geometry (Fig. 2.5). That wireframe visualization does not provide scene renders with reach images and textures as other simulations such as [55, 56]. Detail scene rendering would cause significant computational load and not provide any advantage since the proposed methods work on already detected objects represented in high-level abstraction. Such accelerated software is adequate for online RL, which requires generating new experience data multiple times during the training process.

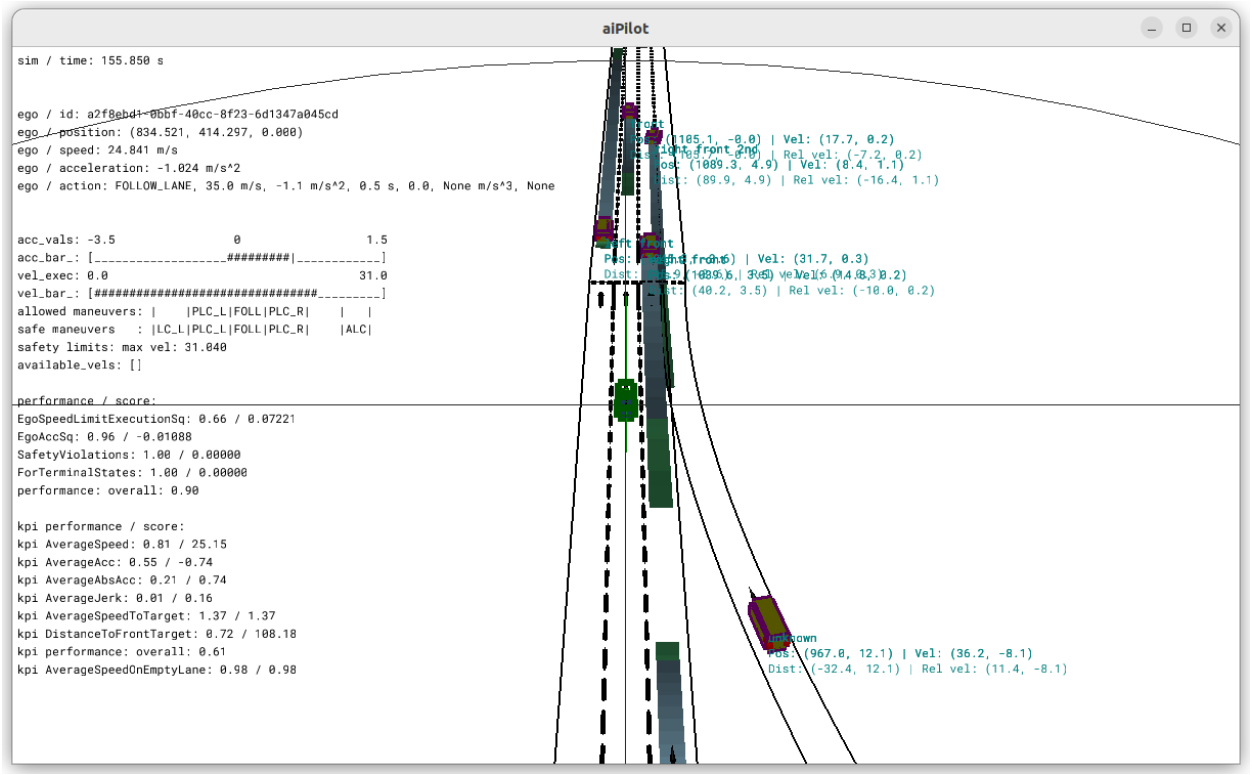


Figure 2.5: Simulatation in Simteract software. The green vehicle in the center is controlled by an RL agent, the rest of the vehicles are driven by a built-in behavior model. Polygons behind vehicles present safety distances with respect to the host car.

2.5 Safety Consideration

Applying Machine Learning in vehicle control increases the probability of undefined, probably dangerous, situations. Therefore it is essential to provide some deterministic rules which guarantee the safety of the motion planning system. For this application, we selected Responsible Sensitive Safety (RSS) [57] frame-

work, which rules are applied to behavior planning and trajectory generation module. RSS is inspired by the human common sense of safe driving. Its general rule is not to cause an accident and avoid crashes caused by others but only when avoiding it will not lead to another one. That approach is formalized in the 5 rules described below. The first two are explicitly used during experiments, while all of them state the foundation of safety consideration of the entire project.

'Do not hit the car in front of you (longitudinal distance)' assumes leaving enough space for the front vehicle in order to have enough time to react if this car suddenly brakes. The Equation (2.19) calculates minimum safety distance to front vehicle w.r.t. response time ρ, regarding the velocity of rear (v_r) and front car (v_f). It applies to situations when the front target is braking with maximum acceleration $b_{\max}$, and the rear car accelerates by at most $a_{\max}$ during response time and then brakes with at least $b_{\min} m/s^2$.

$$
\begin{aligned}
d_{\text{lon\_min}} &= \left[v_r \rho + 0.5 a_{\max} \rho^2 + \frac{(v_r + \rho a_{\max})^2}{2 b_{\min}} - \frac{v_f^2}{2 b_{\max}} \right] \\
&\text{where} \\
&v_r, v_f \text{ velocity of rear and front vehicle} \\
&\rho \text{ response time} \\
&b_{\max} \text{ maximum possible braking acceleration for front vehicle} \\
&b_{\min} \text{ braking acceleration for rear vehicle} \\
&a_{\max} \text{ maximum possible acceleration}
\end{aligned}
\tag{2.19}
$$

'Do not cut in recklessly (lateral distance)' formalizes the safe lateral distance between two vehicles $(c1, c2)$ driving with lateral velocities v_1, v_2 w.r.t. response time ρ, $b_{\min}^{\text{lat}}$, $a_{\min}^{\text{lat}}$, μ if during the time interval $[0, \rho]$ two vehicles approaches with lateral acceleration $a_{\min}^{\text{lat}}$, and after that they apply lateral braking of b_{min}^{lat} until they reach zero lateral velocity, then the final lateral distance between them will be at least μ.

$$
d_{\text{lat\_min}} = \mu + \left[\frac{(v_1 + v_{1,\rho})}{2} \rho + \frac{v_{1,\rho}^2}{2 b_{\min}^{\text{lat}}} - \left(\frac{v_2 + v_{1,\rho}}{2} + \frac{v_{2,\rho}^2}{2 b_{\min}^{\text{lat}}} \right) \right] \tag{2.20}
$$

'Right of way is given, not taken' - such a rule cautions traffic participants who do not adhere to the right-of-way rules. The car should not admit to crashing in the case when others violate traffic rules.

'Be cautious in areas with limited visibility' - this statement assumes that from the region, which is not visible by sensors, other traffic participants may drive from.

'If the vehicle can avoid a crash without causing another one, it must' - this rule accepts breaking the traffic rules if an upcoming accident may be averted but only if provided that this will not cause a different collision.

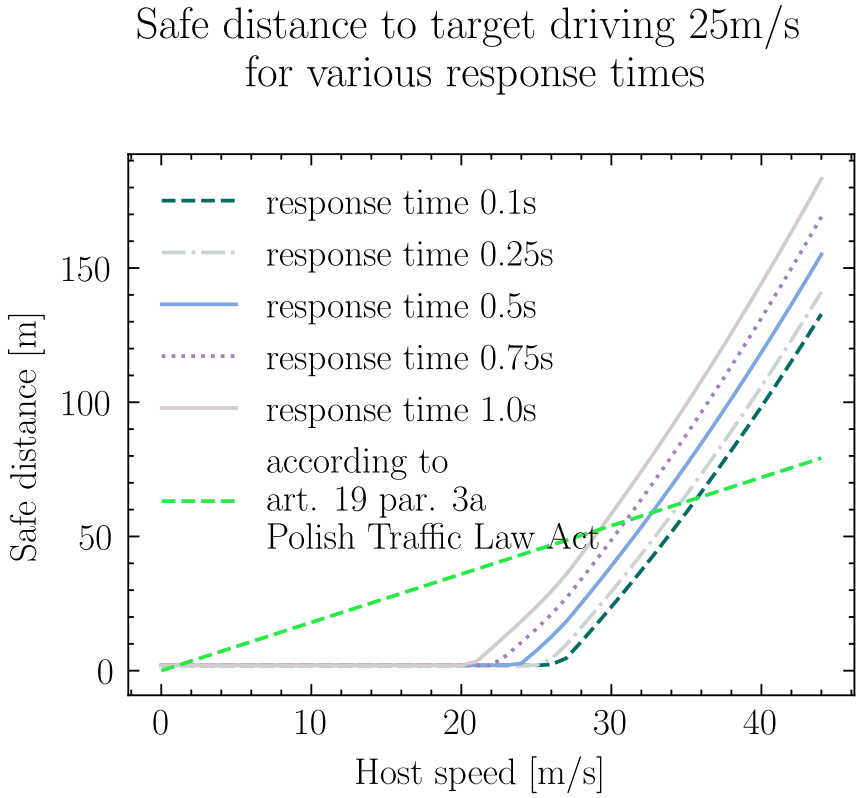


Figure 2.6: Safe distance to target driving 25m/s for various values of host response times parameter ρ. Comparison to the safe distance required by Polish Traffic Law act art.19, par. 3a - which is expressed in meters and specified as not less than half of the number specifying the speed of the vehicle in which the driver is moving, expressed in kilometers per hour.

Chapter 3

Theoretical Introduction to Learning Algorithms

The present dissertation is grounded on the utilization of deep learning techniques that involve the training of Artificial Neural Networks (ANN) through the use of Reinforcement Learning and associated algorithms. This Section aims to introduce the theoretical aspects of Reinforcement Learning (RL), providing an intuitive comprehension of its fundamental principles and demonstrating the state-of-the-art algorithms applied in the experiments.

The conventional background of RL is expanded by introducing the concept of Offline Reinforcement Learning, which encompasses the utilization of real-world experience in an RL context. Furthermore, the issue of the distributional shift that arises between simulation and the real world is discussed. The problem, commonly referred to as the Sim2Real gap in the literature, is presented along with the techniques employed to alleviate it.

3.1 Reinforcement Learning

Reinforcement learning is a machine learning technique that regards learning an artificial agent to solve a problem represented by some environment [40]. The problem typically is defined as a Markov Decision Process (MDP) [58]. MDP is a discrete-time stochastic control process that was originally designed as a mathematical framework for solving decision-making problems. It formulates a tuple (S, A, P, R, γ) in which S is a set of possible states and A is a set of available actions. The symbol P represents the function $P(s_{t+1}|s_t, u_t)$ which defines a probabilistic transition function of transforming the environment from state s_t while executing actions u_t to next state s_{t+1}. The next part of tuple $R(s_t, u_t, s_{t+1})$ is an immediate reward that is granted for selecting action u_t, which transitioned the environment from state s_t to state s_{t+1}. The γ is a discount factor $\gamma \in [0, 1]$. The state and action could be expressed with discrete or continuous values.

A major part of MDP is also a parameterized policy $\pi(s_t, \theta) \rightarrow u_t$ or $\pi(u_t|\theta, s_t)$, which returns action based on the given state. The policy is owned by the agent, which could be defined as a part of a learning algorithm. The agent can interact with the environment, by selecting actions and gathering knowledge necessary for solving the problem stated by MDP (Fig. 3.1).

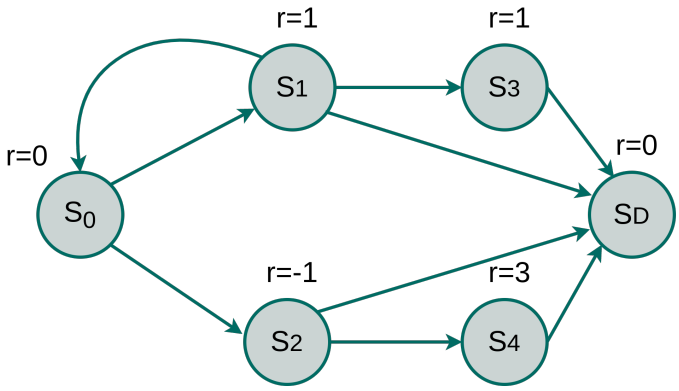


Figure 3.1: Markov decision process

At each time step t, the environment provides the agent an observation of its own state $s_t \in S$ and a set of available actions $u \in A$. The agent chooses an action $u_t \in A$ based on the given state s_t, which is executed by the environment that causes the change of the environment's state s_{t+1}. The transformation function $P(s_{t+1}|s_t, u_t)$ calculates the new state s_{t+1} based on the previous state s_t and picked action u_t. Additionally, the reward function $R(s_t, u_t, s_{t+1}) \rightarrow r_t$ calculates the immediate reward for the agent. The agent can interact with the environment until reaching a terminal state $s_d \in S$ of the environment. The set of consecutive states from the initial state s_i to the terminal state s_d is called an episode (Fig. 3.2). All consecutive states which are accompanied by selected actions and given rewards, compose a trajectory of length T:

$$\tau = \{(s_0, u_0, r_0, s_1), (s_1, u_1, r_1, s_2), \ldots, (s_{T-1}, u_{T-1}, r_{T-1}, s_d)\} \tag{3.1}$$

Based on the interactions, the agent learns how to select actions in order to maximize the cumulative sum of the reward. The agent's objective function is:

$$J(\pi_\theta) = \mathbb{E}_{\tau \sim p(\tau;\pi_\theta)} \left[\sum_{t=0}^{T-1} \gamma^t r_t \right]$$

where:

$\mathbb{E}_{\tau \sim p(\tau;\pi_\theta)}$ denotes the expectation over trajectories τ generated by following policy π_θ.

$p(\tau;\pi_\theta)$ is the probability distribution of trajectories τ under policy π_θ

T is a number of time steps in an episode

r_t is the reward obtained at time step t

γ is the discount factor$\gamma \in [0, 1]$. (3.2)

The learning process is described by the various reinforcement learning algorithms that define how to collect the experience and when and how to update parameters θ of the policy π_θ. The outcome of RL algorithms is the optimal policy π^*_θ.

The discount factor γ describes how the future return is relevant to the current state and is defined as

$$G_t = r_t + \gamma r_t + \gamma^2 r_{t+1} + \ldots + \gamma^{T-t} r_{T-t} = \sum_{k=0}^{T-1} \gamma^k r_{t+k} \tag{3.3}$$

Defaults γ equals less than one because the forthcoming rewards are less relevant than the immediate ones. The γ defines the length of the horizon to be considered by a trained policy and is one of a hyperparameter of all RL algorithms. If γ equals 0, the policy will act short-sighted, paying attention only to the closest reward. As the γ value approaches 1, the horizon extends, and the policy plans action further into the future.

A more generalized form of the MDP is the Partially Observable Markov Decision Process (POMDP) [59]. In the environment defined as POMDP (3.4), the agent still selects actions that transform the state of the environment to maximize the sum of collected rewards. However in this setup, an agent is not able to directly observe the entire state of the environment, but it is limited to perceive only the limited observation of the state.

The agent uses its observations to form a belief about the current state of the system. This belief is represented as a probability distribution that includes all possible states. To solve the problem of POMDP, a policy is developed that determines the best course of action in each belief state. It is worth noting that belief states are continuous, which means there are an infinite number of possible states. This makes solving POMDPs more challenging than MDPs.

The versatility of the POMDP framework makes it suitable for modeling various real-world problems that involve sequential decision-making. Applications range from robot navigation and machine maintenance to general planning under uncertainty.

$$
\begin{aligned}
&P = (S, A, P, R, \Omega, O, \gamma) \\
&\text{where:} \\
&S \in \mathbb{R} \text{ - state space,} \\
&A = \{u_1, u_2, \ldots, u_n\} \text{ - set of actions,} \\
&P(s_{t+1}|s_t, u_t) \text{ - probability of transition from state s on action } u \text{ in time t,} \\
&\text{to next state } s_{t+1} \\
&R = R(s_t, u_t, s_{t+1}) \text{ - reward function} \\
&\Omega \in \mathbb{R} \text{- observation space,} \\
&O \text{ - a set of observation probabilities } O(o|s_{t+1}, u) \text{ conditioned on the reached} \\
&\text{state and the taken action} \\
&\gamma \in [0, 1] \text{ discount factor}
\end{aligned}
\tag{3.4}
$$

3.1.1 The Intuition Behind Reinforcement Learning

Reinforcement Learning originates from the concept of conditioning in psychology. The term conditioning describes a psychological phenomenon where the learning process links an unconnected stimulus with the response. The actions that cause pain or discomfort are impaired, and actions that lead to pleasure are positively reinforced. To discover how to behave optimally, the learning entity explores the environment, investigating which actions are profitable in the given states. During the learning, it compromises between

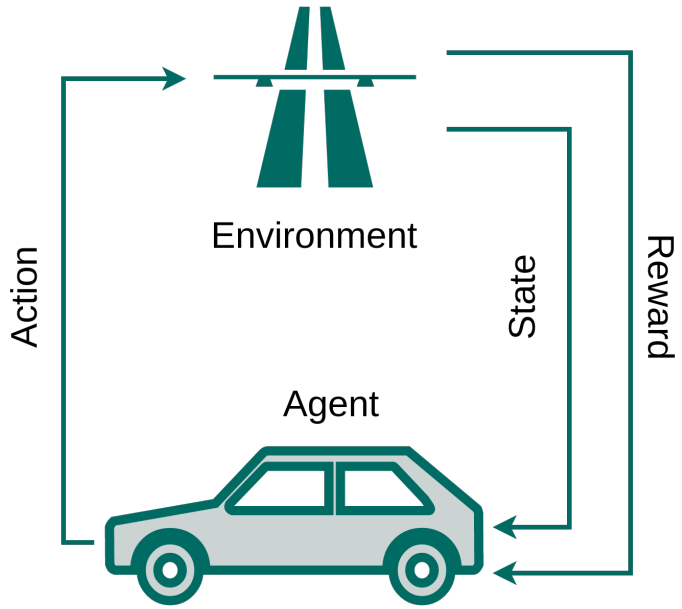


Figure 3.2: Reinforcement learning framework - the agent interacts with the environment by picking the action from a set of available actions. The environment executes the selected action, changes its own state, and rewards the agent. The agent's goal is to maximize the sum of rewards until the environment reaches a terminal state.

picking beneficial action and selecting the action whose results are already unknown. In this case, it exposes itself to the lower reward. However, it also gets the opportunity to discover regions with even higher returns. This problem is well-known in literature as an exploration-exploitation trade-off. Conclusively, the goal of the training algorithm is to train the policy to behave optimally. In contrast to other optimization methods, it often leads to situations where policy is able to sacrifice a high immediate reward for much higher compensation in the future.

3.1.2 Value-Based Methods

The following sections (3.1.3-3.1.7) present the value-based methods of solving the problem defined as MDP. These methods formulate policy $\pi(s_t)$ based on the estimation of the sum of rewards that can be accumulated by the policy starting from the given state of the MDP. Concurrently with the value-based methods, there also exists the policy-based approach, which will be described in further sections (3.1.8 - 3.1.11).

3.1.3 Dynamic Programming

The problem formulated as MDP (Sec. 3.1) may be solved using dynamic programming [60]. Dynamic programming breaks down the complete problem into simpler subproblems and solves each individually. If the already solved problem is met, dynamic programming doesn't work it out again but uses the previously calculated results. In the case of MDP, the main task can be broken down into sub-tasks that estimate the maximum reward that can be achieved starting from each of MDP's states.

To solve the MDP in a dynamic programming manner, it is helpful to use the Bellman Equation:

$$V(s_t) = \max_{u_t}(R(s_t, u_t) + \gamma V(s_{t+1})) \tag{3.5}$$

On the left side of the Equation stands the state value function $V(s_t)$. It represents the expected cumulative reward that can be achieved when starting from a particular state s_t in MDP and following a given policy. The $V(s_t)$ equals the sum of the immediate reward and the discounted value of the next state. It is the sum of two factors on the right side of the equation. First of which is the value of the highest possible reward, which could be collected by picking the most profitable action u_t in the given state s_t. The second one is the result of the multiplication of the discount factor and state value of the next state $V(s_{t+1})$.

$$V(s_t) = \max_{u_t}(R(s_t, u_t,) + \gamma \sum_{s_t} P(s_{t+1}|s_t, u_t)V(s_{t+1})) \tag{3.6}$$

Equation 3.7 presents the evolved Bellman equation and the relation between γ and the relevance of the future rewards. The τ shows the time horizon related to a discount factor. If $\gamma = 1$ corresponds to $\tau = \infty$, and any rewards that are much more than τ time steps in the future are exponentially suppressed.

$$V(s_t) = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + ...\gamma^n r_{t+n} = \sum_{n=0}^{\infty} \gamma^n r_{t+n} = \sum_{\Delta t=0}^{\infty} e^{-\Delta t/\tau} r_{t+\Delta t} \tag{3.7}$$

3.1.4 State-Value Function

The Bellman equation (3.5, 3.6) is used to define the state-value function. The state-value function $V^{\pi_\theta}(s_t)$ is the expected cumulative discounted reward that can be achieved, starting in the state s_t and acts accordingly to the output of policy π.

$$V^{\pi}(s_t) = \mathbb{E}\left[\sum_{t=0}^{T-1} \gamma^t r_t]\right] \forall s \in S \tag{3.8}$$

If the $V^{\pi}(s_t)$ equals the highest possible value in MDP compared to other value functions for all states, it means that it is the optimal state-value function (equation 3.9).

$$V^*(s_t) = \max_{\pi} V^{\pi}(s_t) \qquad \forall s \in S \tag{3.9}$$

By knowing the optimal value function, we also know the optimal policy π^* (equation 3.10).

$$\pi^* = \arg\max_{\pi} V^{\pi}(s_t) \qquad \forall s \in S \tag{3.10}$$

3.1.5 Q-Value

The state-value function $V^{\pi}(s_t)$ returns only one value for each state which describes the expected reward. This distinguished it from the action-value function $Q^{\pi}(s_t, u_t)$, which returns the expected cumulative discounted reward for taking action u_t in the state s_t and continuing acting according to the policy π.

$$V^*(s_t) = \max_{u_t} Q^*(s_t, u_t) \qquad \forall s \in S \qquad u \in A \tag{3.11}$$

$$Q^\pi(s_t, u_t) = \mathbb{E}\left[\sum_{t=0}^{T-1} \gamma^t r_t]\right] \qquad \forall s \in S \qquad u \in A \tag{3.12}$$

The optimal policy π^*, which is based on the action-value function selects the action with the highest return in a set of available actions A for all consecutive steps $s_t, s_{t+1}, \ldots, s_d$.

$$\pi^*(s_t) = \arg\max_{u} Q^*(s_t, u_t) \qquad \forall s \in S \qquad u \in A \tag{3.13}$$

3.1.6 Temporal Difference

To obtain policy π^* based on value methods, one has to estimate the value function $V(s_t)$ or action-value function $Q(s_t, u_t)$. This estimation could be achieved by using Monte Carlo methods [61] or Dynamic Programming [60]. However, the Reinforcement Learning methodology introduced a novel method called temporal difference (TD).

TD estimates the $V^\pi(s_t)$ by interacting with the environment and improving current estimation every time it picks action and makes the transition in the environment by observing a new state and a corresponding reward. Referring to the Equation (3.14), it improves its own estimation based on the difference between the value estimation of the state s_t and actual immediate reward r_t summed with an estimation of the next state value $V(s_{t+1})$ (algorithm details: Algorithm 1). Because the updates based on the current estimation are often called bootstrapping methods, and the difference itself is called a TD error. The TD error is commonly used in various reinforcement learning algorithms, and it is considered to be essential in the RL domain.

$$V(s_t) \leftarrow V(s_t) + \alpha \underbrace{\left[\overbrace{r_{t+1} + \gamma V(s_{t+1})}^{\text{TD-target}} - V(s_t)\right]}_{\text{TD-error}} \tag{3.14}$$

Algorithm 1 Temporal Difference algorithm for estimating value function $V^\pi(s)$.

Require: policy π

while Loop for N episodes **do**

while Loop for steps until reaching terminal state **do**

$u_t \leftarrow$ action picked by π in s_t

Take action u_t, observe r_{t+1}, s_{t+1}

$V(s_t) \leftarrow V(s_t) + \alpha\left[r_t + \gamma V(s_{t+1}) - V(s_t)\right]$

$s_t \leftarrow s_{t+1}$

end while

end while

The advantage of the Temporal Difference method is that it does not require knowledge of the environment model (as the Monte Carlo method does) because it updates the estimation of $V(s_t)$ in each step of the

episode. Therefore it also does not need to know about the reward and next-state probability distributions. Related to the update frequency, there is also an advantage that the learning algorithm does not need the entire episode to learn. This could be crucial in the case of environments with very long episodes.

3.1.7 Q Learning

One of the algorithms that rely on TD is Q-learning [62]. The algorithm is the value-based off-policy method (Sec. 3.1.12) that aims to approximate the action-value function $Q(s_t, u_t)$. For interaction with the environment, it uses behavior policy, for example, ϵ-greedy policy. It means that it takes random action with a probability of ϵ and the action from greedy policy ($\max_{u_t} Q(s_t, u_t)$) in $1 - \epsilon$ probability. The ϵ-greedy policy is specific for Q-learning and is used to solve the exploration-exploitation problem. During the training, as the policy quality improves, the value of ϵ may decrease.

The update of each of the $Q(s_t, u_t)$ takes place accordingly to

$$Q(s_t, u_t) \leftarrow Q(s_t, u_t) + \alpha \left[r_{t+1} + \gamma \max_{u_t} Q(s_{t+1}, u_t) - Q(s_t, u_t) \right] \tag{3.15}$$

where α is the learning rate. The learning algorithm is defined as (Algorithm 2):

Algorithm 2 Q-learning algorithm

Require: $\alpha \in (0, 1]$, $\epsilon \in [0, 1)$

while Loop for N episodes **do**

while Loop for steps until reaching terminal state **do**

Choose u_t on s_t using for example ϵ - greedy policy

Take action u_t, observe r_{t+1}, s_{t+1}

$Q(s_t, u_t) \leftarrow Q(s_t, u_t) + \alpha \left[r_t + \gamma \max_{u_t} Q(s_{t+1}, u_{t+1}) - Q(s_t, u_t) \right]$

$s_t \leftarrow s_{t+1}$

end while

end while

Convergence results of the Q-learning algorithm were shown in [63, 64]. The convergence is provided by the value iteration operator which ensures that the Q-values for all state and action pairs update gradually to the true optimal Q-values of the underlying MDP. The work [64] proved that the random iterative process $\Delta_{n+1}(x) = (1 - \alpha_n(x))\Delta_n(x) + \beta_n(x)F_n(x)$ convergence with probability one under the assumption that

- the state space is finite $S = \{s_1, s_2, \ldots, s_n\}$,
- $\sum_n \alpha_n(x) = \infty; \sum_n \alpha_n^2(x) < \infty; \sum_n \beta_n(x) = \infty; \sum_n \beta_n^2(x) < \infty$, and $\mathbb{E}\{\beta_n(x)|P_n\} < \mathbb{E}\alpha_n(x)|P_n$ uniformly w.p.1.
- $||\mathbb{E}\{F_n(x)|P_n\}||w \leqslant \gamma||\Delta n||w$, where $\gamma \in (0, 1)$
- $Var\{F_n(x)|P_n\} \leqslant C(1 + ||\Delta_n||w)^2$ where C is constant.

where $P_n = \Delta_n, \Delta_{n-1}, \ldots, F_{n-1}, \ldots, \alpha_{n-1}, \ldots, \beta_{n-1}, \ldots$ stands for the past at step n. $F_n(x), \alpha_n(x)$, and $\beta_n(x)$ are allowed to depend on the past insofar as the above conditions remain valid. The notation $|| \cdot ||w$ refers to some weighted maximum norm.

The Q-learning algorithm given by (3.15) converge to the optimal $Q^*(s_t, u_t)$ values if

1. the state space is finite $S = \{s_1, s_2, \ldots, s_n\}$,
2. $\sum_t \alpha_t(s_t, u_t) = \infty$ and $\sum_t \alpha_t^2(s_t, u_t) \leqslant \infty$ uniformly w.p.1,
3. $Var\{c_s(u_t)\}$ is bounded
4. If, $\gamma = 1$, all policies lead to a cost free terminal state w.p.1.

3.1.8 Policy Gradient Methods

As distinct from the value-based methods described in previous sections, there are also methods that learn direct parameterized policy (Sec. 3.1.10-3.1.14). In contrast to the value-based methods, they do not rely on the estimation of state-action value but tune parameters θ of the policy function that selects the action. Although the algorithms do not need value function for inference with the environment, they could employ it to learn the function parameters (actor-critic methods Sec. 3.1.11).

The policy function $\pi(u|s, \theta)$ could be described as a probability of selecting the action u_t at time t in a state s_t with vector parameter θ:

$$\pi(u|s, \theta) = P\{A_t = u|S_t = s, \theta_t = \theta\} \tag{3.16}$$

Learning of such policy function relies on the tuning of the parameters θ based on the gradient of some objective function $J(\theta)$ with respect to the policy parameters. The objective function usually considers the performance of the trained policy, calculated on the basis of collected rewards. The parameters θ are updated according to gradient ascent with α-size steps:

$$\theta_{t+1} = \theta_t + \alpha \overbrace{\nabla J(\theta_t)}^{\in \mathbb{R}^{d'}} \tag{3.17}$$

The methods that directly improve policy with gradient ascent are called policy gradient methods. If they consider also the value function in learning, they are called actor-critic methods (Sec. 3.1.11). This name suggests that the actor part regards the policy function and the critic refers to the value function as one is responsible for acting and the latter for performance evaluation.

The policy gradient methods require policy to be differentiable with respect to its parameters θ. The partial of derivatives of $\pi(u|s, \theta)$ with respect to the θ should exists and be finite for all $s \in S$, $u \in A$, $\theta \in \mathbb{R}^{d'}$. Additionally, it is required from the policy to be stochastic in order to assure the possibility of an environment exploration, which is necessary for the correct operation of the algorithm.

3.1.9 Policy Definition

Consider the environment characterized by a discrete action space where the agent could choose $u \in A$ where $A = \{u_j, u_{j+1}, \dots u_N\}$ in each state $s \in S$. In this case, the parameterized function would return parameterized numerical preferences $h(s, u, \theta) \in \mathbb{R}$ for each state-action pair. Such numerical preferences could be converted to a probability distribution with the use of the softmax function:

$$\pi(u_j|s, \theta) = \frac{e^{h(s,u_j,\theta)}}{\sum_{u=A} e^{h(s,u,\theta)}} \tag{3.18}$$

To calculate the action preferences we could use any parameterized functions. This work focuses on utilizing Artificial Neural Networks (ANN) as a parameterized function where the vector θ contains all ANN's trainable weights. In that case the $h(s, u, \theta)$ is represented as $f^{\text{ANN}}(s, \theta) \rightarrow \mathbb{R}^N$.

In the case of the environment with continuous action space with an infinite number of actions, policy-based methods provide effective solutions. These methods avoid directly computing probabilities for each individual action. Instead, they focus on learning the statistical characteristics of the probability distribution. For instance, in scenarios where the action set comprises real numbers, actions can be selected from a normal (Gaussian) distribution. This approach simplifies the representation and allows policy-based methods to handle complex action spaces efficiently [40].

The probability density function $f_{\text{PDF}}(x|\mu, \sigma)$ for the normal distribution for the random variable x is defined as

$$f_{\text{PDF}}(x|\mu, \sigma) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \tag{3.19}$$

where μ is the mean (or expected value) and σ is the standard deviation of the normal distribution. To create a policy representation, we can define the policy as a normal probability density function over a real-valued scalar action as

$$\pi(u|s, \theta) = \frac{1}{\sqrt{2\pi}\sigma(s, \theta)} e^{-\frac{(u-\mu(s,\theta))^2}{2\sigma(s,\theta)^2}} \tag{3.20}$$

where $\mu : \mathbb{S} \times \mathbb{R}^{d'} \rightarrow \mathbb{R}$ and $\sigma : \mathbb{S} \times \mathbb{R}^{d'} \rightarrow \mathbb{R}^+$.

The mean and standard deviation of distribution are determined by parametric function approximators that rely on the state as input $f^{\text{ANN}}(s, \theta) \rightarrow \mathbb{R}^2$.

3.1.10 REINFORCE Algorithm

The seminal policy-gradient algorithm is called REINFORCE [65]. This algorithm starts with collecting a trajectory τ of experience (3.1). The trajectory consists of n consecutive tuples of state, action, rewards, and next states (s_t, u_t, r_t, s_{t+1}). The length of the trajectory (n) is unrestricted and could enclose part of an episode or even several episodes. After collecting the trajectory, the algorithm calculates the return of the trajectory $G(\tau, t)$ for each time step t in T length trajectory (eq. 3.21).

$$G(\tau, t) = \sum_{k=t}^{T-1} \gamma^{k-t} r_k \tag{3.21}$$

The algorithm aims at tuning the parameters vector θ in such a way that the policy $\pi(u|s,\theta)$ parameterized by this vector maximizes the expected return $U(\theta,\tau)$. The expected return is expressed as a function of trajectory τ and parameters θ as a sum of probabilities of getting specified in trajectory rewards (eq. 3.22).

$$U(\theta,\tau) = \sum_{\tau} P(\tau,\theta)R(\tau) \tag{3.22}$$

To estimate the expected return, the algorithm calculates the weighted average of the previously computed returns of each trajectory weighted by its probability. The probabilities are taken from the policy function $\pi(u|s,\theta)$ (3.18, 3.20) and describe how much possible is to select action u_t in a given state s_t.

The probabilities are differentiable with respect to the θ parameters, therefore they could be used for policy updates, which are carried out with the step of gradient ascent (3.24).

However, it would be impossible to calculate the gradient directly on trajectories, because the model of the transition of the environment's states is unknown. Therefore in this case the gradient should be estimated as a derivative of the logarithm of the probability of selecting the action with respect to the parameters θ. The probability of selecting an action by the behavior policy which returned a higher reward is increased and the probability of taking unprofitable action is decreased.

$$\nabla_\theta U(\theta) \leftarrow \sum_{t=0}^{T-1} \nabla_\theta \ln\pi(u_t|s_t,\theta)G(\tau,t) \tag{3.23}$$

$$\begin{aligned} &\theta \leftarrow \theta + \alpha\nabla_\theta U(\theta) \\ &\alpha \text{ is step size parameter of gradient update} \end{aligned} \tag{3.24}$$

Algorithm 3 REINFORCE algorithm

Require: differentiable policy parameterization $\pi(u|s,\theta)$, α in (0,1]

Initialize the policy parameters θ randomly

while until converge **do**

Collect T steps trajectory

$\tau = ((s_0,u_0,r_0,s_1),\ldots,s_{T-1},u_{T-1},r_{T-1},s_T)$ with policy $\pi(u|s,\theta)$

Calculate the return of a trajectory τ

where $G(\tau,t)$ is the expected return for the transition t

$G(\tau,t) = \sum_{k=t}^{T-1} \gamma^{k-t} r_k$

Estimate gradient $\nabla_\theta U(\theta)$ using the trajectory τ

$\nabla_\theta U(\theta) \leftarrow \sum_{t=0}^{T-1} \nabla_\theta \ln\pi(u_t|s_t,\theta)G(\tau,t)$

Update the weights θ of the policy

$\theta \leftarrow \theta + \alpha\nabla_\theta U(\theta)$

end while

Although the REINFORCE algorithm fulfills its purpose as is able to converge under strict conditions, it is characterized by noisy gradients and high variance. These cause instability in the learning process and threaten to get stuck in a local minimum, far from the optimal solution. These issues are caused by

the method of collecting the trajectory by the algorithm, which depends on random sampling from action distribution. This causes much deviation between collected trajectories which results in great variability of logarithms of probability selecting actions by the policy $\ln\pi(u_t|s_t,\theta)$ as well as in varying cumulative reward $G(\tau,t)$.

3.1.11 Actor-Critic Methods

The disadvantages of REINFORCE algorithm, which were described at the end of Section 3.1.10, led to the formulation of the actor-critic methods [40, 66]. In order to reduce the aforementioned variance these methods suggest subtracting the cumulative reward by some baseline $b(s_t)$ in the REINFORCE gradient calculation 3.23. This subtraction is included in a new objective function defined as:

$$\nabla_\theta U(\theta) \leftarrow \sum_{t=0}^{T-1} \nabla_\theta \ln\pi(u_t|s_t,\theta)(G(\tau,t) - b(s_t)) \tag{3.25}$$

Subtracting expected reward with baseline results in smaller values of gradients, therefore a more stable learning process.

The expected return of the trajectory G_t could be expressed as

$$G(\tau,t) = \mathbb{E}\left[\sum_{t=0}^{T-1} \gamma^t r_t]\right] \quad \forall s \in S \quad u \in A \tag{3.26}$$

what is exactly the same as in action-value function $Q(s,u)$ (3.12). Therefore the Q-value $Q(s_t,u_t)$ or state value $V(s_t)$ could be used as a baseline in the actor-critic method. Actually, the name of this method refers to the fact that it employs two parameterized functions - actor and critic. The actor function selects the action and it is optimized with the policy-gradient method (Sec. 3.1.10). And critic is used during the learning to estimate the expected return and is updated based on TD-error (Sec. 3.1.6). The overall algorithm is presented in the Alg. 4.

3.1.12 On/Off Policy

Reinforcement Learning algorithms could be categorized as on-policy or off-policy methods. The on-policy method updates its policy $\pi(u|s,\theta)$ using the samples (s_t,u_t,r_t,s_{t+1}) generated by the exact same policy. For any new update, fresh samples are needed, and the old ones are forgotten. An example of such a method is the REINFORCE algorithm (Sec. 3.1.10). Generally, most policy-based methods belong to the group of on-policy methods.

On the contrary, the off-policy method is able to update the policy $\pi(u|s,\theta)$ using the samples generated by some other policies. In the case of Q-learning (Sec. 3.1.7) the behavior policy collects samples (ex. ϵ-greedy policy), however, the main policy is updated. Q-learning, like the rest of the value-based methods, could be considered an off-policy algorithm.

The off-policy algorithms require some replay buffer that stores transition samples generated during the learning and serves a batch of samples during each policy update. It has a finite capacity, therefore some part of the experience is forgotten when the replay buffer which stores the trajectories τ runs out of space. Naive

Algorithm 4 Q Actor Critic

Initialize parameters of policy function θ and action-value function w

Initialize learning rates $\alpha_\theta > 0, \alpha_w > 0$

sample action u_t from policy $\pi(u_t|s_t, \theta)$

for $t = 1...T$ **do**

Sample reward $r_t \leftarrow R(s_t, u_t)$ and next state $s_{t+1} \leftarrow P(s_{t+1}|s, u)$

Then sample the next action $u_{t+1} \leftarrow \pi(u_t|s_t, \theta)$

Update the policy parameters:

$$\theta \leftarrow \theta + \alpha_\theta Q_w(s_t, u_t)\nabla_\theta ln\pi(u_t|s_t, \theta)$$

Compute the td-error for action-value at time t:

$$\varsigma = r_t + \gamma Q_w(s_{t+1}, u_{t+1}) - Q_w(s_t, u_t)$$

and use it to update the parameters of Q function:

$$w \leftarrow w + \alpha_w \varsigma \nabla_w Q_w(s_t, u_t)$$

Move to $u_t \leftarrow u_{t+1}$ and $s_t \leftarrow s_{t+1}$

end for

replay buffer samples randomly and overwrites the oldest data. However, the authors of the work Prioritized Experience Replay [67] presented the method which prioritizes samples with weights based on the TD-error (Sec. 3.1.6) and manages them accordingly. The prioritization helps algorithms to update policy parameters with samples that are most problematic, where the TD-error is higher so the estimation of the Q-value is worse. Updating function parameters more frequently with such samples improves their estimation, which increases the learning speed in relation to the random sampling method.

3.1.13 Deep Reinforcement Learning

Deep Reinforcement Learning broadens classical RL with the utilization of Artificial Neural Networks (ANN). In this case ANN serves as approximation of policy function $\pi(u|s, \theta)$, action-value function $Q(s, u, \theta)$ or state-value function $V(s, \theta)$. These functions are parameterized by the vector of parameters θ which refers to the vector of trainable weights of ANNs.

ANNs are mostly utilized in the case of environments characterized by large state space S or action space A. It could outperform linear functions or tabular methods due to their non-linearity. Additionally, ANNs are known for their generalization capacity, which means that they can act correctly on inputs that they have never seen before, assuming that they met similar states during the learning.

The introduction of ANNs in Reinforcement Learning (RL) has led to the emergence of new learning algorithms, marking a new era in RL. This chapter presents one such algorithm, Proximal Policy Optimization (PPO)[68] which is relied upon by most of the discussed work.

3.1.14 Proximal Policy Optimization

Proximal Policy Optimization (PPO) [68] is an on-policy method and belongs to the family of policy gradient algorithms (Sec. 3.1.8). Compared to earlier RL algorithms, PPO stands out with two improvements. The

first regards using trust region for updating parameters instead of the linear search method. The second one considers the method of calculating the advantage function which is a part of PPO's objective.

The trust region technique relies on determining the maximum size of the step during the parameters update in the first place and then the direction of that step (3.17). The maximum step defines an area in parameter space around the current best solution which cannot be left by stepping in gradient descent update. This protects the new policy from being too far from the old one after the parameters update which increases the learning stability.

The authors of PPO proposed two ways to define the trust regions and how to incorporate them into the objective function. As presented in Section 3.1.10, to update the policy π_θ the gradient of the objective function with respect to the θ must be estimated (3.23). The general form of gradient estimator could be formalized as:

$$\hat{g} = \hat{\mathbb{E}}_t \left[\nabla_\theta \ln \pi(u_t|s_t,\theta) \hat{A}_t \right] \tag{3.27}$$

where $\hat{A}_t$ is the estimator of the advantage function at time t (3.31). The expectation $\hat{\mathbb{E}}_t[\ldots]$ stands for the empirical average over a finite batch of samples.

Accordingly, to the general form, the PPO defines its own base objective function (3.28). This function is formulated based on the seminal work [69] which introduced the concept of trust regions in RL optimization. The innovation of PPO, however, relies on simplifying the predecessor method with the utilization of clipping gradients. To calculate the gradient, the objective function (3.28) should be differentiable with respect to the parameters θ.

$$J^{\mathrm{CPI}}(\theta) = \hat{\mathbb{E}}_t \left[\frac{\pi(u_t|s_t,\theta)}{\pi(u_t|s_t,\theta_{\mathrm{old}})} \hat{A}_t \right] \tag{3.28}$$

To simplify the notation lets denote the probability ratio $r_t(\theta) = \frac{\pi(u_t|s_t,\theta)}{\pi(u_t|s_t,\theta_{\mathrm{old}})}$. Notation θ_{old} states for θ parameters actual for previous optimization step.

The algorithm introduced a clipping function

$$\mathrm{clip}(r_t(\theta),\epsilon) = \begin{cases} 1-\epsilon, & \text{if } r_t(\theta) < 1-\epsilon \\ 1+\epsilon, & \text{if } r_t(\theta) > 1+\epsilon \\ r_t(\theta) & \text{otherwise} \end{cases} \tag{3.29}$$

with new algorithm's hyperparameter ϵ which typically equals 0.2. Using clipping function $\mathrm{clip}(r_t(\theta),\epsilon)$ the new objective function is formulated as

$$J^{\mathrm{CLIP}}(\theta) = \hat{\mathbb{E}}_t \left[\min \left(r_t(\theta)\hat{A}_t, \mathrm{clip}(r_t(\theta),\epsilon)\hat{A}_t \right) \right] \tag{3.30}$$

The presented clipped objective takes the minimum value of the unclipped objective (eq. 3.28) and the clipped value to the range of $(1-\epsilon, 1+\epsilon)$. The final objective assumes the pessimistic bound on the unclipped and clipped objective. Therefore the probability ratio is ignored when it improves the target and is respected in the case of target degradation.

The second innovative part of PPO considers the estimation of the advantage estimation $\hat{A}_t$ (3.31). The work presented a combination of techniques that aim to reduce the variance of estimation and incorporate the entropy bonus that incentivizes policy to explore the environment. To reduce the variance of advantage function estimation it uses the state-value function $V(s_t, \theta)$ like the actor-critic method (Sec. 3.1.11). Besides that, it also considers the option that policy and value functions which are represented by ANN, share the same parameters θ to some extent. To preserve flexibility, the PPO assumes that policy updates may be made after collecting the trajectory τ of arbitrarily determined length. That assumption means that policy updates do not require collecting the entire episode. The following Equation 3.31 presents the estimator of $\hat{A}_t$ which meets all aforementioned requirements.

$$\begin{aligned} &\hat{A}_t = \varsigma_t + (\gamma\lambda)\varsigma_{t+1} + ... + (\gamma\lambda)^{T-t+1}\varsigma_{T-1} \\ &\textit{where:} \\ &\varsigma_t = r_t + \gamma V(s_{t+1}, \theta) - V(s_t, \theta) \\ &\lambda \text{ is a hyper parameter} \end{aligned} \tag{3.31}$$

The overall PPO algorithm is presented below in Algorithm 5. The algorithm enables the use of a set of actors that collect the experience trajectory simultaneously. When the trajectories of length T are gathered, the trainer computes advantage estimation $\hat{A}_t$ for all of the trajectories. Then the policy $\pi(u_t|s_t, \theta)$ is updated with gradients calculated based on differentiation of clipped objective function (eq. 3.30) with respect to parameters θ.

Algorithm 5 Proximal Policy Optimization

Require: Initialized parameters of θ, learning rates α_θ, α_w, and hyperparameters ϵ, λ

Set trajectory horizon T, K number of epochs, mini-batch size M

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

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

Run policy $\pi(u_t|s_t, \theta)$ in environment for T timesteps collecting trajectory τ

Compute advantage estimation $\hat{A}_1, ..., \hat{A_T}$

$\hat{A}_t = \varsigma_t + (\gamma\lambda)\varsigma_{t+1} + ... + ... + (\gamma\lambda)^{T-t+1}\varsigma_{T-1}$

where $\varsigma_t = r_t + \gamma V(s_{t+1}, \theta) - V(s_t, \theta)$

end for

Optimize objective

$J^{\mathrm{CLIP}}(\theta) = \hat{\mathbb{E}}_t \left[\min\left(r_t(\theta)\hat{A}_t, \mathrm{clip}(r_t(\theta), \epsilon)\hat{A}_t\right)\right]$

wrt θ, with K epochs and minibatch size $M < NT$

$\theta \leftarrow \theta + \alpha\nabla_\theta J^{\mathrm{CLIP}}(\theta)$

end for

3.2 Offline Reinforcement Learning

Reinforcement Learning is an effective method for learning-based control. It could achieve a near-optimal policy that knows how to pursue accordingly to reward function stated in MDP. However, the provided techniques depend heavily on online interaction with the learning environment. Such interaction may not be available for all problems, either because interacting is too expensive, dangerous, or taking too long. Offline Reinforcement Learning alleviates that issue by replacing online interaction with training on the offline dataset (Figure 3.3). The ANN-driven Offline RL is able to associate input states with actions, akin to supervised learning methods that learn input-label relationships. However, by incorporating a reward signal it states a decision-making learning technique that may provide behavior policies that are competitive with those produced by RL algorithms.

Typically Offline RL algorithms require a dataset D_{train} which is filled with transition tuples (s_t, u_t, r_t, s_{t+1}) generated by some unknown expert policy π_e during interaction with the target environment. During the training, the dataset could not be extended nor the trained policy cannot explore the environment. The policy could be trained by various algorithms, for example, Behavioral Cloning from Observation (BC) [70] or Monotonic Advantage Re-Weighted Imitation Learning (MARWIL) [71]. During the training, the policy performance is evaluated with off-policy evaluation metrics (weighted importance sampling [72] or distributional assumptions [73]). When trained policy meets the criteria, it is deployed to the environment and runs directly in the target domain.

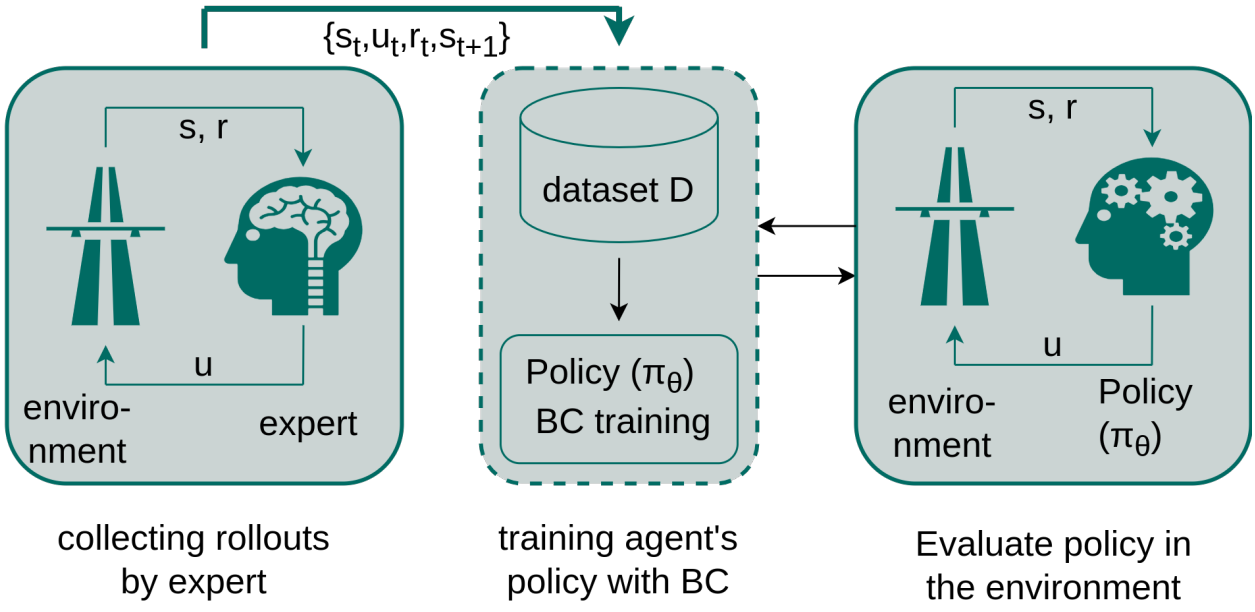


Figure 3.3: Offline Reinforcement Learning uses a learning dataset that is filled with experiences collected by some unknown behavior policy. The policy π_θ is trained on that dataset and it does not interact with the environment until it is finally deployed in the environment.

While the applied policy is running it could encounter transitions that are far from the distribution of these presented in the training dataset. It either could be varying states or selected actions in which con-

secutive state or expected outcome is unknown for the policy. Typically, the policy cannot handle deviating situations, which reduces its effectiveness. That situation is a result of a distributional shift and is common for all Offline RL algorithms becoming its major challenge. Usually, it is caused by the limited number of samples collected in the learning dataset that do not maintain the ideal distribution of samples over all possible environment states.

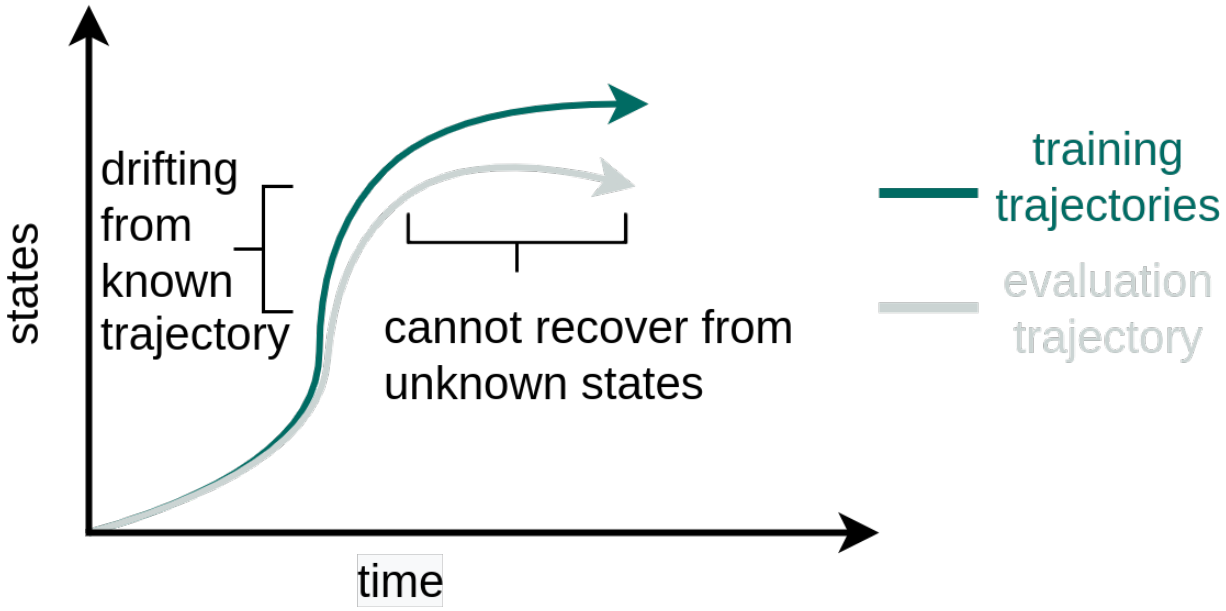


Figure 3.4: Drift of trajectory caused by distributional shift.

3.2.1 Extrapolation Error in Offline Learning

It is worth mentioning a phenomenon that disables learning with value-based methods such as Q-learning from the static offline dataset, without any exploration. As [74] stated, most RL algorithms are not able to learn effectively from historical data which does not correlate with the trained policy. It is caused by the extrapolation error which manifests in the erroneous estimation of q-values of state-action pairs not presented in the dataset. It also results in an optimistic value overestimation of these actions presented in the dataset. The policy that selects as an action the argument of the maximum q-value, not taking into account the uncertainty of estimation, causes persistent overestimation bias. Typically online RL algorithms, such as SAC [75], enable refining the estimation of the q-value which estimation variance is the highest by pushing the model to explore that part of state-action space. In offline learning such a refinement could not be carried out and even increasing the size of the dataset is ineffective [76].

There are three main reasons for the emergence of estimation divergences. The first of which is the absence of data in the training batch. If a particular state-action pair or near lying (s_t, u_t) one is not there, the initial estimation of the Q value cannot be adjusted. The following reason is the model bias towards the transitions of the consecutive states. The model estimates values over the next states, but it only knows trajectories that occur in the dataset. In value estimation, the model completely disregards the stochastic nature of MDP. The last factor influencing this phenomenon is the optimization process in relation to uniform

sampling from the dataset. Even if the dataset contains a sufficient number of samples, sampling results in loss which is weighted respectively to the likelihood of data in a batch. Sample distribution that does not correspond with state distribution may lead to increasing exploration error.

The following sections present the two most common Offline RL algorithms.

3.2.2 Behavioral Cloning

Behavioral Cloning was introduced in [77] and further elaborated in [70]. It relies on tuning the policy function to map the environment states to proper actions $\pi(s, \theta) \rightarrow u$. The set of state-action pairs is granted by the expert policy π_e, often human. It collects the trajectories τ of consecutive state-action tuples (s_t, u_t) that lead to achieving some desired goal. The trajectories τ are gathered in demonstration dataset $D_{\text{train}} = \{\tau_i, \tau_{i+1}, \ldots, \tau_N\}$. The algorithm adjusts the policy parameters θ minimizing the Kullback-Leibler Divergence between probabilities of selecting an action by trained and expert policies:

$$J = D_{KL}^{d}(\pi_e || \pi_\theta) = -\mathbb{E}_{(s_t,u_t)\in D_{\text{train}}}(\ln\pi(u|s,\theta) - \ln\pi_e(u|s)) \tag{3.32}$$

where $d(s)$ is state distribution. The loss function is expressed as a sum of differences between behaviors and policy natural logarithm of the probability of selecting the action that the expert took in a given state. The θ parameters of behavior policy are updated according to the gradient of the (3.32) in the gradient ascent step. This basic algorithm is quite similar to supervised learning. It does not employ reward signals which are crucial in Reinforcement Learning. Therefore it simply clones the expert's behavior without paying attention to the quality of actions. It cannot differentiate between two different actions being selected by expert policy in the same state.

3.2.3 Monotonic Advantage Re-Weighted Imitation Learning (MARWIL)

MARWIL [71] is a more advanced algorithm than BC since it takes into account the rewards received for performed actions. Similarly to BC, it also needs training dataset D_{train}, however, the state-action tuples are extended with reward value. Moreover, tuples in the dataset need to be organized in trajectories τ of consecutive time steps in order to calculate the cumulative sum of rewards G_t (3.3). This allows MARWIL to weigh available samples accordingly to advantage estimation of action $\hat{A}_t(s_t, u_t, \theta)$. The higher weight is put on a sample with a higher advantage, which inclines policy to select more profitable actions. Therefore theoretically, it could outperform the expert policy by avoiding unprofitable actions.

To calculate $\hat{A}_t(s_t, u_t, \theta)$ authors proposed to use

$$\hat{A}_t(s_t, u_t, \theta) = (G(\tau, t) - V(s_t, \theta))/c \tag{3.33}$$

where $V(s_t, \theta)$ is a value function, and c is an average norm that stabilizes training across different environments. The parameters θ are optimized to maximize the objective function:

$$J = \mathbb{E}_{(s_t,u_t)\in D_{\text{train}}} e^{\beta \hat{A}_t(s_t,u_t,\theta)} \ln\pi(u_t|s_t,\theta) \tag{3.34}$$

where β is a hyperparameter that controls the influence of advantage weighting of samples. If β equals 0, the algorithm transforms to basic BC (Sec. 3.2.2).

3.3 Sim2Real Gap

Robotics development relies on physical simulation which allows for rapid growth and progress of robotic applications. This is possible because simulation provides full controllability of occurring events and the possibility of their repetitions. Moreover, the simulation could be very effective by running much faster than real-time or by engaging fewer resources than real experiments. Its scalability over distributed computing allows for conducting a huge amount of experiments in parallel. The great advantage of simulation is also its variability which enables diversifying processes through parameterization. Finally, the evaluation of already developed robotics systems may be performed in homogenous conditions which allows for comparing systems without environmental errors.

Despite the broad range of advantages, simulation have also its limitations with correctly replicating real-world phenomena. For example, in the case of traffic simulation, there are plenty of scopes that can be simulated properly such as vehicle kinematics, traffic flow, or traffic accidents. However, there may be missed simulation of the broad range of edge cases which made autonomous driving so difficult to achieve. Moreover, there may be flawed replication of the common driver's behaviors which impacts the optimization of the policy behavior.

Modeling errors can have a significant impact on the successful transition of robotic systems from simulation to real-world environments. The process of such transferring is called sim2real transfer. This difficulty comes from the way in which robotic systems are created. During the development, the system may be optimized to handle only these states that were simulated. However, the states that were absent during optimization or were far from a presented one in the real domain cause the robotic system may behave suboptimally, incorrectly, or even dangerously. The difference between simulation and real-world which causes the aforementioned threat is called the sim2real gap and it is a special case of distributional shift.

3.3.1 Domain Randomization

Recently, a significant number of works has been done (ex. [78], [79], [80]) to alleviate the problem that restrains the application of RL-driven robotic systems in real products. Works considered two approaches, the first is to improve the simulation to close the sim2real gap. The second is to utilize real data in parallel with simulated one.

An example of the first approach regards closing the sim2real gap by increasing the diversity of simulation by steering its parameters. Work [81] introduced a method called domain randomization. The work [82] enhanced it and proved its efficiency by training the robot hand to solve the Rubik's cube. The robot was trained initially in simulation, while evaluation took place on real hardware.

Generally, domain randomization relies on changing the simulation parameters during the training according to some broad distribution. This method assumes diversification of a set of parameters including those that characterize a real setup. Therefore agent that handles all possible environment states can handle actual states occurring in the real world. The process of changing parameters can be manual or automated. The [82] proposed starting with a narrow range of distribution which simplifies the environment helping agents learn. Whenever an agent masters the current setup, Automatic Domain Randomization (ADR) ex-

pands the range of distribution which forces the agent to adapt to broader ranges of states. It maintains the hardness of training and agents become resistant to versatile situations.

The disadvantage of this method is its time consumption. Because the agent is required to cope with a broad range of environment versions, the training takes much more time than is needed to learn for one particular setup. The time required increases exponentially with an increasing number of tweakable parameters. As in the case of machine learning it needs to evaluate policy in the target domain frequently to determine if training covered already real parameters and if it could be stopped or at least prevent the expansion of the randomization range.

The second drawback of this method is that the optimal policy that copes with all possible conditions may not be optimal under real conditions. This counterintuitive observation could be explained by example. Imagine that one of the simulation parameters is the force that is needed to move the robot's joint for some distance unit. If the value of this force is included in observation space or the robot can infer it somehow, the action may depend on that value. However, if it is not observable in any way, the agent would search for a policy that satisfies the entire range of distributions. It means that the policy would not be optimized with respect to real value, therefore it would not be optimal under real conditions.

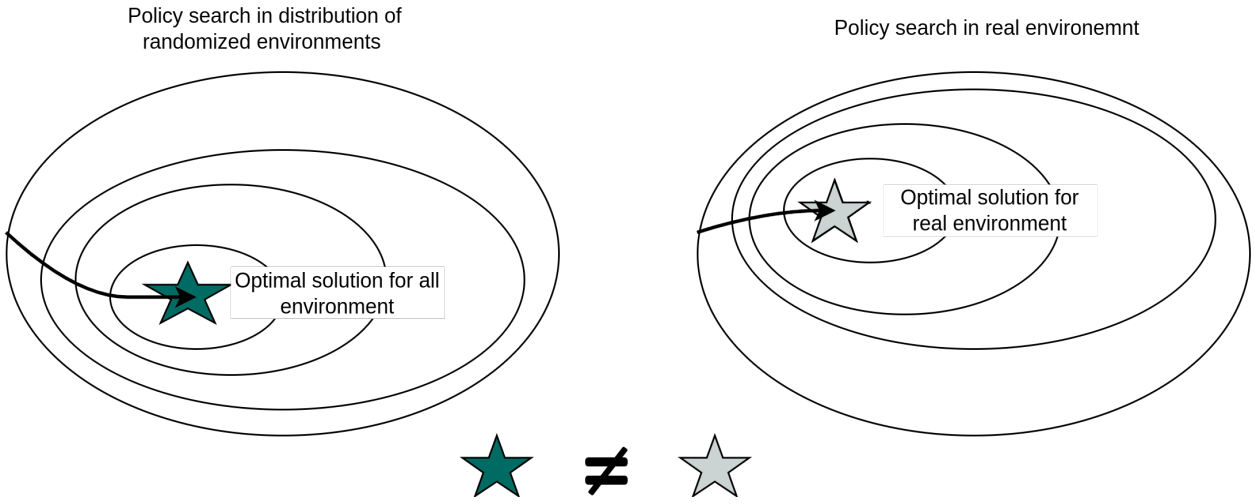


Figure 3.5: Policy optimal for all distribution of environments may be not optimal for the real environment.

3.3.2 Domain Adaptation

The second method that closes the sim2real gap is Domain Adaptation (DA). The most popular method in this class is the utilization of Generative Adversarial Networks (GAN [83]) in order to improve simulated observations. GAN consists of two ANNs, one of which is a generator and the second one is a discriminator. The Generator is trained to generate the samples in such a way that they are similar to the samples in the training set. The Discriminator is trained to recognize whether a given sample is from a dataset or from the Generator. These two ANNs are trained simultaneously in the form of a zero-sum game. One of the recent works that incorporated this family of solutions is [84]. It regarded typical end-to-end training of robots that should grasp some object. RL agent which controls the robot receives images from the camera sensor as observations of the environment state. The sim2real gap considered here is the unrealistic image rendering

and faulty sensor model in simulation. To overcome these issues authors proposed to train two GANs. The first of which transformed simulated images to mimic realistic pictures and the second one mapped real images into simulated ones. This type of training required a dataset filled with images from real hardware. The policy is trained with DQN [85] on the samples modified by the first Generator. During the evaluation, the agent received images directly from the real camera, which were not processed by the Generator. This method could be applied also to other types of states as far as it is possible to determine the architecture of GANs.

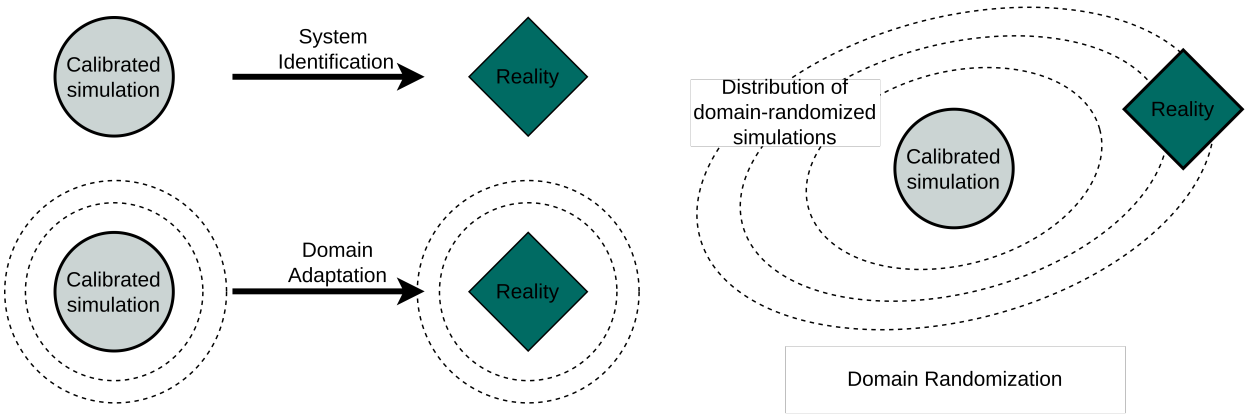


Figure 3.6: Various methods of closing sim2real gap. System identification starts the process of building simulation tools. The simulator needs expensive calibration to resemble reality. Domain Adaptation relies on transfer learning techniques that transform simulated states to match the real ones. Domain Randomization randomly parameterized simulation and train model across the distribution of environments. It assumes that one of them matches reality.

3.3.3 Real Data Utilization

The last method which may bypass the sim2real gap, is to utilize real data in training in conjunction with the simulated one. Offline learning (Sec. 5.4) formalized learning methods on data collected in a real environment by some external policy. However, as already indicated above, these methods suffer from distributional shifts due to limitations on data coverage (Fig. 3.6). Therefore a number of works presented an approach of combining offline and online learning to minimize the sim2real gap while preserving coverage over state distribution. This type of approach enables minimizing the drawbacks of the particular method. However, the approach of simply pre-training policy on offline data and then fine-tuning it in simulation faces at least two perils. One of which is the extrapolation error presented in Section 3.2.1, and the second one is an issue of catastrophic forgetting knowledge learned during the pretraining stage on the fixed dataset.

One example is [86], which exploited real-world data to train flying micro UAV and utilized simulated data to generalize behavior policy. The work considered the problem of modeling complex physics and air currents as a significant factor that reduces the quality of learning in simulation. The authors proposed to separate neural network policy into the perception part and control module. The perception subsystem is trained on a significant number of simulated data with a Q-learning algorithm (Sec. 3.1.7). Then control

module is trained to predict future rewards based on images from the real robots and a sequence of future actions. This module was part of the control policy, which selected actions with the highest predicted reward. Because the control module was trained on a narrow data set, it was required to generalize better across various states. Therefore, the perception module trained in the simulation was transferred into the reward prediction model.

Another combination of online and offline data is proposed by [87] in order to develop a chatting agent. The authors proposed to pre-train generative model $\pi_{\mathrm{gen}}(u|s)$ on a batch that contained state-action pairs from interaction demonstrations. Then conduct RL training on Q-network policy on samples collected by chatting with humans online. To suppress deviation from prior knowledge, the policy is penalized for divergence between prior π_{gen} and actual π_θ. The difference was calculated with a modified form of Kullback-Leibler (KL) divergence. To minimize the overestimation of the Q-value of state-action pairs, authors pondered using the clipped double Q-learning method [88]. However, because the used Q-network had a considerable size, doubling it may result in tremendous memory and computation load. Therefore authors decided to use a single target Q-network and sample distribution of actions applying dropout on each layer [89]. For every sampling, the dropout parameters were changed according to the Gaussian distribution. During the training and inference, the action corresponded to the minimum value from the action distribution.

Chapter 4

Experimental Setup

4.1 Highway Environment

The experiments assume that driving takes place on a highway with a maximum number of six lanes. Additionally, the highway infrastructure contains merging ramps and exit lanes, which diversify traffic scenarios introducing maneuvers such as merging in and out, slowing down, and speeding up in order to join traffic or exit the highway. The speed limit is not assigned to the lane. However, the vehicle user selects the velocity setpoint, which may be dynamically adjusted. The expected objects on the road are cars, motorcycles, and trucks, while pedestrians are not likely to participate in highway traffic. The controlled car has parameters corresponding to the Lincoln MKZ model (Tab. 4.1).

For experiments, we developed an ACC environment, which is implemented according to the OpenAI Gym interface [90]. The implementation provides the following functionalities: observation space, actions space, reset, step, and render. Observation space specifies the size of the feature vector and the range of values that are returned from the reset and step functions. What the agent observes and how the environment state is encrypted is described in detail in Section 4.1.2.

Action space determines valid ranges of action that the agent can pass to the environment. In this case, action space is continuous in the range $u \in [-1, 1]$. The action value corresponds to the targeted longitudinal acceleration value in the range $a_{s,\text{target}} \in \left[-3.5m/s^2 - 1.5m/s^2\right]$. More details about the design and implementation of action space may be found in Section 4.1.3.

The reset function randomly generates new traffic scenarios from predefined simulation parameter ranges and returns the first observation in the new episode. We customized the generation process in order to achieve realistic simulation conditions (Sec. 4.1.1).

The step function takes agent action as an argument and executes it in simulation. The function is called every 0.5s of simulation, which means that the agent specifies acceleration that should be achieved in such a period of time. Action execution starts with recalculating it to the targeted acceleration value $a_{s,target}$ (4.1). Then the trajectory module (Sec. 2.2) plans a continuous reference trajectory to achieve it. The trajectory is then executed consecutively in the simulation until the function is called again. The simulation step is called ten times in 0.05s intervals to properly simulate the traffic and calculate possible collisions. During the update, the simulation samples the acceleration value from the trajectory for a current time and updates

the position and state of the host vehicle based on a specified motion model (Sec. 2.4). At the same time, the simulation updates the state of other objects using a built-in motion model. After updating the states of all vehicles, the simulation checks collisions between all traffic participants.

In the end, the step function returns the observation of a current environment state, reward, information about agent performance, and signal, which notifies whether the agent reaches the terminal state. The reward is calculated by the reward function (Sec. 4.1.4) based on a given state and executed action. The episode achieves a terminal state when the host vehicle crashes or the episode reaches 150 steps.

The remaining render function visualizes the simulation and presents some information about the host vehicle, agent performance, and traffic state (Fig. 2.5).

4.1.1 Scenario Generation

The scenario generation process includes designing a map, selecting traffic parameters, and picking a velocity set point. The map is created procedurally and starts with selecting the number of lanes of a major highway in the range of 1 to 4. The road is created by selecting the length and curvature of the consecutive road sections. Every several sections, the process may add merging or exiting ramps, in which parameters such as the number of lanes in the range of 1-2, length, and curvature are also randomized. The procedure is constrained in advance by the parameters specified by the user to avoid semantic incorrectness.

Having the highway map prepared, the scenario generator proceeds to specify traffic by selecting the parameters of the simulated drivers that will be spawned. This process is divided into selecting parameters of vehicles, characteristics of drivers, and targeted traffic flow. The parameters of the vehicles are taken from different models of real vehicles, while the drivers are drawn from a pool of driver behavior models. The specified profiles include drivers who drive carefully, aggressively, or recklessly.

The targeted traffic flow specifies the number and frequency of adding the new drivers to the simulation. New drivers are only spawned at the beginning of the road or merging ramps. In the end, the scenario generator decides on the velocity set point for a host agent. Velocity is drawn from a range of 15-40 m/s. It allows for the diversification behavior of the trained agent and teaches it to distinguish between absolute velocity and target velocity.

There were specified four types of scenarios that are different in resulting traffic flow. The most straightforward scenario has no other objects besides the trained agent. Subsequent sets increase parameters gradually to make traffic flow denser. In the last set, the scenario is able to force slow driving on the cluttered highway resulting finally in traffic congestion. Such separation may be used as a ground for curriculum learning which relies on increasing the complexity of the scenarios adequately to agent performance [82].

4.1.2 State Observation

The environment state in time t is represented as $s_t \in S \in \mathbb{R}$. The state is defined as a vector that contains all the relevant information about each vehicle and the environment states $s_t = \left[s_i^{\mathrm{v}}, s_{i+1}^{\mathrm{v}}, \ldots, s_N^{\mathrm{v}}, \rho, \ldots \beta, \ldots\right]$. Where s_i^{v} is the state vector of the i^{th} vehicle: $s_i^{\mathrm{v}} = [x, y, v_x, v_y, \zeta_{WCS}]$. Road model ρ is a vector that includes information about the road layout and geometry, such as curves, joints, and lane markers. $\rho = [c_i, c_{i+1}, \ldots, c_n, j_i, j_{i+1}, \ldots, j_m]$ where c_i is a vector that includes an identification number, marker

Lincoln MKZ parameters		
length l	4.925	m
width w	1.864	m
height h	1.475	m
center to front axle l_f	1.42	m
center to rear axle l_r	1.42	m
max speed	68.9	m/s
max reverse speed	2.8	m/s
max acceleration	4.5	m/s^2
max braking decelaration	10.0	m/s^2
mass m	1774	kg
wheel radius r_w	215.9	cm
Engine parameters		
min rpm	600	rpm
max rpm	6000	rpm
min torque	100	Nm
max torque	373	Nm
Gear ratios		
-2.94, 0.0, 4.58, 2.96, 1.91, 1.45, 1.0, 0.75		

Table 4.1: Parameters of host vehicle 'Lincoln MKZ'

type of given c_i, and points coordinates that describe given curve $c_i = [i, c_{\text{type}}, x_i, y_i, \ldots, x_n, y_n]$. Marker (delimiter) type $c_{\text{type}} \in \{0, \ldots, N\}$ what symbolizes all typical lane marker types such as solid, dashed, double solid, double dashed, dotted, etc. Then j_i is a tuple that includes a set of identification numbers of c_i that create a road joint. The curve connections provide comprehensive information about road topology.

β contains the data related to the controlled by agent vehicle, such as targeted preset velocity ($v_{s,\max}$), actions that were selected in the last step ($a_{s,\text{target},t-1}$), information whether the host is colliding with other object and caused a collision. The symbol $'\ldots'$ denotes any other relevant variables or information that influence the traffic scenario or internal simulation state machine. In summary, the state s_t provides a complete depiction of the vehicles and environment, essential for a precise simulation of the traffic scenario.

The training environment is defined as POMDP due to the fact that the agent is not able to observe the entire state s_t. Therefore state in time t is represented to the agent as a state observation $o_{s,t} \in \Omega$ by $O(s_t, u_{t-1}) \rightarrow o_t$. Observation consists of essential information from the perspective of behavior planning. The $o_{s,t}$ is narrower than s_t and it is limited only to the features that are visible from the host position perspective and sensor models capacity (Sec. 2.3). Overall, it consists of three major parts, which regard the host state, perceived targets, and road model $\Omega = \Omega_{\text{host}} \times \Omega_{\text{objects}} \times \Omega_{\text{road}}$.

The host state $\Omega_{\text{host}} \in \mathbb{R}^4$ is represented as a feature vector that includes: absolute velocity in VCS ($v^{\text{vcs}} = \sqrt{(v_x^{\text{vcs}})^2 + (v_y^{\text{vcs}})^2}$), level of preset speed limit execution ($v_{\text{s\_limit\_exec}} = v^{\text{vcs}}/v_{\max}$), absolute acceleration ($a^{\text{vcs}} = \sqrt{(a_x^{\text{vcs}})^2 + (a_y^{\text{vcs}})^2}$) and acceleration targeted in previous time step ($a_{s,\text{target},t-1}$). The $v_{\text{s\_limit\_exec}}$ is crucial from the perspective of satisfying vehicle user preferences. Its value depends on $v_{s,\max}$ which is the maximal speed at which a vehicle could drive in accordance with user settings. $\Omega_{\text{host}} = \{v^{\text{vcs}}, v_{\text{s\_limit\_exec}}, a^{\text{vcs}}, a_{s,target,t-1}\}$

Objects observation is a matrix that consists of at most ten feature vectors, and each of them describes one vehicle $\Omega_{\text{objects}} \in \mathbb{R}^{10\times 11}$. The observation considers up to 10 vehicles that are closest to the host. The features include relative to the host information about longitudinal and lateral distance $(x^{\text{vcs}}, y^{\text{vcs}})$, velocity $(v_x^{\text{vcs}}, v_y^{\text{vcs}})$), and acceleration $(a_x^{\text{vcs}}, a_y^{\text{vcs}})$. It also consists data about target's rotation (ζ_{vcs}) and its dimensions (w, l).

For training purposes, we also added information on whether the target is visible b_{visible} from host sensors and whether the target is valid b_{valid}. A valid flag informs the neural network whether the target described in this row exists. Valid and visibility flags are used to mask out objects in the transformer attention layer. To ensure software compatibility, observation size must adhere to the predefined shape of (10x11). However, it is important to note that there may not always be ten valid objects present, such as in the case of an empty road. An object not perceived by the host may also be included in observation for different purposes. For example, algorithms that use the critic part for learning (ex. SAC [75]) may diverge in observation composition for the actor and critic. Since critic is not used during evaluation, it may know the entire environment state, including information unavailable for the actor [91]. The visibility mask is used to filter out information that the actor should not consider. $\Omega_{\text{objects}} = \{s_{\text{obj}}^{\text{vcs}}, d_{\text{obj}}^{\text{vcs}}, v_{x,\text{obj}}^{\text{vcs}}, v_{y,\text{obj}}^{\text{vcs}}, a_{x,\text{obj}}^{\text{vcs}}, a_{y,\text{obj}}^{\text{vcs}}, \zeta_{\text{vcs,obj}}, w_{\text{obj}}, l_{\text{obj}}, b_{\text{visible}}, b_{\text{valid}}\}$

The last part of observation regards the road model which is represented by a 3-dimensional tensor (number of marking lines $\times$ sampled points along lines $\times(x^{\text{vcs}}, y^{\text{vcs}}, c_{\text{type}}))\Omega_{\text{road}} \in \mathbb{R}^{6\times 10\times 3}$. Each row

describes one physical marker line on the road, which forms a driving lane boundary. The sensor allows only forward perception of a road in a maximal range of 150m. Each row consists of coordinates of 10 consecutive points in the vehicle coordinate system (VCS) and information about delimiter type (Fig. 4.1). Points are separated from each other at constant distances over the entire detected line. The information is completed by the 'edge_index' vector, which tells which points are adjacent, and the cluster vector, which groups the points into lines. These two complementary vectors are required by the graph part of Neural Network (sec. 4.3)

A valid range of values has been designed for each feature described above. This range is used to normalize observation within the range $[-1, 1]$ before passing it to the neural network. Normalizing inputs helps in achieving faster and more stable convergence during the training process. It provides stability during backpropagation by keeping gradients within a reasonable range, preventing undesirable issues like exploding or vanishing gradients. It also acts as a form of regularization and helps to avoid overfitting. Constraining the input values to a similar scale encourages the neural network to focus on the patterns and relationships within the data rather than being influenced by absolute values. This promotes better generalization and improves the network's ability to perform well on unseen data.

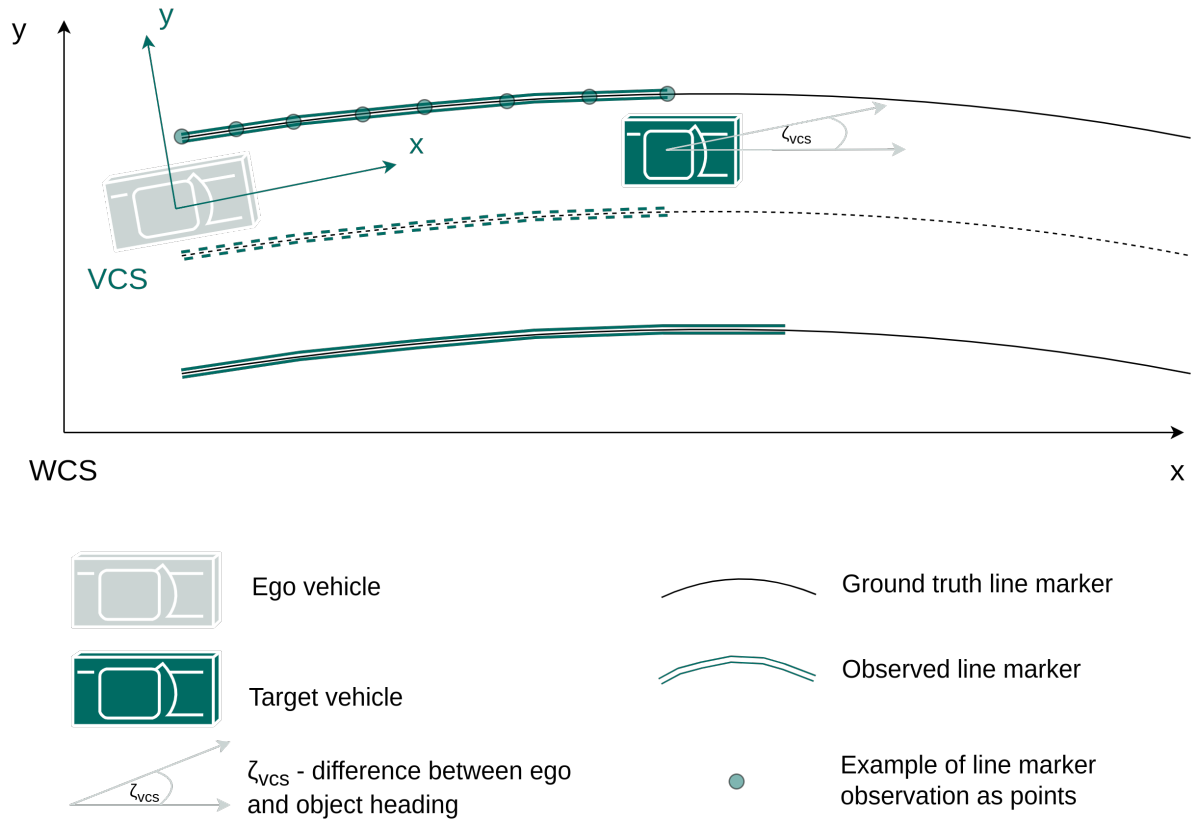


Figure 4.1: Visualization of observation space - environment state s_t is presented to the agent as an observation $o_{s,t}$.

4.1.3 Action Space

Action space in the ACC environment is defined by a continuous range between $u \in [-1, 1]$. The action value is recalculated to the acceleration value which is in range $a_{s,\text{target}} \in [-3.5m/s^2 - 1.5m/s^2]$ with

$$a_{s,\text{target}} = a_{\min} + \frac{(a_{\max} - a_{\min})(u - u_{\min})}{u_{\max} - u_{\min}} \tag{4.1}$$

where $a_{\min} = -3.5$, $a_{\max} = 1.5$, $u_{\min} = -1$, $u_{\max} = 1$, and $a_{s,\text{target}}$ is targeted longitudinal acceleration, u - action returned by policy π_θ; $u \in \{-1, 1\}$.

The calculated value is a target acceleration that the agent expects to achieve in the next environment state $t + 1$ which states half a second. As demonstrated in figure 2.3, the acceleration value derived from agent action goes directly to the trajectory generation module. This module plans future vehicle trajectory in such a way as to reach the target acceleration in settled duration, keeping constant jerk. The trajectory is then executed by simulation using the motion model. Therefore the limits of possible acceleration are determined arbitrarily by the trajectory generation module developed by 3rd party. From the agent's perspective, training can be adapted to other acceleration ranges. In the current setup, the acceleration range is available from the perspective of the trajectory planner module and dynamic motion model. Target values should be feasible essentially from the perspective of the motion model used in simulation 2.4. In residue situations when

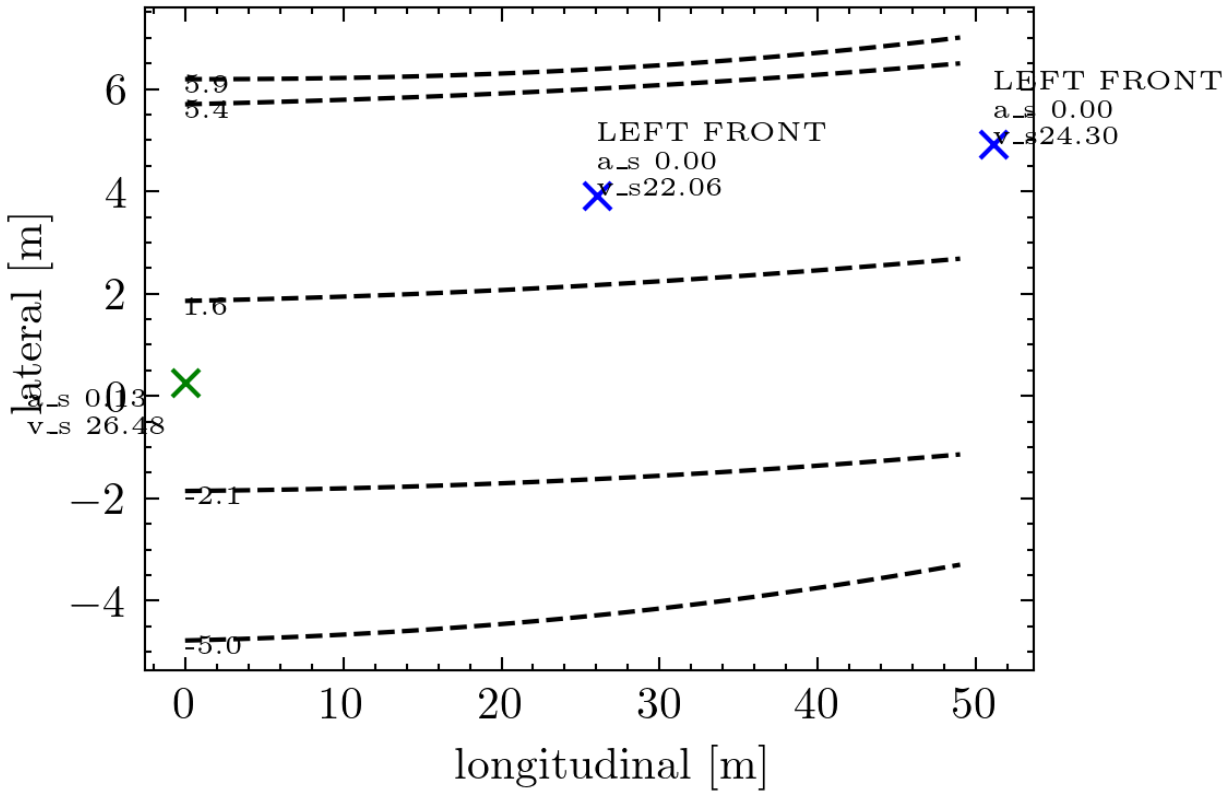


Figure 4.2: Example of state observation from the driving log. There are five marker lines that create the host and left lanes. The most left and right marker lines represent the road border. The host position is marked by a green cross. Perception detected two objects on the left lane (blue crosses). Figure 4.3 presents the view from the front-facing camera, where can be seen two additional vehicles that were not detected.

Figure 4.3: View from the front-facing camera. State observation is presented in Figure 4.1.

target acceleration cannot be achieved, we expect the agent to infer such an aspect during the learning. The range of acceleration also complies with the analysis of quality requirements. The work [92] reported that maximal comfortable acceleration for 50% of drives is $1.6m/s^2$ and maximum comfortable deceleration for 90% of drivers is $3.3m/s^2$. The value of deceleration is selected to provide comfortable driving. It is much lower than values reached during the activation of the Autonomous Emergency Braking System in near collision situations $7.7m/s^2 <$.

4.1.4 Reward Function

The reward function is a sum of several weighted terms, and each incentivizes appropriate behavior (positive reinforcement) or discourages undesirable actions (negative reinforcement). The reward function used for training the ACC agent R_{acc} consists of the following terms:

- **Speed execution** $c_0(v^{\text{vcs}}, v_{\max})$ - calculated as a ratio of current host velocity and setpoint velocity. The reward is adequately lower if the speed is higher than the velocity setpoint. The maximum value is 1 as shown in Figure 4.4. It incentives agents to maximize speed limit execution and not exceed it and it is defined as

$$c_0(v^{\text{vcs}}, v_{\max}) = \begin{cases} \left(1 - 3\frac{|v_{\max} - v^{\text{vcs}}|}{v_{\max}}\right), & \text{if } v_{\max} < v^{\text{vcs}} \\ 1 - \frac{|v_{\max} - v^{\text{vcs}}|}{v_{\max}}, & \text{otherwise} \end{cases} \tag{4.2}$$

  where $v_{\max} \in [0, 35]$ and $v^{\text{vcs}} \in [0, 45]$.

- **Squared acceleration** $c_1(a^{\text{vcs}})$ - the squared value of acceleration

$$c_1(a^{\text{vcs}}) = (a^{\text{vcs}})^2 \tag{4.3}$$

  This negative reward promotes a comfortable ride without upheaval. The weight of this term should be carefully selected in accordance with the speed execution weight. This term considers the actual acceleration value which is performed by the host vehicle a_t^{vcs}, not the planned acceleration by the agent $a_{s,\text{target},t-1}$.

- **Safety violation** $c_2(s^{\text{v}}_{\text{host}}, s^{\text{v}}_{\text{target}})$ - a negative reward for entering the safety distance while driving too close to the front target. Distance is calculated according to Responsibility Sensitive Safety (Sec. 2.5). This term incentivizes agents to drive safely and to avoid triggering emergency braking.

$$c_2(s^{\text{v}}_{\text{host}}, s^{\text{v}}_{\text{target}}) = \begin{cases} 1, & \text{if } \Delta s \leqslant d_{\text{lon\_min}} \\ 0, & \text{if } \Delta s \geqslant d_{\text{lon\_min}} \end{cases} \tag{4.4}$$

  where $d_{\text{lon\_min}}$ is defined as (2.19) and Δs is the distance alongside lane centerline between the front bumper of the host vehicle and the rear bumper of the leading vehicle $\Delta s = s_{\text{obj}} - 0.5l_{\text{obj}} - s_{\text{host}} + 0.5l_{\text{host}}$, where s i position in Frenet Coordinate System of the vehicle center.

- **Terminal State** $c_3(\Delta s, v^{\text{vcs}})$ - a negative reward for causing a collision or speeding too much. If any of these conditions are fulfilled the episode ends because detecting such an event causes the termination of an episode.

$$c_3(\Delta s, v^{\text{vcs}}) = \begin{cases} 1, & \text{if } \Delta s \leqslant 0 \vee v^{\text{vcs}} \geqslant 45 \\ 0, & \text{otherwise,} \end{cases} \tag{4.5}$$

$$R_{\text{acc}}(s^{\text{v}}_{\text{host}}, s^{\text{v}}_{\text{target}}) = 0.11c_0(v^{\text{vcs}}, v_{\max})^2 - 0.02c_1(a^{\text{vcs}}) - 0.3c_2(s^{\text{v}}_{\text{host}}, s^{\text{v}}_{\text{target}}) - 10c_3(\Delta s, v^{\text{vcs}}) \tag{4.6}$$

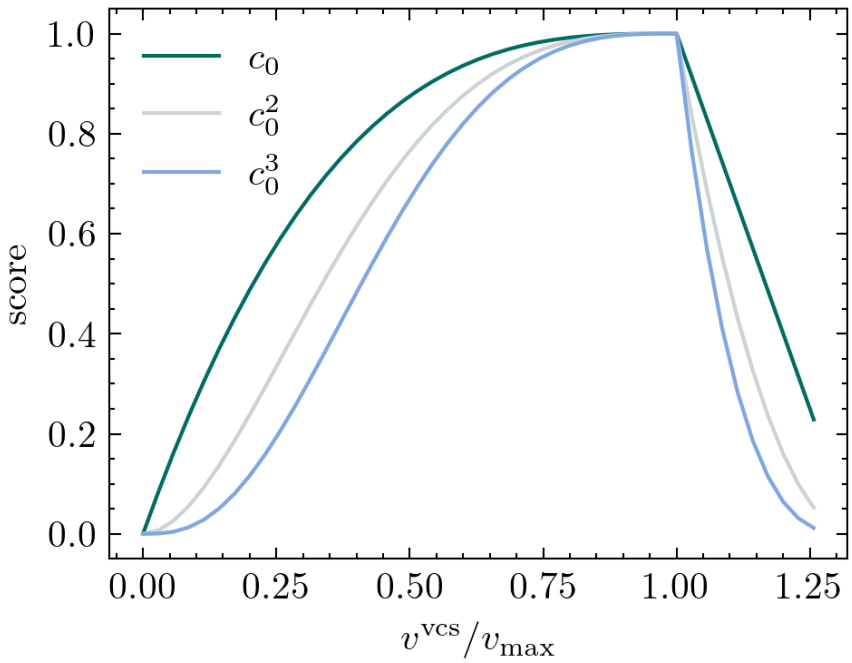


Figure 4.4: Reward function of $c_0(v_s, v_{\text{max}})$ and its square.

The reward function was created on the basis of multiple agents' training sessions conducted using PPO in a highway environment. It is important to notice that in the case of reinforcement learning, the reward function does not serve only for loss calculation in the policy optimization process. It also drives the agent's exploration of the environment. Therefore it needs to be carefully designed, and partial terms weights must be balanced. Since the reward function incentivizes environment exploration, its weight also depends on the distribution of transitions and scenarios. For example, in the contemplated ACC case, the general objective is to minimize the sum of acceleration while maximizing speed. It means that the negative reward for acceleration should be high. However, if the sum of the rewards is too high with respect to the reward for maximizing velocity, an agent would not be incentivized to explore outcomes from states with high-velocity driving. According to experiments, it suppresses exploration in order to avoid negative reinforcement for acceleration just from the beginning of training, therefore agent never experiences high rewards for fulfilling preset velocity.

The presented reward function focuses on velocity and acceleration values. However, the other approach for designing a reward function for ACC is to concentrate on the distance to the leading target rather than on velocity (such as [25]). It relied on the assumption that by minimizing the separation, the speed is also maximized. However, our conducted experiments on such a defined reward function contradict this. The evaluations showed that the agent did not maximize speed but allowed other cars to cut in front of it. Thus distance was always small, and the reward was maximized, but the real ACC objective was not fulfilled. This result may be due to the characteristic of the behavior model built-in simulation used for controlling other vehicles, which tend to change lanes often. It is possible that such an approach would be effective in a single-lane road environment.

Regarding weight balance, we found out that too small a negative reward for collisions provokes an agent to especially cause a crash to avoid collecting negative rewards for the rest of the terms in each step of an episode.

As a part of reward designing, we investigated the issue related to the responsibility for safety. Initially, we assumed that the agent would be wrapped by the RSS (Sec. 2.5) safety envelope, which would trigger emergency braking when the agent enters the safety distance. However, for training purposes, we switched off this option for two reasons.

In the first place, we intended to train agent which understand the safety issues. Therefore, we can punish it with the safety violation reward term c_2, and through that, it could comprehend the consequence of accelerating being close to the leading vehicle.

Moreover, we found out that transparency between actions and state transitions contributes much to the learning pace. Therefore, transparency was achieved by allowing the agent to accelerate at any time and not override its action by braking signal from RSS.

The performance of the outcome policies was evaluated based on defined KPIs (Sec. 6.2). The reward function that led to the highest-performing policies in terms of KPIs was selected as the basis for the remaining experiments.

4.2 Dataset

The training dataset D_{train} used for experiments consists of 672211m driving experience. The data set covers various traffic situations, road geometries, weather conditions, and broad traffic flow distribution. Table 4.2 presents major features that describe the characteristics of collected samples. The below figures show the distribution of driver speed limit execution (Fig. 4.6) and acceleration (Fig. 4.5). The data set comprises around 60000 state-action pairs sampled every 0.5s from raw data. That complies with an interval of selecting behavior in the ACC environment (4.1). Data was collected on the highways and roads outside cities, mainly in Germany. The data has been segmented into episodes that have a duration of up to two minutes which corresponds to 240 steps. Records from roads inside cities or parking areas were automatically filtered out from available data based on the host's mean speed in the episode. Raw data is collected from vehicle sensors such as the front-facing camera, all-around radars, and internal car sensors such as IMUs and wheel encoders, as described in Section 2.3). Raw data is processed into trajectories of consecutive SARS' tuples $\tau = \{(s_t, u_t, r_t, s_{t+1}), \ldots, (s_{T-1}, u_{T-1}, r_{T-1}, s_d)\}$, which are necessary for further training, with the following pipeline.

First of all, each sample is parsed to a common representation used in our implementation of the Highway Environment (sec. 4.1). To simplify the implementation of all experiments, we ensure that objects that are created from sensor data and simulation implement a common interface. This approach additionally guarantees that objects from both sources are coherent and both cover common types of data. Common representation consists of three major objects which are being created: Host object; Targets - representation of all visible adjacent vehicles; Relative Road object, which represents road lanes in relation to the host vehicle. Such representation of the environment could be processed further to the state observation $o_{s,t}$ as defined in Section 4.1.2.

Parameter	Original Dataset	Unit
Road travelled	672.211	km
Acceleration Mean	-0.011	m/s^2
Acceleration Std	0.372	m/s^2
Speed wrt Speed Limit Mean	0.563	%
Speed wrt Speed Limit Std	0.196	%
Speed Mean	22.513	m/s
Speed Std	7.847	m/s
Jerk Mean	0.784	m/s^3
Jerk Std	6.707	m/s^3
Headway to front Mean	10.066	m
Headway to front Std	90.010	m
Lane Bias absolute mean	0.341	m
Lane Bias absolute std	1.522	m

Table 4.2: Characteristics of collected samples in dataset

To complete SARS' tuple, there is a need to determine the actual driver action $u_{e,t}$. The action is defined as a normalized value of target acceleration (4.1). Unfortunately, acceleration data from the sensors could not be directly used since the driver did not apply the interfaces and steering methods employed in the experiments. The plotted acceleration curves indicated that drivers changed acceleration more frequently and in a broader range than specified, or that sensor readings were too noisy. Nevertheless, vehicle acceleration reported by sensors could not be directly used as a reference for action in agent training. It is mainly due to the fact that acceleration specifies the desired value of acceleration and not the resulting one from trajectory execution $(a_{s,target,t} \neq a_{s,t+1} \forall t \in T)$.

In order to estimate the value of the action, we hypothesized that the achieved speed in a particular time step remains close to the driver's intention. Based on this assumption, the targeted acceleration value $a_{s,target,t}$ was calculated based on the reached velocity in time step $t + 1$. For calculation, we employed a 3rd order spline to represent the velocity function with knots at each timestamp of the samples. This method was found to approximate the dynamics of velocity changes effectively and it complied with approach of calculating trajectory by trajectory planning module. The acceleration values were then obtained by computing the function derivative at the sampling timestamp and scaling it into the range of [-1, 1], as outlined in Section 4.1.3 (4.1). We utilized the dynamic model to verify that computed $a_{s,target,t}$ actually leads to achieving velocities performed by the driver. After inspection, it was determined that using a 3rd order polynomial to approximate the acceleration profile was justified. This choice resulted in a trajectory that closely resembled the original trajectory and remained feasible.

In order to finalize the SARS' tuple, it was necessary to compute the reward, which is dependent on both the action and states. To address this, we gathered consistent observations of the states and driver actions and used reward function (4.6) to calculate the reward for each sample.

In order to conduct further experiments, we divided the dataset into a test set and a train set with a 1:9 ratio.

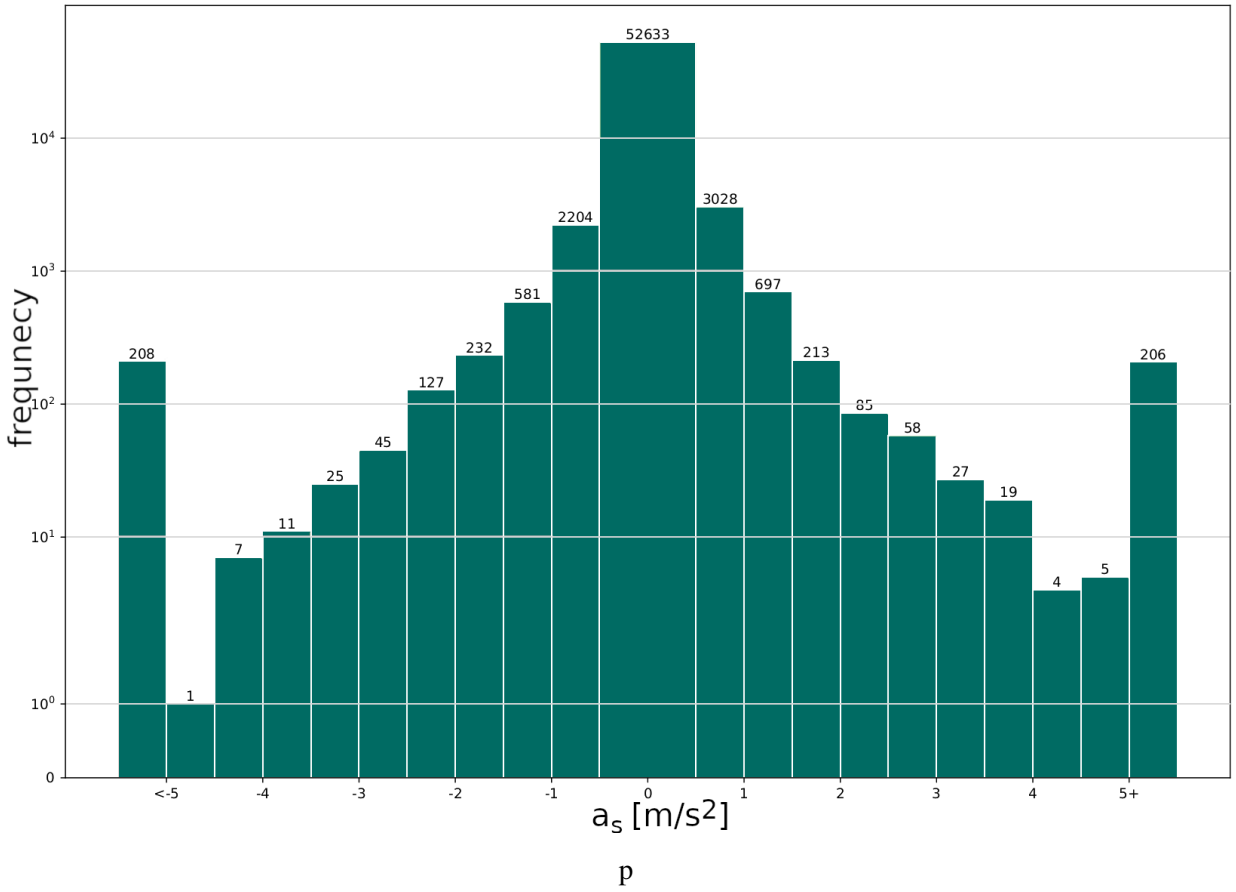


Figure 4.5: Histogram of accelerations in the dataset D_{train}.

4.3 Neural Network Architecture

The Neural Network architecture, designed for experiments (Fig. 4.7), complies with several requirements resulting from data characteristics. First of all, ANN should be able to process a variable number of objects and lane markers. To efficiently handle this case and simultaneously keep a fixed input shape to ANN as the software requires, we employed TransformerEncoder structure [93] and PointNet architecture [94]. Moreover, in order to effectively address the challenge of varying numbers of lane markers and sensing ranges, we conducted a search for an efficient representation. Also to consider the topology of driving lanes, we utilized the ordered graph structure to represent lane markers. This representation was processed by a graph neural network [95] with message passing operation [96].

Additionally, to address the challenge of temporal aspects in processing the observation of the POMDP environment, we utilized the Long-Short Term Memory layer (LSTM) [97]. This was necessary because the history of objects is not included in environment observation but has a significant impact on the decision-making process. This enabled us to retain valuable information from previous time steps and improve our overall performance.

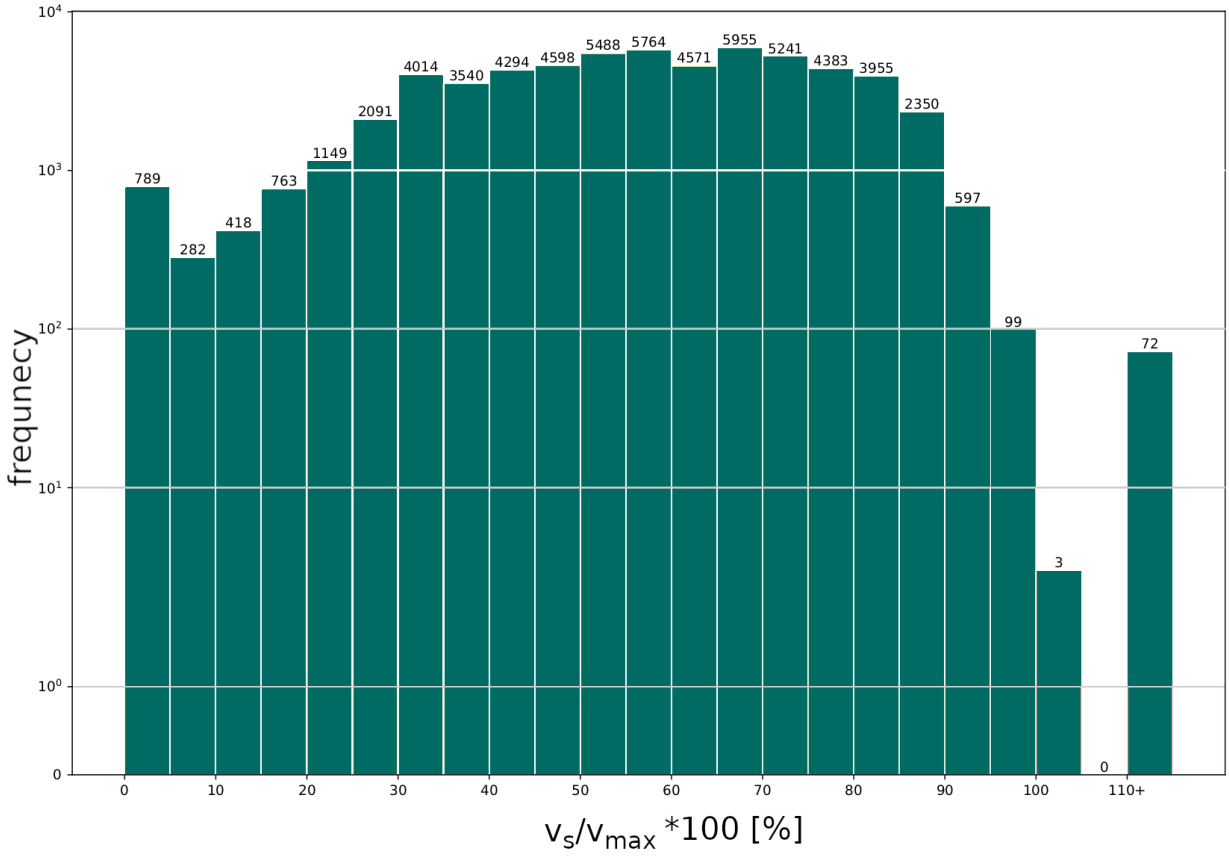


Figure 4.6: Histogram of speed limit execution ratio for all samples in dataset D_{train}.

Overall, ANN architecture is divided into sequentially arranged modules: perception, brain, and control. Each of these modules consumes the output of the preceding part beside the first perception module, which retrieves the sensor perception and user inputs in the form of environment observation $o_{s,t}$.

The perception module, firstly, processes host, targets, and road data separately. The neural network parts that process the host and targets are straightforward and consist of a couple of feed-forward layers with ReLU activation. All processed objects are encoded in transformer embedding with a feature vector with size of 256 (defined as a parameter called 'transformer_emb_dim' in Tab. 4.3).

The module which processes road observation is built based on the Graph Neural Network and is similar to the approach presented in [98, 14]. We chose the Graph Neural Network to process roads because the complex road structure with assigned logic and rules is commonly depicted as a graph in some form. For example, data in Open Street Map [51] (OSM), HD maps, GIS-related data [99, 100], etc. uses vector representations of connected nodes, which is an example of the graph structure. In the current work, the lane markers are depicted as a sequence of ten nodes evenly spaced along the marker, forming a polyline. This can be seen in Figure 4.1. Each node consists of x, and y coordinates in VCS and encoded delimiter type.

A polyline represented as a direct graph is processed by the subgraph module (Fig. 4.8). Firstly subgraph processes each point with a fully connected layer. The size of this layer is defined as 'subgraph_width' and equals 64. The output is normalized with Layer Normalization and activated with the ReLu activation function. Then graph layer conducts message propagation [101], which concatenates the representation of

connected nodes doubling its size. Connections are encoded in the 'edge_index' feature vector which is included in state observation. Using the same approach as was used for the single point, the extended representation of the edge is processed once again. The process is repeated three times (defined by parameter: 'num_subgraph_layers'). Finally, the operation returns a feature vector with dimensions $6 \times 10 \times 24$. Then, as PointNet architecture proposed [94], the most significant polyline representation is selected using the scatter max operation. Selected feature vectors are further processed with a fully connected layer whose output size equals 256, matching the size of the representation of other objects.

Encoded representations of the host, lane markers, targets, and one additional trainable input token go to the Transformer Encoder layer [93]. The selection of the transformer architecture was based on its capability to handle different numbers of inputs making it suitable for our application where there are varying numbers of legitimate targets and lanes in each sample. Since the shape of the input tensor should be constant from a software perspective, the current architecture requires providing mask vectors that inform the transformer which inputs to disregard.

The transformer is defined according to PyTorch documentation [102]. It consists of 3 layers (param 'transformer_num_layers'), 2 heads ('transformer_num_heads'), and the size of the feedforward layer ('transformer_ff_dim') is 512. The module is trained to recognize the relation between embedded scene features and encodes meaningful representation of the entire scene in input token (as applied in [103]). The feature output associated with the input token should represent all relevant details about the traffic scene in the latent state.

The brain module has a form of a Long-Short Term Memory layer (LSTM) [97] with a hidden size of 128. The LSTM layer should learn to remember important features from past states, encoded by the perception module, that are crucial for the control task. The LSTM layer passes the embedded state of the current situation and convolutes it with the hidden memory state. It decides which part of the information to remove and which to keep for upcoming interference. It produces the feature vector, which goes to the control module, and the new hidden state, which is preserved for the subsequent network inference.

The control module depends on the action space definition of the experiment. The control head suitable for acceleration action outputs the parameters of the Normal Distribution: mean value μ and the natural logarithm of standard deviation $\ln(\sigma)$. The mean value is output by the linear neural layer and activated with $\tanh(x) = \frac{\sinh(x)}{\cosh(x)} = \frac{e^x - e^{-x}}{e^x + e^{-x}}$ function in the range $\mu \in [-1, 1]$. The $\ln(\sigma)$ could take the value of any real number $\ln(\sigma) \in \mathbb{R}$ and is a standalone input-independent trainable parameter. This parameter is initialized as 0 and it decreases during the training. By decreasing it limits agent exploration [69, 104] and increases its confidence in selected action.

As mentioned above, the proposed architecture of the Neural Network is parameterized, which results in a various number of trainable weights θ. The strongest emphasis is put on the perception module because of its role in processing input data into a meaningful representation. The major hyperparameters of ANN architecture are presented in Table 4.3 and include those which influence the size of the transformer: 'transformer_emb_dim', 'transformer_num_heads', 'transformer_num_layers', 'transformer_ff_dim'; and subgraph: 'subgraph_width', 'num_subgraph_layers'. The parameter values, that were used in the following experiments, were selected by conducting several sessions of training with different values of major

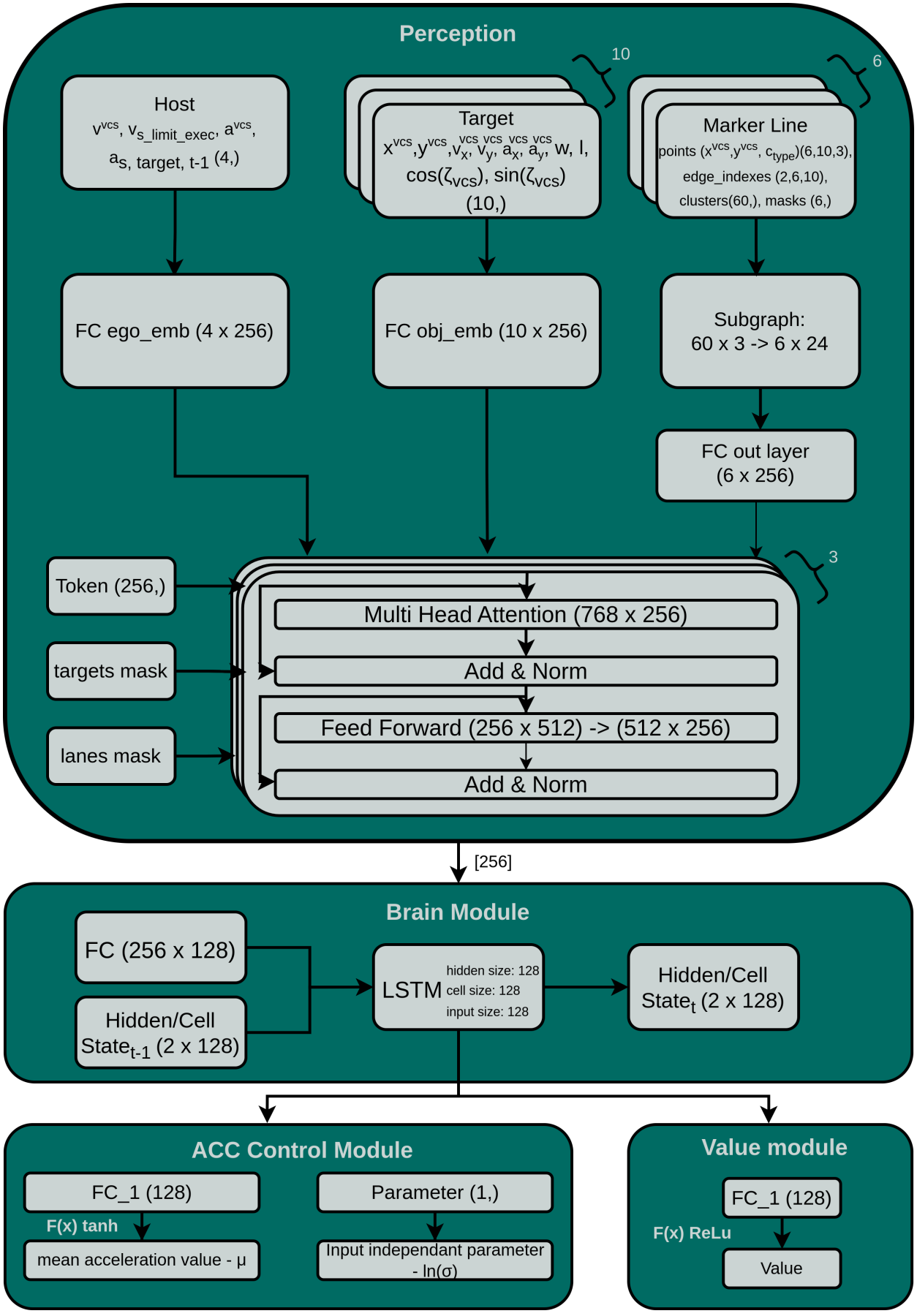


Figure 4.7: ANN consists of 3 major modules: perception, brain, and control. The subgraph is shown in detail in Fig. 4.8

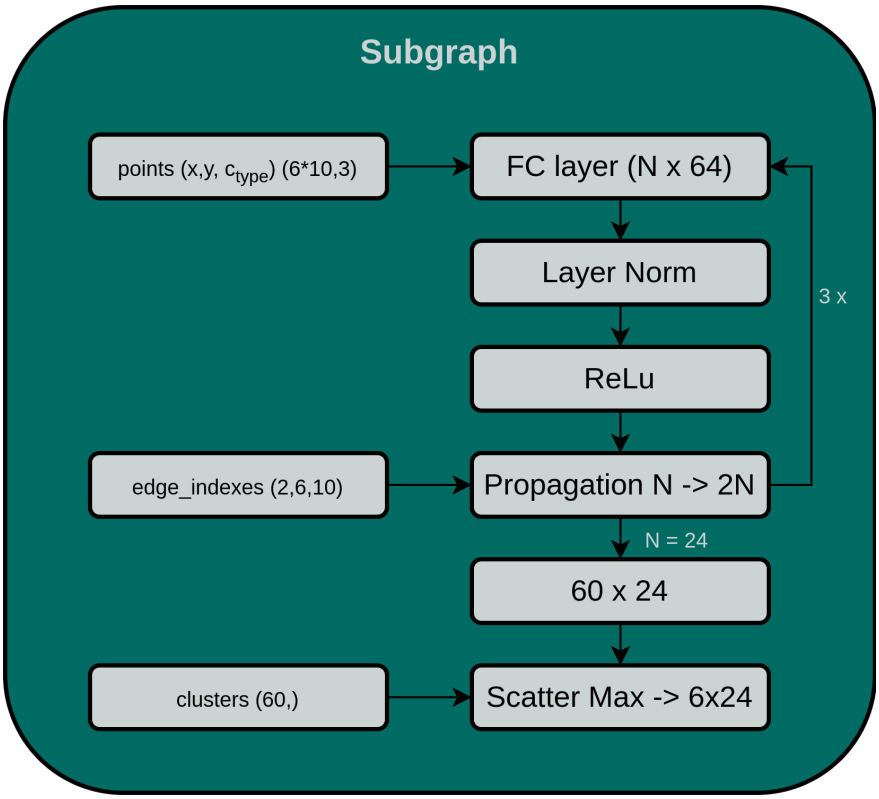


Figure 4.8: Subgraph is a part of ANN (Sec. 4.3) which processes the lanes geometry which is represented as a directed graph (compare with Fig. 4.1)

parameters. I specified four different sets of parameters that increased the number of trainable weights. The consecutive networks were called adequately ‘small’, ‘medium’, ‘default’, and ‘huge’. Table 4.3 shows selected parameter values for given networks as well as the number of trainable parameters θ and inference time of NN on modern GPU. Figure 4.9 presents the course of training from the perspective of the mean sum of reward in episodes.

After thoroughly analyzing the training results, it is evident that the ‘small’ architecture fell short in terms of performance and can be disregarded. The ‘medium’ and ‘default’ architectures have similar training progress, although the ‘medium’ network took longer to reach the reward level compared to the ‘default’ architecture. Additionally, the high variance in the mean reward sum during training indicates instability in the learning process. In comparison to other networks, the ‘huge’ network achieved a desirable reward level quickly. However, it experienced a decline in performance, as illustrated by the chart, and failed to reach the reward level attributed to the default network. Furthermore, the huge network’s inference time is approximately four times longer than that of other networks. Based on this experiment, the default parameters were chosen for future experiments due to their superior performance and training stability.

4.4 Assumptions and Limitations

The thesis intends to provide a Neural Network policy that may be feasible for preset target acceleration values for trajectory generation modules in highway environments in order to fulfill the objective of Adaptive

	huge	**default**	**medium**	**small**
num_subgraph_layers	4	3	3	2
subgraph_width	128	64	32	16
transformer_emb_dim	768	256	192	128
transformer_ff_dim	1024	512	384	256
transformer_num_heads	3	2	2	1
transformer_num_layers	4	3	2	1
inference time [ms]	178.43	15.81	9.29	4.25
trainable parameters	16,058,231	1,761,727	761,695	286,147

Table 4.3: Different parameter values of the Neural Network were examined in order to determine the most appropriate for the rest of experiments

.

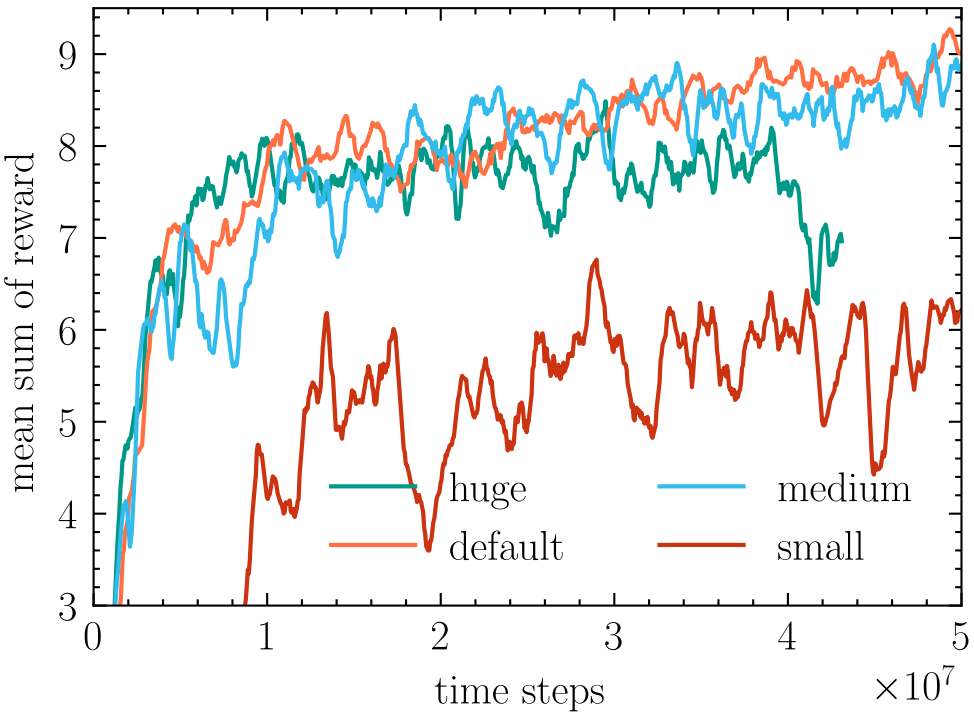


Figure 4.9: The mean of rewards collected during the training of neural networks parameterized accordingly to Table 4.3.

Cruise Control. Experiments assume driving at speeds in the range of 0 to 35 m/s. Accordingly, traffic flow generated in simulation scenarios covered a wide range of scenarios from low traffic flow to traffic congestion, and the solution is tested in such conditions.

It is essential to recognize that the dataset collected for training and validation consists only of logged data from test drives. The driving was conducted in company vehicles equipped with a sensor stack explicitly designed for the market platform. Driving episodes in which the average speed was lower than 8m/s were filtered out to ensure that driving occurred on freeways and not in urban. The presented solution does not apply to in-town driving. However, the proposed architecture of the ANN was designed to cover complex urban road topology, as the project coverage is intended to be extended with such scenarios.

Testing on the actual car in natural conditions could not be performed due to the absence of the vehicle, which consists of all crucial automated driving components. Moreover, it was impossible to fulfill all criteria required to perform such testing, resulting from the Traffic Law Act, Section 6 'The use of roads for research work on autonomous vehicles' [105].

Due to limited computing infrastructure, our training experiments were conducted using a cluster of computers with a single GPU and a maximum of 80 CPUs per experiment. That influenced the training process limiting the batch size and capacity of the ANN. As could be noticed in the literature, the success of RL training is directly connected to the volume of computing infrastructure. For reference, a state-of-the-art solution such as [106] utilized 128000 CPUs and 256 P100 GPUs for distributed training lasting 180 days. We expected that the course of training and performance of outcome policy may differ while using a much more powerful infrastructure.

It should be noted that the presented solution is intended to work in real-time on a testing vehicle on selected computing hardware. That influences the size of the observation, the number and size of ANN layers, and the computational complexity of creating observation from sensor readings.

Chapter 5

Methodology

The chapter presents the approach for training an Adaptive Cruise Control (ACC) policy using both online and offline reinforcement learning methods. The presented approach focuses on developing a policy that can handle natural driving scenarios. The study is based on simulation and real driving logs obtained from a test vehicle. The latter serves as a training dataset (Sec. 4.2) and baseline for evaluation of the agent's performance.

The novel methods presented below address the challenges discussed in the missing factors section. (Sec. 1.6). First of all, Section 5.2 presents the method for alleviating the issue of suboptimality of human actions which are included in the training dataset. The method is adapted to the ACC problem and relies on the numerical optimization of actions performed by the driver. The section demonstrates the preliminary results of the presented algorithm by comparing policies trained on the optimized and original samples.

In the following Section 5.3, the aforementioned technique is incorporated into optimizing the dataset created based on driving logs. The resulting dataset and the original one are then utilized for training agents with Behavioral Cloning (Sec. 3.2.2) and MARWIL (Sec. 3.2.3) algorithms. The cross-comparison of trained agents is presented in the next chapter in Section 6.2.

The outcome best-performing agent is then fine-tuned with the PPO algorithm in simulation. This step aims at increasing the distribution of known situations and reducing the extrapolation error of the agent trained initially on limited data. Having in mind that tuning in simulation may introduce the issue of catastrophic forgetting which would result in forgetting knowledge of real data. To address this issue, Section 5.5 presents the method of online training in simulation with the incorporation of real data. This is done by extension of the simulated scenarios with the resimulation of driving logs. This approach allows for constantly refreshing the real states during training and further optimizing the initial policy with new situations.

For further comparison of the policy that resulted from training with invented methods, Section 5.1 presents the standard way of training the baseline agent in simulation with the PPO algorithm.

5.1 Baseline PPO Training

This section covers the standard training process of the PPO agent (Sec. 3.1.14). The policy that is obtained through this training will be used as a reference to test the hypothesis (Sec. 1.3). It will be done by comparing the baseline policy with that obtained through the proposed methods.

The Neural Network presented in Section 4.3 was trained from randomly initialized weights in a simulated ACC environment (Sec. 4.1). The training was performed using the RLlib framework [107] in version 1.12.1. The process used 80 distributed parallel CPU workers for collecting experience and a single trainer, which performed policy optimization on GPU. The training lasts around 12M steps, and collected experience equals almost 74 days of constant driving while traveling approximately 144038km. Table 5.1 consists of the hyperparameters selected for this training. Parameters are named accordingly to RLlib [107] documentation and relate to algorithm PPO [68].

The primary parameter gamma γ, which directly specifies planning horizon (3.5), was set to 0.99. The parameter γ affects the number of future steps and rewards the agent takes care of. Considering the relation between gamma and time horizon (3.7) we may notice that with a gamma of 0.99, the 69th future step is half as important as the first step. It corresponds to step number 7 in case of $\gamma = 0.9$ and step number 693 when $\gamma = 0.999$. Previous experiments showed that a gamma of 0.99 is feasible for such a task where the horizon could not be too far due to the prediction of the future state. Neither should be too short to avoid fluctuating actions, which is the case when the planning horizon is too limited.

The parameter λ smooths the training process by reducing the variance of calculated advantage (3.31). It was set as suggested in the original work [68] to 0.95. Such value leverages both short-term and long-term actions.

The clip parameter and KL coefficient parameters may be used to restrict the policy updates in such a way that the new one does not deviate much from the older one in gradient space which results in a more stable training process. As a result, the performance of PPO policy does not collapse or increase unexpectedly during the training, as is the case with unconstrained policy optimization steps such as DQN 3.1.7 or SAC [75].

In the initial version of [68], the utilization of Kullback-Leibler divergence was suggested, with the KL coefficient parameters serving as controls. This method measures the distance between two probability distributions, and in the context of PPO, it is utilized to restrict excessive policy changes during optimization. In contrast to the previous approach, the updated version of PPO [68] utilizes a clipping mechanism (3.29) to limit the loss value within the defined range of clip parameter (ϵ parameter in equation 3.30). In conducted experiments, the second approach was selected and ϵ set to 0.2. This value appears to provide adequate stability during training without being overly limiting.

The next set of parameters specifies train batch size, SGD minibatch size, shuffle sequences, and the number of SGD iterations. These parameters determine the process of sample collection and optimization. The train batch size parameter defines the number of samples to collect before starting the optimization round (Alg. 5). The training batch is then divided into mini-batches based on the SGD minibatch size parameter. Optimization is performed on each mini-batch one by one, and one round of optimization uses all the samples in the batch. The number of SGD iterations parameter specifies the number of such rounds.

The trainer creates the mini-batches separately for each round, randomly selecting from the batch if shuffle sequences are enabled. The size of the mini-batch is limited by the CUDA memory since the GPU processes all the samples simultaneously.

The below charts (Fig. 5.1) show the progress of the optimization process concerning the reward function and its individual components (compare to the definition of reward function 4.1.4). Additionally, KPIs are calculated (Tab. 6.3) which show the agent performance in detail.

As training progresses, the parameter of the control module in NN $ln(\sigma)$ (Fig. 4.7) decreases which causes the narrower distribution of selecting action (Fig. 5.1). This process suppresses exploration due to the fact the agent is becoming more confident of its actions.

Parameter	**Value**
gamma γ	0.99
lambda λ	0.95
batch mode	complete episodes
train batch size	50000
SGD minibatch size	1000
shuffle sequences	True
number of SGD iterations	15
grad clip	3.0
KL coefficient	0.0
clip parameter	0.2
entropy coefficient	0
learning rate	1e-4

Table 5.1: PPO hyperparameters used for training of baseline PPO agent. Naming accordingly to applied RLlib library [107].

The agent is further evaluated using defined methods (Sec. 6.2) and compared to other agents to see if the use of real data is cost-effective in terms of the feasibility of the agents under natural conditions (Tab 6.2).

5.2 Improving Experts' Experience

In order to alleviate the issue of data imperfection (Sec. 1.7), which affects Offline RL algorithms (Sec. 5.4), I propose a method called Optimization-based Imitation Learning (ObIL) [108]. The idea regards optimization of the experts' actions u_e which are frequently suboptimal due to human factors and the inability to perfectly predict future traffic situations, which is crucial for selecting appropriate control signals. The method proposes to use gradient optimization methods to improve experts' actions, and then use them in the offline training process. The optimization is done separately for each trajectory $(s_t, u_{e,t}, r_t, s_{t+1}) \in \tau$ included in the training dataset D_{train}. The dataset which included optimized trajectories is denoted D_{opt} (optimized dataset).

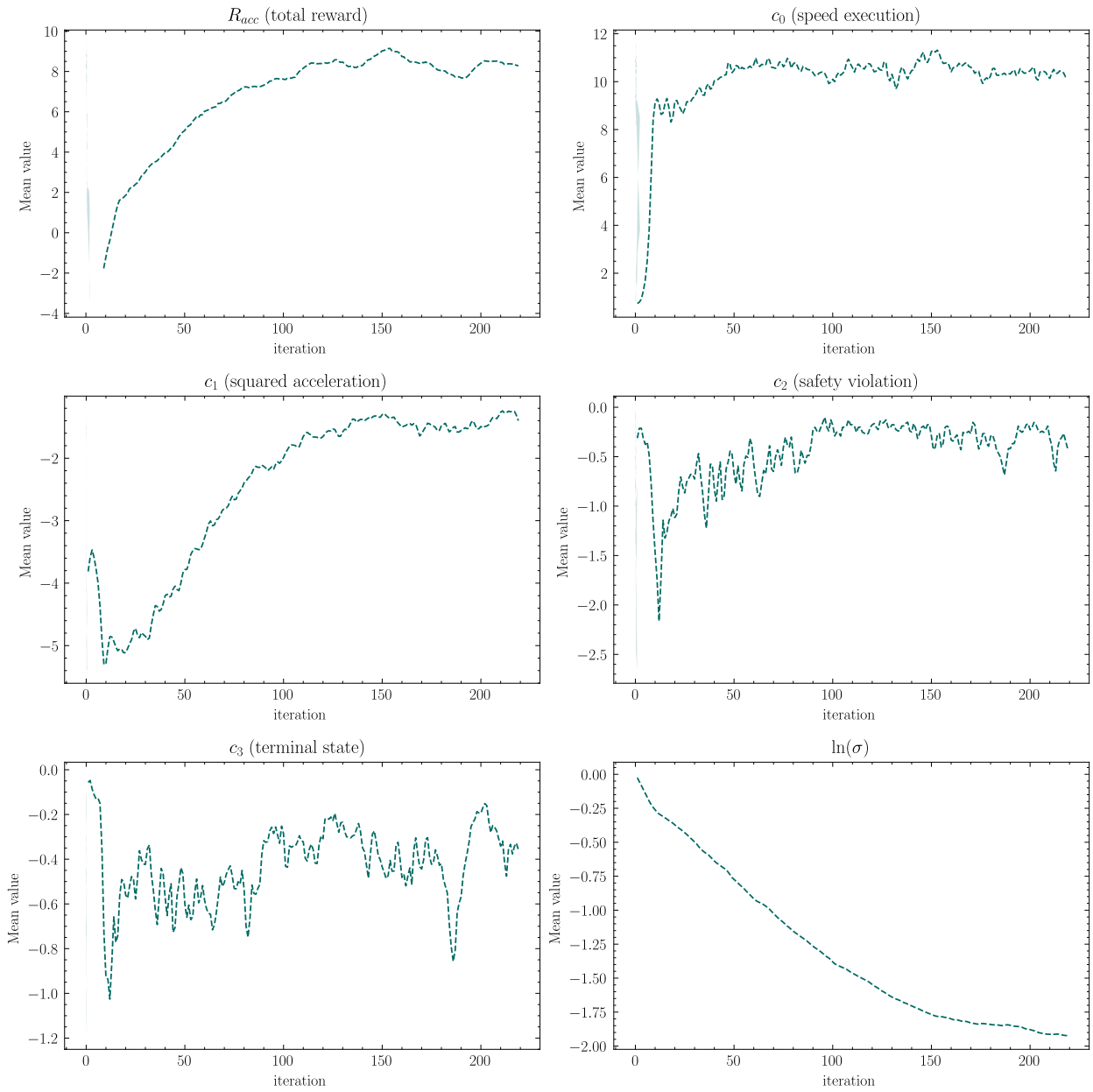


Figure 5.1: The course of training baseline PPO agent in terms of collected rewards.

The idea assumes its feasibility in the case when the agent's actions influence only the agent's state and affect the rest of the environment in a negligible way. To recalculate the environment state, based on adjusted actions, the method requires knowing the function $\psi(s_t^{\mathrm{v}}, \vartheta_t, \delta_t, b_t) \rightarrow s_{t+1}^{\mathrm{v}}$ (2.2), $k(s_{i,t}^{\mathrm{v}}, a_{s,\mathrm{target}}, \rho)$ (2.10) and $O(s_t, u_{t-1}) \rightarrow o_{s,t}$ (2.11). These functions calculate the next vehicle state s_{t+1}^{v} and observation resulting from action u_t executed in the state s_t.

If the accurate model of environment dynamics is already known $\Psi(s_{t+1}|s_t, u_t)$ (2.12), the method may evolve to multi-hypothesis trajectory optimization with a model-based solution such as Monte Carlo Tree Search [109] or [110, 111, 112].

The effectiveness of this method is based, in the first place, on leveraging the assumption that all states in the trajectory are already known, therefore, the motion prediction of adjacent vehicles is no longer nec-

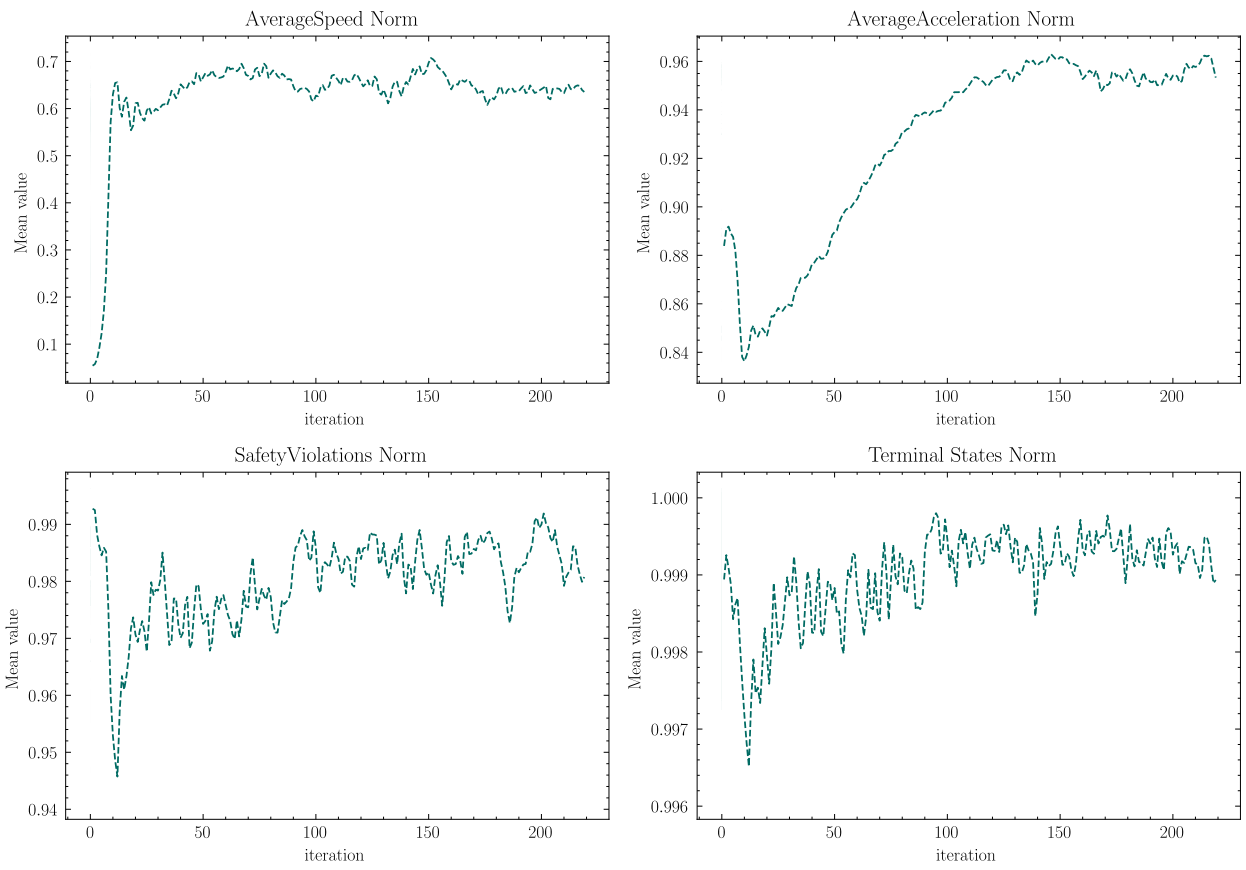


Figure 5.2: The course of training baseline PPO agent in terms of performance of rewards terms (6.3), (6.5), (6.17), (6.18).

essary and prediction imperfection does not degrade action selection. Secondarily, the method utilizes the knowledge of the function $\psi(s_t^{\mathrm{V}}, \vartheta_t, \delta_t, b_t) \rightarrow s_{t+1}^{\mathrm{V}}$, which approximates the environment state when actions differ from the original ones. In the given timestamp, it only changes the state of the controlled vehicle while keeping the state of the other vehicles as they were originally.

The idea behind this approach to offline learning is to provide the trajectories with enhanced actions that would benefit the agent training. This is grounded on the assumption that the agent will be able to recognize patterns in the data and utilize them as a basis for learning, ultimately exceeding the performance of the human expert. The following section showcases the efficacy of this technique by conducting a distinct experiment using simulated data. The outcome of the experiment provides evidence which supports the hypothesis.

5.2.1 Methodology

The method assumes that the expert trajectory τ contains the set of actions that could be parameterized by the vector x that contains n parameters, which are subject to optimization. The optimization process aims to minimize the cost function, concerning the function $\psi(s_t^{\mathrm{V}}, \vartheta_t, \delta_t, b_t)$ and defined constraints. To obtain the optimal action the following optimization problem should be solved:

$$\begin{aligned}
&\min_x f(x) = \sum_{j=1}^{m} w_j c_j(x) \\
&\text{subject to constraints: } g(x) \geqslant 0 \text{ defined in Eq. (5.10-5.12)} \\
&\text{where } c_j \textit{ for } j = 0 \dots 2 \\
&\text{denotes cost terms defined in Eq. 5.4-5.6} \\
&\text{weighted by predefined weights} \\
&w_j \text{ for } j = 0 \dots 2
\end{aligned} \tag{5.1}$$

In order to optimize the actions included in the dataset for ACC agent training, it is required to define a differentiable acceleration function parameterized with vector x ex. $f_{\mathrm{bsf}}(x,t)$, utilize functions $\psi(s_t^{\mathrm{v}}, \vartheta_t, \delta_t, b_t)$ (2.2),and $k(s_{i,t}^{\mathrm{v}}, a_{s,\mathrm{target}}, \rho)$ (2.10), and select the optimization method.

For a given experiment, acceleration is expressed as a continuous BSpline function (5.2). A B-spline function of degree k is defined recursively based on control points. The B-spline function of degree k is denoted as $b_{j,k;h}$, where i is the index of the control point and h is the parameter within the knot span.

$$\begin{aligned}
&f_{\mathrm{bsf}}(x,t) \to a_{s,t} \\
&f_{\mathrm{bsf}}(x,t) = \sum_{j=0}^{n-1} x_j b_{j,k;h}(t) \\
&\text{where:} \\
&b_{j,k;h} \text{ is B-spline basis functions of degree } k \text{ and knots } h
\end{aligned} \tag{5.2}$$

The function is parameterized by n coefficients knots, one for every second of the trajectory and degree $k = 4$. The spline function was selected as a basis for acceleration curvature because it allows the representation of the solution in a smooth, differentiable form and reduces the nonlinearity of the cost function compared to the polynomial representation.

The cost function incorporates three main cost terms in order to achieve a trajectory that is desired from the perspective of a perfect adaptive cruise controller:

1. Maximizing speed limit execution

   This term is introduced to achieve optimized actions that make full use of available speed limitations. At the same time, this term and related additional constraints result in avoiding exceeding the speed limit or stopping a car on the road.

$$v(x,t) = \int_0^t f_{\mathrm{bsf}}(x,\tau)\, d\tau + v_0 \tag{5.3}$$

$$c_0(x) = (v_{\max} T - \int_0^T v(x,t)\, dt)^2 \tag{5.4}$$

2. Minimizing the velocity changes

   This term induces the resulting acceleration function to be as flat as possible thereby it minimizes the acceleration and jerk values over the trajectory. Such minimization increases the passengers' comfort and reduces fuel consumption. Additional constraints restrict the function to be in the range of maximal and minimal acceleration.

$$c_1(x) = \int_0^T f_{\text{bsf}}(x,t)^2 \, dt \tag{5.5}$$

3. Maximizing time in the sensible range to the leading target

   The ACC objective assumes that the distance to the leading target should be in an optimal span. The minimal range value depends on the current speed of traffic participants and their maximal possible values of acceleration and deceleration according to RSS rules (Sec. 2.5). Regarding the maximum range value, it should not be too high to avoid possible cutting in of other cars, nor too small to leave space for the agent's maneuvers. We assume that the maximum distance equals the maximal sensor range to keep tracking the target.

$$s(x,t) = \int_0^t v(x,\tau) \, d\tau \tag{5.6}$$

$$c_2^{\text{cond}}(x,t) = \begin{cases} 0, & \text{if } \Delta s_{\min} \leqslant |s^{\text{obj}}(t) - s(x,t)| \leqslant \Delta s_{\max} \\ 1, & \text{otherwise} \end{cases} \tag{5.7}$$

where $s^{\text{obj}}(t)$ is a distance travelled by leading vehicle and $\Delta s_{\min/\max}$ is maximal/minimal distance to following vehicle. The complete cost term is calculated for episode time:

$$c_2(x) = \int_0^T c_2^{\text{cond}}(x,t) dt \tag{5.8}$$

As a primary optimization method, I selected a Sequential Least Squares Programming Optimizer (SLSQP). It optimizes the f_{bsf} function of n parameters by minimizing the cost function (5.1). The optimization method is selected due to its suitability for problems where the cost function and the constraints are twice continuously differentiable.

Additionally, optimization is constrained by inequality constraints to enforce physical feasibility.

1. Acceleration constraint assumes that values of f_{bsf} in all valid time steps are in a range of -3.5, 1.5:

$$g_0^{\text{cond}}(x,t) = \begin{cases} 0, & -3.5 \leqslant f_{\text{bsf}}(x,t) \leqslant 1.5 \\ 1, & \text{otherwise} \end{cases} \tag{5.9}$$

$$g_0(x) = \int_0^T g_0^{\text{cond}}(x,t) dt \leqslant 0 \tag{5.10}$$

2. Velocity constraint function to achieve velocity values in given range $v_s \in [0, 35]$:

$$g_1^{\text{cond}}(x,t) = \begin{cases} 0, & 0 \leqslant v(x,t) \leqslant 35 \\ 1, & \text{otherwise} \end{cases} \tag{5.11}$$

$$g_1(x) = \int_0^T g_1^{\text{cond}}(x,t)dt \leqslant 0 \tag{5.12}$$

5.2.2 Proof of Concept

In order to test the hypothesis, the experiments were conducted on a simulated version of an Adaptive Cruise Control (ACC) environment (Sec. 4.1). The simulation was used to collect a dataset of expert trajectories. The expert manually selects the target acceleration value using the same interface as the agent to meet the expectations of a perfect ACC controller. As the major function of ACC is to follow the leading target, the target was present and tractable most of the time. In order to make the task more challenging, the leading target behaved unstably and oscillated around the acceleration set point in specific scenarios. This caused a significant variation in the target's speed.

The trajectories in the dataset were optimized using the SLSQP optimization method and SciPy programming library for scientific computing [113], with respect to the objective function defined in Equation (5.1). Figures 5.3-5.6 depict the course of one of the trajectories, presenting the acceleration, speed, and jerk of the vehicle controlled by the expert and the target vehicle, as well as the distance between them. Additionally, the corresponding plots for the optimized acceleration are also visible.

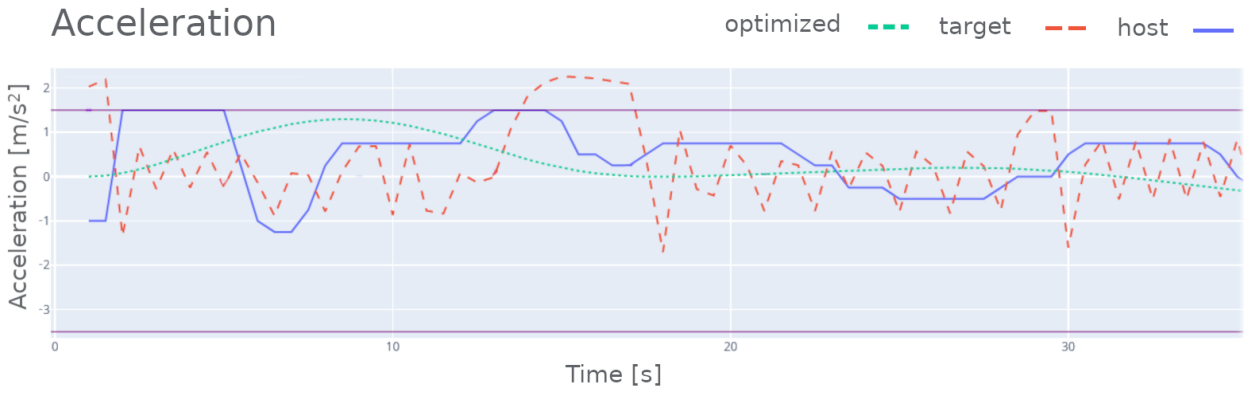


Figure 5.3: The acceleration of the leading target (red), human driver (blue), and optimized one (green). The horizontal lines represent the limit values of acceleration (-3.5, 1.5 m/s^2).

Subsequently, the optimized trajectories were utilized for training the ANN, which serves as the behavior policy for the ACC agent. To accomplish this, the MARWIL algorithm, which is currently state of the art in the field of Imitation Learning (Sec. 3.2.3), was employed.

To assess the effectiveness of the presented approach in comparison to the typical Imitation Learning approach, I conducted a baseline training session using the original trajectories as the training dataset. All other training parameters were kept consistent with the previous case.

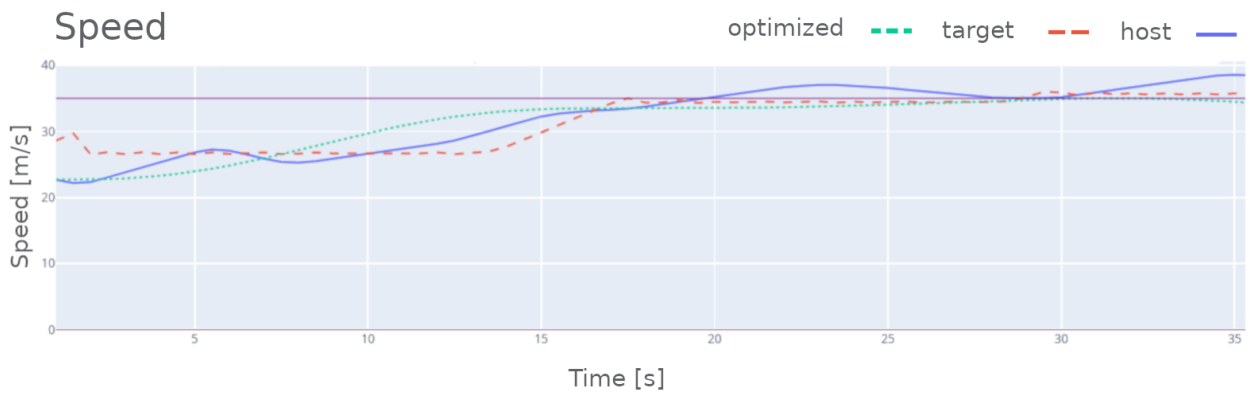


Figure 5.4: The speed of the leading target (red), the human driver (blue), and the speed calculated from the optimized spline function which represents the acceleration function (green). The horizontal lines represent the limit values of velocity (0, 35 m/s)

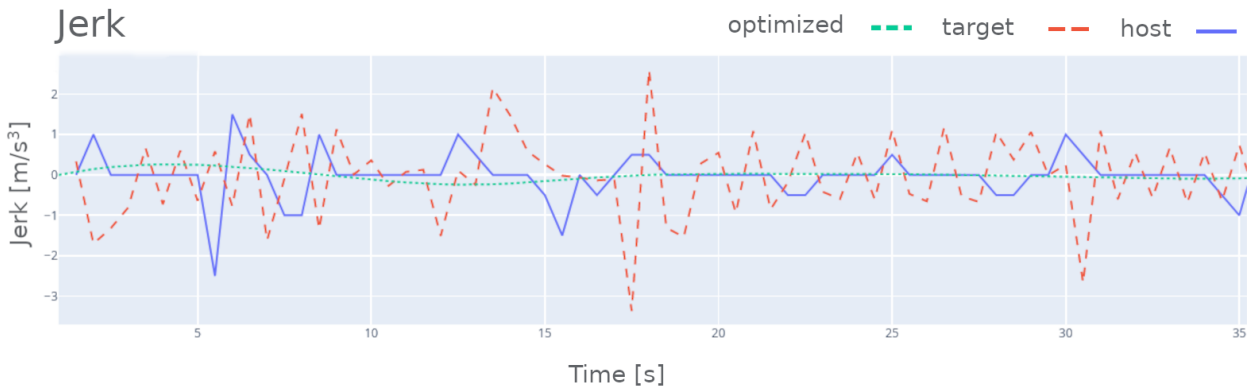


Figure 5.5: The jerk values of the leading target (red), the human driver (blue), and the jerk calculated from the optimized spline function, which represents the acceleration function (green).

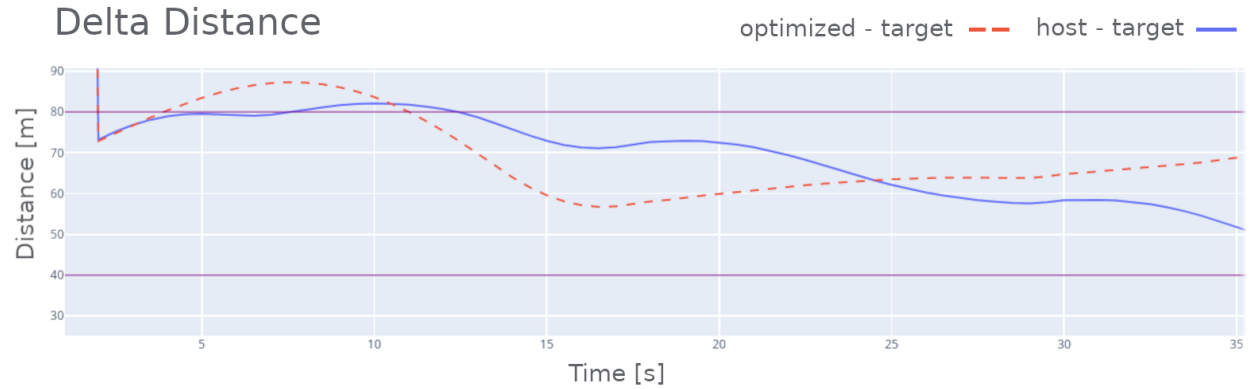


Figure 5.6: The distance between the human driver and the leading target (blue) and between the optimized state and the leading target (red). It could be noticed that the function optimization contributed to the reduction of the distance, which prevented other vehicles from cutting in. The horizontal lines represent the limit values of the distance from the agent to the leading vehicle- (40m, 80m).

Following both training sessions, I evaluated the new behavior policies in the test environment by running ten episodes. Based on the trajectories obtained, I calculated and compared the Key Performance Indicators (KPIs) as described in Section 6.1. The comparison results are summarized in Table 5.2.

As anticipated, the behavior policy generated with the usage of optimized trajectories outperformed the policy obtained from the original trajectories. The metric shows that the optimized agent drove faster and more comfortably than the baseline agent. Based on these findings, it can be concluded that this type of dataset optimization leads to a more effective policy in terms of ACC objective.

KPI	**Policy based on optimized dataset**	**Policy based on original dataset**
Average Speed $[m/s]$	20.303	13.781
Average Acc $[m/s^2]$	-0.037	-0.763
Average Abs Acc $[m/s^2]$	0.112	0.883
Average Abs Jerk $[m/s^3]$	0.343	2.719
Distance to Front Target $[m]$	137.771	140.356
Oscillation on Empty Lane Rate $[m/s^2]$	0.114	1.024
Heavy Braking Events $[\%]$	0.200	0.600
Safety Violation $[\%]$	0.400	1.100
Collisions $[\%]$	0.0	0.0

Table 5.2: KPIs values from the evaluation of two behavior policies. The center column shows the values for an agent generated with ObIL algorithm. While the right column presents the agent's results generated with original trajectories.

5.3 Optimizing Training Dataset

The method ObIL method was employed in the process of training the ACC agent based on the collected dataset D_{train} (Sec. 4.2). To optimize the dataset, firstly, I filtered out samples that did not contain any valid target which supposed to be followed by the host vehicle. In the created sub-dataset, I left only these parts of episodes in which the followed target is visible for more than 10 consecutive steps. However, I retained samples where the target vanishes for less than three successive steps. The target disappearing may result from perception issues, and such a small gap does not influence the optimization process. Moreover, since objects vanishing is a commonly encountered problem in perception, the presence of such samples allows the policy to learn this aspect of real data.

The resulting subset contains 240 episodes with about 13000 samples. Each of the trajectories was the subject of optimization which was conducted as described above (Sec. 5.2). The optimized actions were used for the recalculation of new vehicle states and observations in trajectories. The set of optimized trajectories is denoted as D_{opt}.

In order to guarantee the comparability of various trained agents across all further experiments, I matched the weights w of the objective function (5.1) with the weights of the ACC reward function (4.6), providing the common objectives of optimization (eq. 5.13):

$$J = -0.11c_0(x) - 0.02c_1(x) - 0.3c_2(x) \tag{5.13}$$

Additionally, this experiment assumed that a minimal distance to the following target ($\Delta s_{\min}$ in Eq. (5.7)) should be calculated according to the longitudinal rule of RSS (Sec. 2.5, Eq. 2.19).

The example of the real episodes and optimization results are presented in the below figures (5.9 - 5.11).

To outline the optimized dataset D_{opt}, I calculated the same set of KPIs as for the original dataset D_{train} (Sec. 4.2). The comparison table is presented below (Tab. 5.3), the same as the acceleration histogram (Fig. 5.7, compare with Fig. 4.5) and histogram of execution of speed limit (Fig. 5.8, compare with Fig. 4.6).

This dataset was split into test and train sets in the ratio of 1 to 9. The histograms of actions in these sets are presented in Figure 5.13.

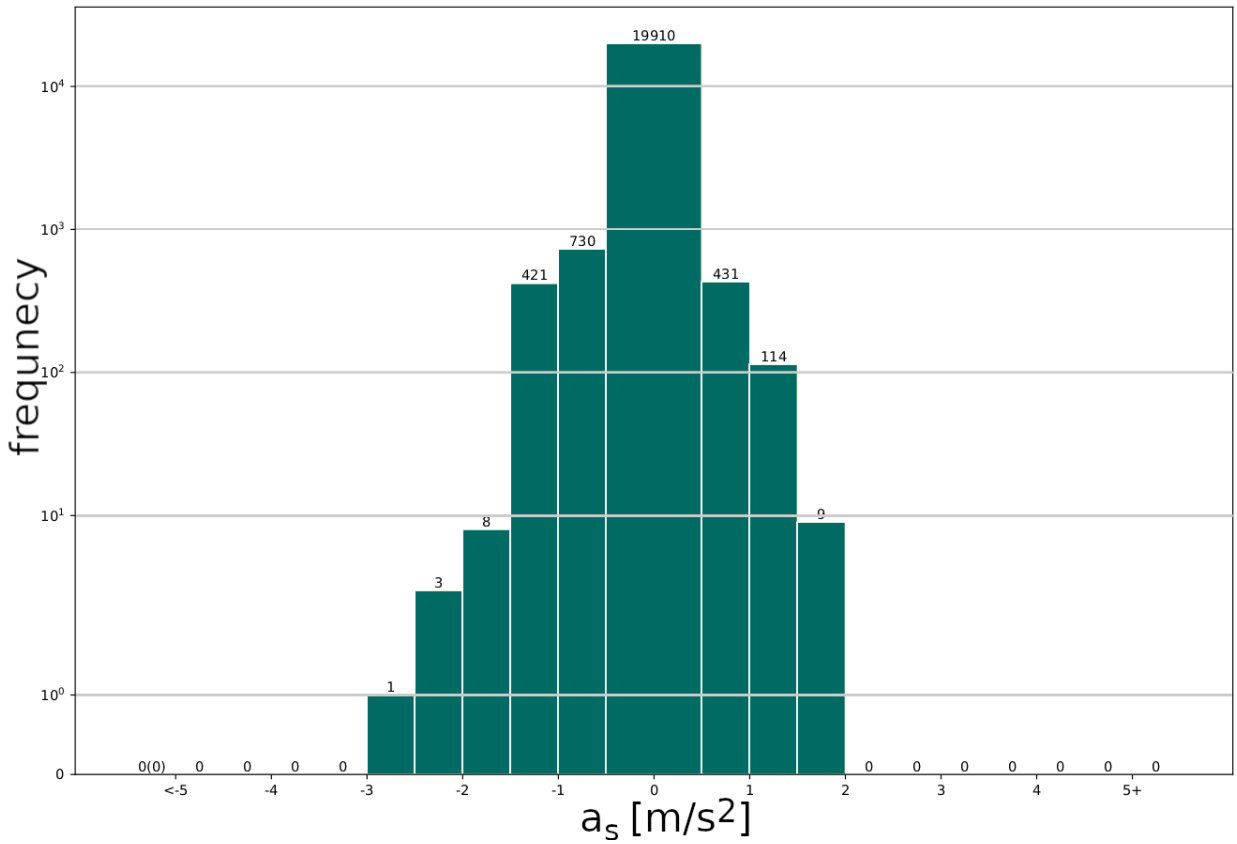


Figure 5.7: Histogram of host accelerations existed in the optimized dataset. Compare with the original distribution (Fig. 4.5)

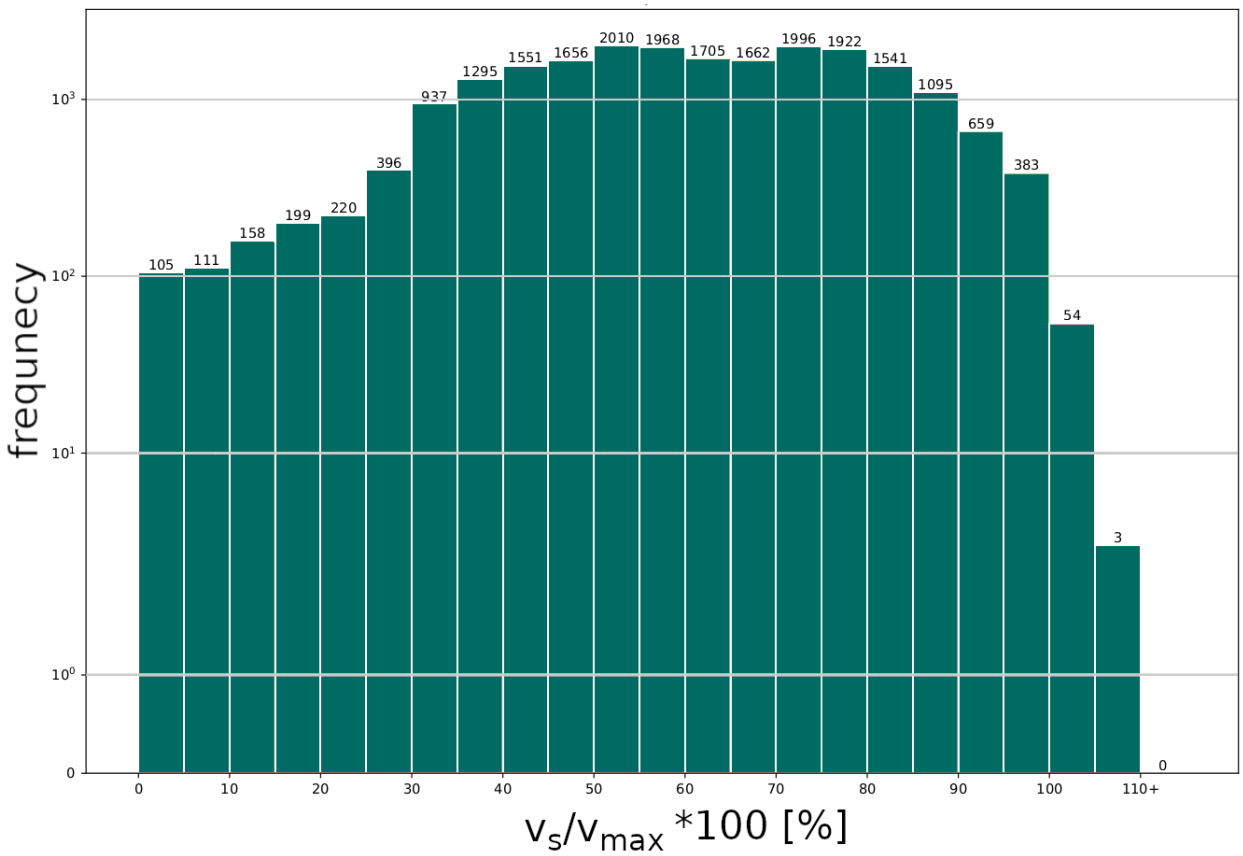


Figure 5.8: Histogram of speed limit execution ratio for all samples in the optimized dataset. Compare with the original distribution (Fig. 4.6)

Parameter	**Original Dataset**	**Optimized Dataset**	**Unit**
Distance traveled	672.211km	134.651km	km
Acceleration Mean	-0.011	-0.002	m/s^2
Acceleration Std	0.372	0.300	m/s^2
Speed wrt Speed Limit Mean	0.563	0.602	%
Speed wrt Speed Limit Std	0.196	0.195	%
Speed Mean	22.513	21.088	m/s
Speed Std	7.847	6.825	m/s
Jerk Mean	0.784	0.098	m/s^3
Jerk Std	6.707	0.229	m/s^3
Headway to front Mean	10.066	33.519	m
Headway to front Std	90.010	3764.95	m

Table 5.3: Comparison between original D_{train} and optimized dataset D_{opt}.

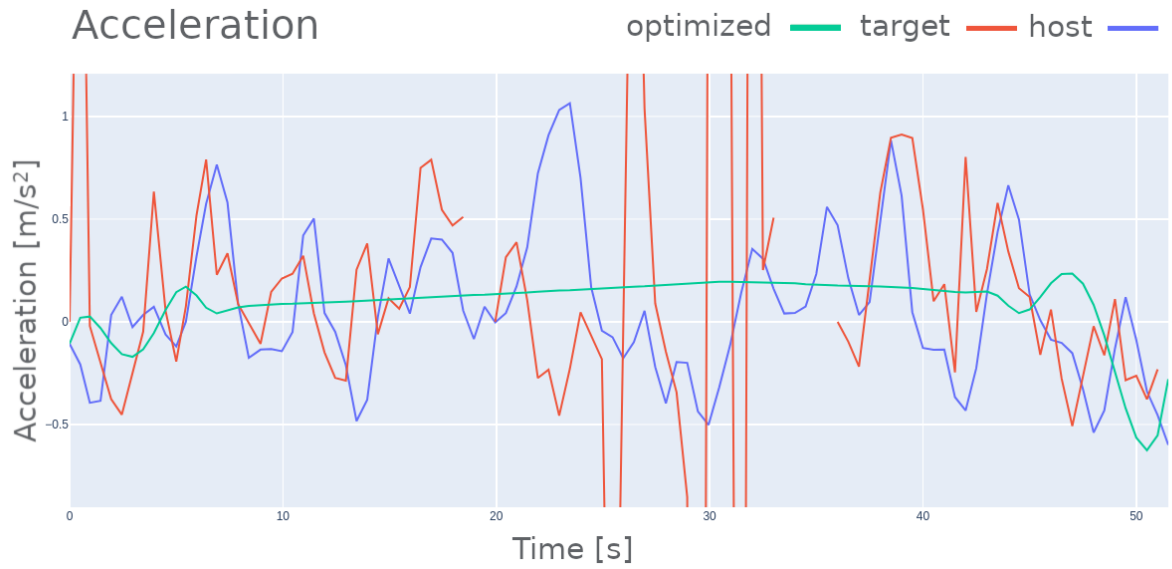


Figure 5.9: The acceleration of the leading target (red), human driver (blue), and optimized one (green).

Figure 5.10: The speed of the leading target (red), the human driver (blue), and the speed calculated from the optimized spline function which represents the acceleration function (green).

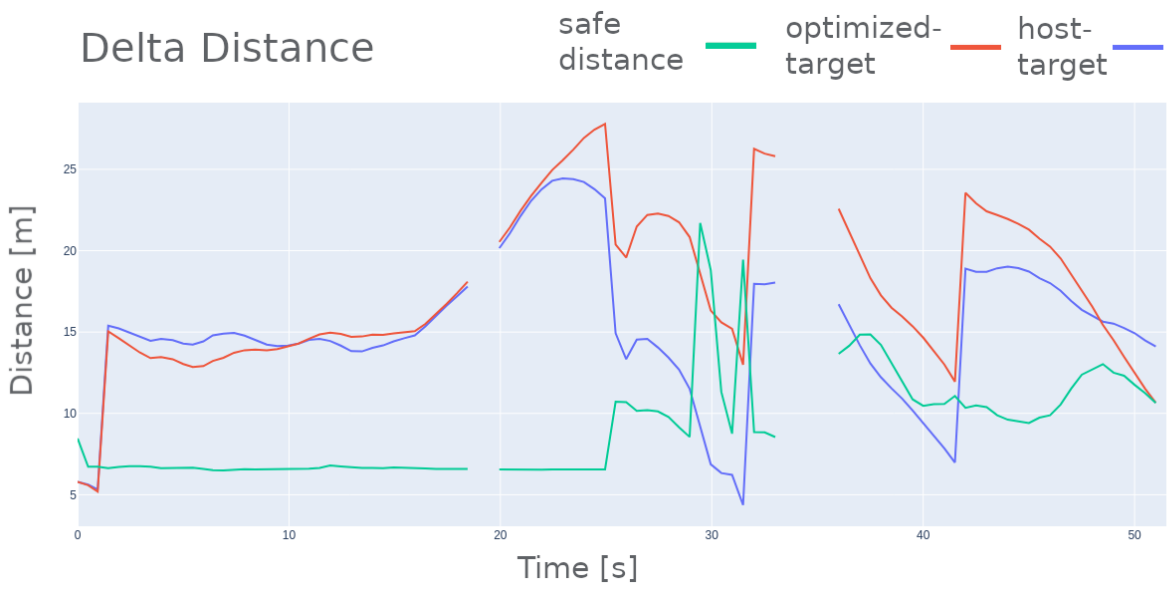


Figure 5.11: The distance between the human driver and the leading target (blue) and between the optimized state and the leading target (red). The green line shows the safe distance to the target calculated for the optimized host.

5.4 Offline Learning on Datasets

The original D_{train} and optimized dataset D_{opt} were used for training the initial driving policy. Experiments utilized Behavioral Cloning (Sec. 3.2.2) and MARWIL (Sec. 3.2.3) algorithms with additional tricks and parameter tweaking. As described in the overview of offline RL algorithms [114], Offline RL algorithms require a large amount of parameter tweaking to acquire acceptable results. As an illustration, the authors suggested utilizing dropout regularization, which is often avoided in neural network architectures for RL policies due to the exploration process resulting in a high variance in observation. Another example is conducting a thorough search for learning rate values, which may vary for the critic and actor.

All final algorithms' hyperparameters used for policy training are presented in Table 5.4. The training and testing losses for training on the original dataset D_{train} are presented in Figure 5.17 and accordingly for the optimized dataset D_{opt} in Figure 5.18.

In order to enhance policy training, a few additional improvements were implemented. The first enhancement results from the inspection of the charts of policy losses. They showed huge variance across optimization steps, which indicated that the training batch size was too small. Although memory used for ANN and batch samples already exploited the available VRAM, I implemented the accumulated gradients algorithm [115]. It allows multiplying batch size based on summing the gradients from the backward propagations of several loss calculations on consecutive mini-batches. Only after that, the optimization step is executed, and gradients are applied and reset.

The next enhancement came from the evaluation of policies which showed that policies favor too high or too low values of acceleration which caused speeding or slowing down too much. Such a problem remained a typical issue of unbalanced datasets in supervised learning. In order to alleviate that issue, I introduced

equal sampling from fixed ranges of actions (as done in preliminary work [116]). The method assumes the clusterization of samples with respect to action values in a selected number of groups. Training batch is created by sampling from each group an equal number of samples. Experiments showed that performance increases when samples are clusterized into two groups, such as the first one includes actions with the corresponding accelerations in the range of $a_s \in [-3.5, 0m/s^2]$ and the second one with remaining.

Additionally, a comparison of all optimized trajectories exposed that at the end of the episodes, the optimization tended to accelerate to minimize the velocity cost. It often accelerated to be as close as possible to safety distance in the last step of the episode. It remains the situation when a driver accelerates if it predicts an empty lane in the near future (the front target is cutting off). However, such an assumption is not correct for all episodes. To overcome this trend, either the optimization problem should be reformulated or training samples that indicate such behavior should be filtered out.

The last enhancement regards the fact that training samples should contain LSTM hidden states. The hidden states should be created by the actual version of the trained policy before each optimization step. This is due to the fact that the optimizer changes LSTM weights which influence how the layer processes information. Usually, for the training step, the dataset returns an adequate number of samples for burning hidden states. On the assumption that LSTM takes into consideration the hidden state generated on N_{hs} previous samples to get the hidden state for current prediction. The dataset returns a batch that contains $N_{\mathrm{hs}} + 1$ consecutive samples for a single learning sample multiplied by the batch size. The neural network should process all of the consecutive samples to make a single prediction, which significantly increases the computational workload. In order to minimize that number, I implemented a custom dataset structure that updates hidden states for each sample after every optimization step. It simply calculates the hidden state of samples in each trajectory starting from the first one. This decreases computing time during each training step and eliminates another algorithm hyperparameter N_{hs}. Since our policy is finally used for predicting actions in continuous episodes, a single pass for hidden state recalculation seemed appropriate.

Each trained policy was evaluated according to the presented evaluation criteria (Sec. 6.1). The comparison was conducted to select the best policy subject for further optimization in the final training step (Sec. 5.5). The calculated KPIs for final experiments may be found in Table 6.1. Regarding the sum of rewards (6.1), it turned out that the best agent was trained with the MARWIL algorithm on D_{opt} dataset.

5.5 PPO Learning with Utilization of Real Data

The model trained with MARWIL on the optimized dataset D_{opt} was further enhanced with the online RL method in simulation. Enhancement needs to be completed in order to train the agent in situations that were not included in the dataset. The simulation incorporated in ACC environment (Sec. 4.1) gives the opportunity to create an infinite number of traffic scenarios on the grounds of a randomized scenario generation process. The greater diversification of the course of episodes is also influenced by the agent's stochastic behavior, which causes various responses from simulated traffic participants.

This learning method considers also the issue of catastrophic forgetting of previously learned data. In this case, it relates to the real-world experience acknowledged during offline learning. In order to alleviate that issue, the presented method incorporates the resimulation of scenarios from logged data. The real cases

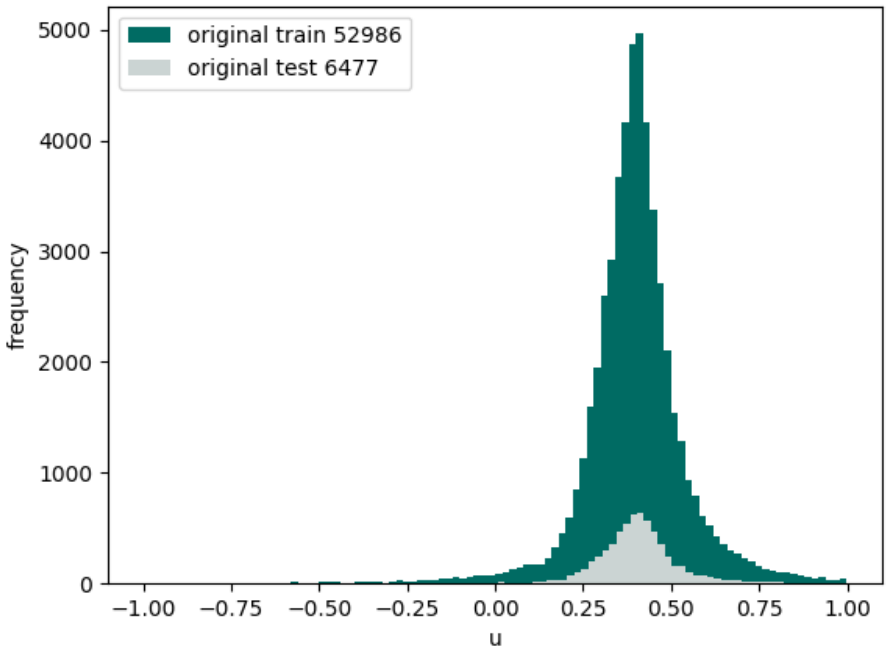


Figure 5.12: Action histogram in the original train and test dataset D_{train}.

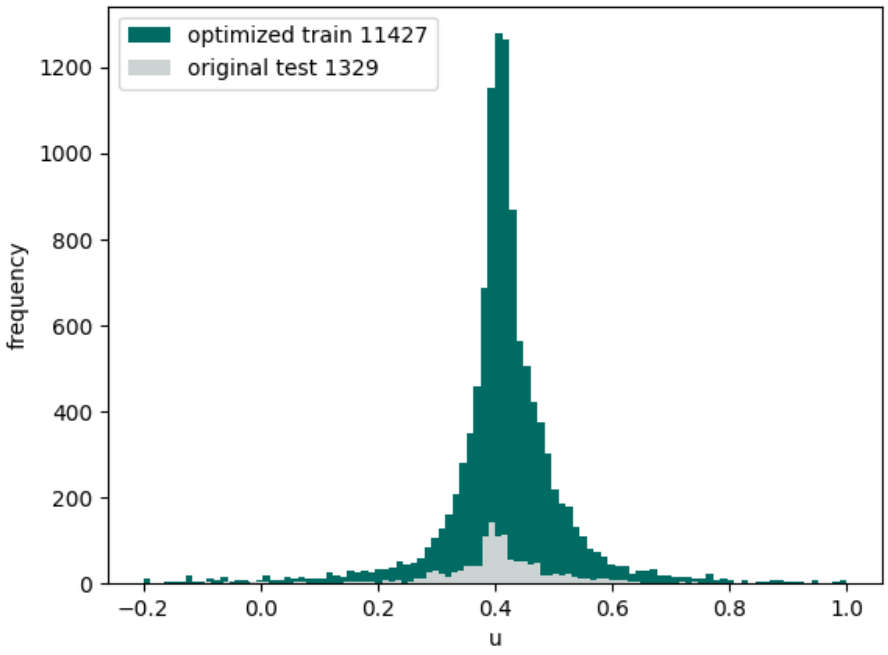


Figure 5.13: Action histogram in optimized train and test dataset D_{opt}.

parameter	value	description
batch size	300	number of samples processed simultaneously in loss function
gradient accumulations	20	number of loss function calls before optimization step
learning rate	1e-4	optimizer learning rate
MARWIL β	1	scaling of advantages in exponential terms.
evaluation	3	test loss calculated every n iteration of policy optimizations
using gae	True	if true, use the Generalized Advantage Estimator (GAE) with a value function
vf coeff	1.0	weight of value estimation loss which is added to policy optimization loss

Table 5.4: Hyperparameters used for training offline agents (BC and MARWIL) on original and optimized data.

Figure 5.14: The figure presents the acceleration of different agents, the human driver, and the optimal trajectory in a resimulated logged episode.

account for the 5th type of scenario in addition to these defined in the scenario generator (Sec. 4.1.1). With this condition, the agent may further optimize policy on unknown situations, refresh comprehended situations, and learn transitions that are close to states from the dataset. These states are crucial for optimizing behavior in situations that reveal when the agent drifts from the known trajectory.

Figure 5.15: Comparison of executed velocity between different policies evaluated on resimulated log and the human driver.

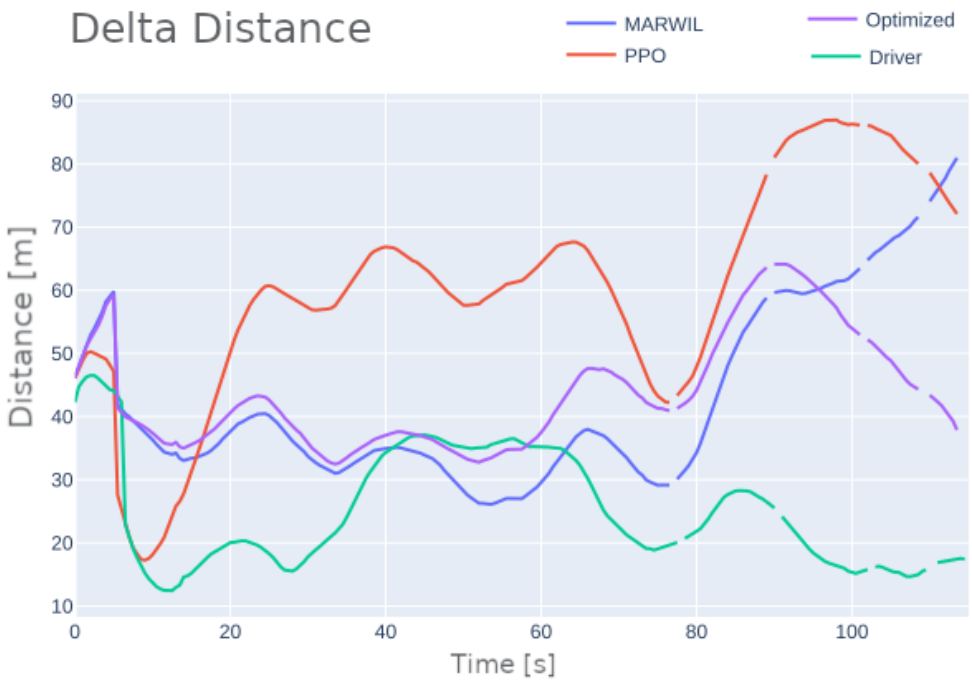


Figure 5.16: Comparison of distances to target in resimulated log for trained policies (MARWIL - blue, PPO - red), optimized trajectory (purple), and the human driver (green).

One may notice that the resimulation may not reflect the real environment accurately as it is and incorporates issues known from the sim2real gap (Sec. 3.3). It could be a correct statement, although, utilization of such samples introduces observations affected by real sensors and traffic situations, which at least slightly decreases the gap between the simulated and actual world. Combining real trajectories with simulated ones

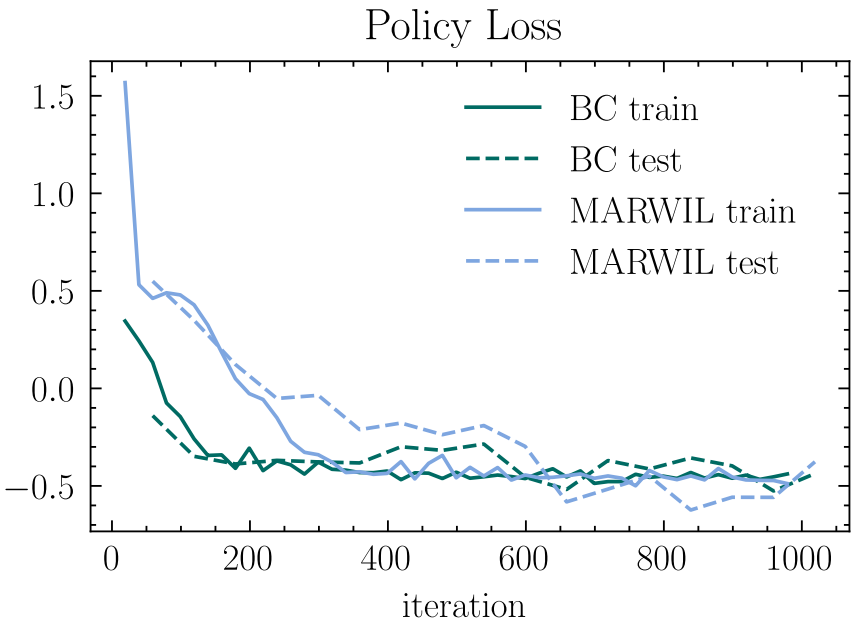


Figure 5.17: Training and testing loss of BC and MARWIL agent on original dataset D_{train}.

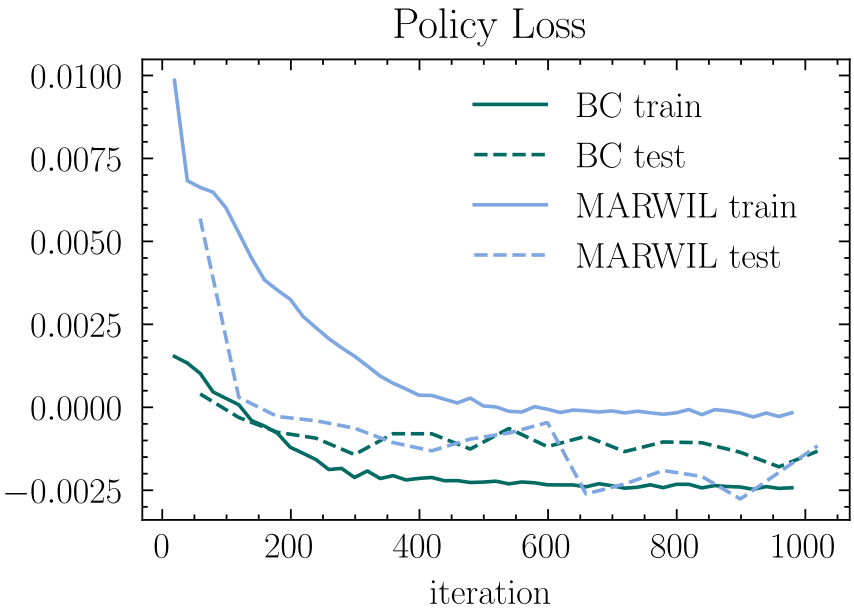


Figure 5.18: Training and testing loss of BC and MARWIL agent on optimized dataset D_{opt},

creates another approach to the sim2real gap problem alongside domain randomization or adaptation methods (Sec. 3.3.1, 3.3.2).

The presented overall method is committed to creating a vast amount of scenarios concerning the expected parameters of roads, vehicle dynamics, and driver behaviors. Identification of such parameters aims at an approximation of the distribution of scenarios into the distribution of real conditions. It would be beneficial to create scene distribution broader than could occur in the real world. That will ensure that most of the situation is covered. However, it could not be too broad to prevent policy generalization to situations that never happen, at the expense of lower quality in situations that policy will actually face. Additionally, it is

worth considering whether the policy should adapt to all driving styles that are performed around the world or focus just on local ones. Of course, this depends on the planned region of deployment and the learning capability of ANN.

In my opinion, the distribution resulting from scenario generation may not perfectly match the natural distribution. However, when combined with scenarios based on driving logs, it has the potential to move toward the target distribution.

The process of log replay is based on reproducing detected objects and road lanes in consecutive steps of simulation. Object dimensions and velocity are set according to sensor readings. The planar coordinates x, y, ζ, which originally was reported in Vehicle Coordinate System (VCS), are transformed based on compensation of host movement function (5.14):

$$\begin{aligned} \zeta(t) &= \phi_0 + t\omega + 0.5t^2\alpha \\ v(t) &= v_0 + ta + 0.5t^2\dot{a} \\ \begin{pmatrix} x(t) \\ y(t) \end{pmatrix} &= \begin{pmatrix} x_0 \\ y_0 \end{pmatrix} + \int_0^t v(t) \begin{pmatrix} cos(\zeta(t)) \\ sin(\zeta(t)) \end{pmatrix} dt \end{aligned} \tag{5.14}$$

where ϕ-azimuth; ω-host angular speed; α-angular acceleration;

The same relates to the position and types of detected marking lines. Whereas the host vehicle drives alongside the origin lane, its position depends on acceleration selected by the agent. The acceleration is given to the trajectory generation module (2.10), performed in simulation by a built-in dynamic vehicle model. It is expected that the position of the simulated host vehicle would differ from the original one. However, the observations are recalculated with respect to a new position, actual velocity, and acceleration by $O(s_t, u_{t-1}) \rightarrow o_{s,t}$. The discrepancy between the original and executed positions may induce some deception in the overall process. This is due to the fact that, the closer the agent is to the original position, the more realistic the simulation is. If the agent drives farther, the perception should detect other objects that actually are not visible in a given frame. Moreover, by the fact that the driver at the back should pay attention to the leading vehicle and not cause a collision, the objects detected behind the host in its lane are not taken into consideration. This is to avoid collisions that would not be the fault of the agent and not to terminate the simulation prematurely.

The training aimed at fine-tuning the agent trained previously on MARWIL on the optimized dataset. For that purpose, I applied the PPO algorithm and ACC environment with the aforementioned log resimulation. This additional set accounted for 25% of scenarios. The training parameters were the same as specified for the baseline PPO agent (Sec. 5.1) and are presented in Table 5.1. The training lasted for 400 epochs, and the course of training in the context of rewards optimization is visible in Figure 5.19. Inspecting it, we may see that the initial sum of the reward was higher than in baseline ppo training (Fig. 5.1), which was expected because the process started on a pre-trained policy. The training started achieving a reward of 5, which was impacted most by the rewards for acceleration ($c_1 = -2$) and for speed limit execution ($c_0 = 9$). However, the training progress differed from the baseline, primarily noticeable on a chart of c_0 reward component. It is possible that this is due to significant changes in the distribution of scenarios, which leads to different

samples being collected. The second factor could be due to the use of a pre-trained policy that focused on reducing acceleration as a priority.

Moreover, the resimulated scenarios forced the agent to drive safer to avoid collision and safety violations. This conclusion was drawn on close inspection of episodes collected by the baseline PPO agent. It revealed that the agent developed an aggressive strategy of fast approaching the leading car to force it to change a lane. This situation occurred on a relatively empty highway when the agent drove fast. It exploited the behavior model of simulated agents in its favor. However, such a situation does not occur in driving logs, forcing agents to drive safer.

The final agent from this training is referred to as 'PPO on MARWIL' and its evaluation results are presented in Section 6.2.

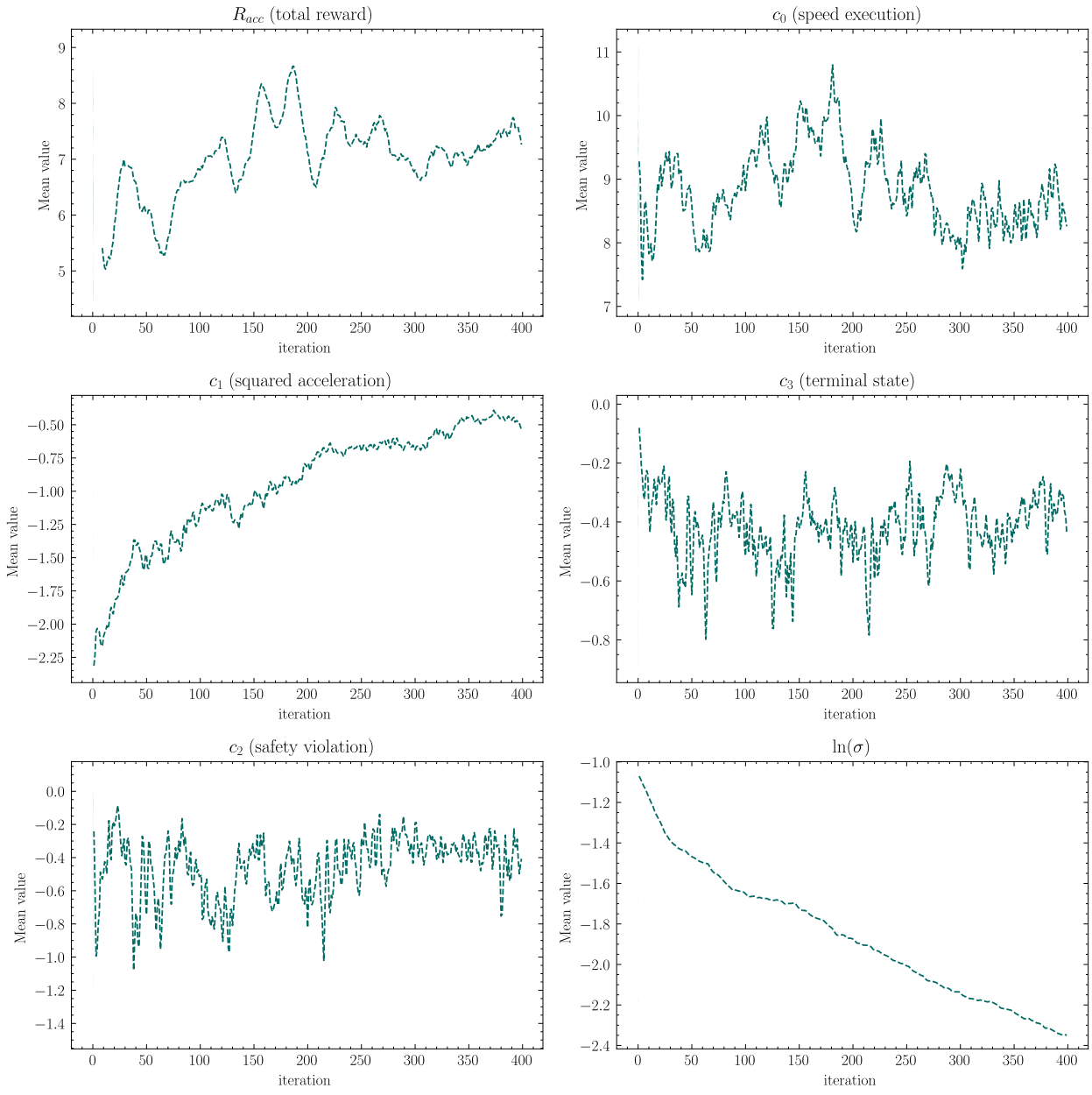


Figure 5.19: The plots of collected rewards during fine-tuning of MARWIL policy and ANN parameter $\ln(\sigma)$. The training was conducted on the ACC environment enhanced with scenarios created based on replaying of driving logs.

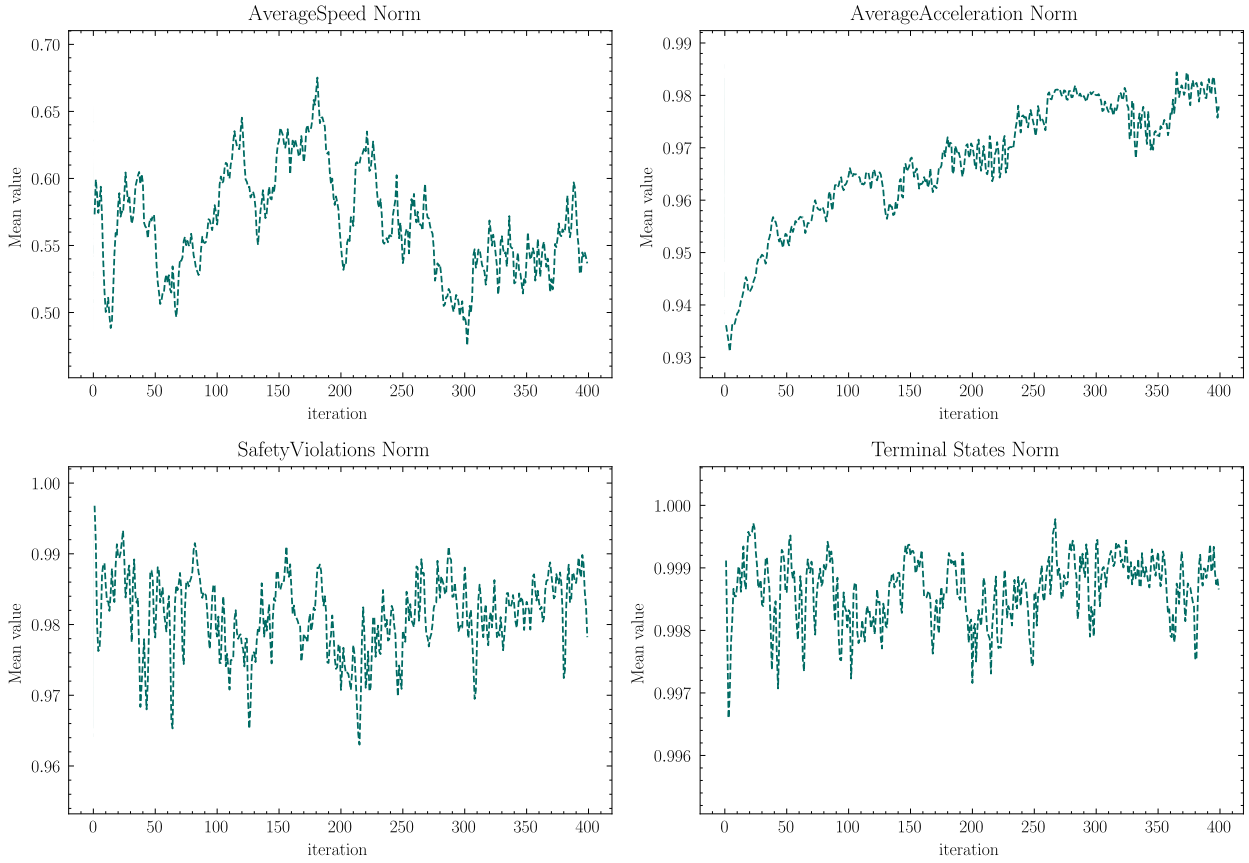


Figure 5.20: The plots of performance of reward terms during fine-tuning of MARWIL policy (6.3), (6.5), (6.17), (6.18)

Chapter 6

Solution Evaluation

6.1 Evaluation Criteria

To compare and evaluate the performance of all previously trained agents the following methodologies were applied. As a key performance metric, the mean sum of rewards calculated by the reward function was considered:

$$R_{\text{eval}}(s^{\text{v}}_{\text{host}}, s^{\text{v}}_{\text{target}}) = \frac{1}{T * E} \sum_{e=1}^{E} \sum_{t=0}^{T-1} (R_{\text{acc},e,t}(s^{\text{v}}_{\text{host}}, s^{\text{v}}_{\text{target}})) \tag{6.1}$$

where $R_{\text{acc},}(s^{\text{v}}_{\text{host}}, s^{\text{v}}_{\text{target}}))$ is defined in 4.1.4 (4.6), T is the number of time steps in an episode, and E is the number of testing episodes. In order to deeply understand what impacts the sum of the reward, during the training, I tracked the mean value of each term of the reward which may be seen in Figures 5.1 and 5.19.

The defined set of Key Performance Indicators (KPIs) strictly measures the agent's performance in various aspects of driving that may not be reflected in the reward function. In a phase of designing the reward function, it helps to understand the course of learning and allows for shaping the multicriterial reward function [117] by changing its weights and terms to maximize the value of all KPIs. Employing them allows for comparing different strategies in different aspects. It also permits for discarding of agents that do not fulfill basic driving assumptions. It is common that agents achieve high rewards by exploiting reward hacking or other tricks commonly encountered in RL experiments [118, 119]. The KPIs were calculated during the training, showing the course of training (Fig. 5.1, 5.19) and afterward in the evaluation phase.

The set of KPIs considers features related to the speed, execution of speed limit, acceleration, jerk, violation of safety distance (Sec. 2.5), heavy braking events, and related metrics. Additionally, most KPIs are accompanied with its norm $\|\text{KPI}\|$

$$\text{AverageSpeed} = \frac{1}{T} \sum_{t=1}^{T} (v_{s,t}) \tag{6.2}$$

$$\|\text{AverageSpeed}\| = \frac{1}{T} \sum_{t=1}^{T} \left(\frac{v_{s,t}}{v_{\max,s}} \right) \tag{6.3}$$

$$\text{AverageAcceleration} = \frac{1}{T} \sum_{t=1}^{T} (a_{s,t}) \tag{6.4}$$

$$\|\text{AverageAcceleration}\| = \frac{1}{T}\sum_{t=1}^{T}\left(\frac{a_{s,t} - a_{s,\min}}{a_{s,\max} - a_{s,\min}}\right) \tag{6.5}$$

$$\text{AverageAbsoluteAcceleration} = \frac{1}{T}\sum_{t=1}^{T}(|a_{s,t}|) \tag{6.6}$$

$$\|\text{AverageAbsoluteAcceleration}\| = \frac{1}{T}\sum_{t=1}^{T}\left(\frac{|a_{s,t}|}{a_{s,\max}}\right) \tag{6.7}$$

$$\text{AverageAbsJerk} = \frac{1}{T}\sum_{t=1}^{T}(|\dot{a}_{s,t}|)) \tag{6.8}$$

$$\|\text{AverageAbsJerk}\| = \frac{1}{T}\sum_{t=1}^{T}\left(\frac{|\dot{a}_{s,t}|}{\dot{a}_{s,\max}}\right) \tag{6.9}$$

$$\begin{aligned} X &= \{t \in 0, 1, ..., T : [\text{leading target}_\text{t} \in \text{host lane}_t]\} \\ \text{DistanceToFrontTarget} &= \frac{1}{X}\sum_{t\in X}(s_{\text{target},t} - s_{\text{host},t}) \end{aligned} \tag{6.10}$$

$$\|\text{DistanceToFrontTarget}\| = \frac{1}{X}\sum_{t=X}\left(\frac{s_{\text{target},t} - s_{\text{host},t}}{s_{\text{max,perception}} = 150}\right) \tag{6.11}$$

$$X = \{t \in 0, 1, ..., T : [\text{leading target}_t \notin \text{host lane}_t]\} \tag{6.12}$$

$$\text{OscillationOnEmptyLane} = \frac{1}{X}\sum_{t\in X}(a_{s,t}) \tag{6.13}$$

$$X = \{t \in 0, 1, ..., T : [\text{leading target}_t \in \text{host lane}_t]\} \tag{6.14}$$

$$\text{AverageSpeedToTarget} = \frac{\sum_{t\in X}(v_{s,t})}{\sum_{t\in X}(v_{s,\text{target},t})} \tag{6.15}$$

$$\begin{aligned} X &= \{t \in 1, 2, \ldots, T : [a_{s,t} < -2]\} \\ \text{HeavyBrakingsEvent} &= \frac{n(X)}{T} \end{aligned} \tag{6.16}$$

$$\begin{aligned} X &= \{t \in 1, 2, \ldots, T : [\text{leading target}_t \in \text{host lane}_t]\} \\ \text{SafetyViolations} &= \frac{1}{X}\sum_{t=X}(s_{\text{target},t} - s_{\text{host},t} < d_{\text{lon\_min},t}) \\ \|\text{SafetyViolations}\| &= \frac{1}{T}\sum_{t=X}(s_{\text{target},t} - s_{\text{host},t} < d_{\text{lon\_min},t}) \end{aligned} \tag{6.17}$$

where $d_{\text{lon\_min},t}$ calculated accordingly to (2.19).

$$\|\text{TerminalStates}\| = \frac{1}{T}\sum_{t=1}^{T}(c_3(\Delta s, v_s)) \tag{6.18}$$

where $c_3(\Delta s, v_s)$ is R_{acc} component defined as 4.5.

To calculate all of these metrics, the evaluation phase after each training was performed. In that phase, the agent does not explore the environment. It means that action is the value which ANN returns as mean acceleration value μ (Fig. 4.7).

The evaluation phase consists of the following steps. The first evaluation phase relies on a testing agent behavior on a number of randomized episodes (usually 100 episodes) in the exact environment definition, which was used for training. However, the distribution of the basic types of scenarios is uniform.

The second step of testing includes running the agent in resimulated real logs. Resimulation is performed in an open-loop manner which means that agent actions are executed, but the trajectories of other vehicles remain original (as described in Sec. 5.5). This stage is crucial for determining agent performance under real-life conditions.

The final testing stage relies on running the agent in a set of predefined scenarios that verify its response in typical road situations. The scenarios are scripted in order to make sure that agents always meet the same situations. It allows for comparing agent behavior to the expected one. If the agent does not fulfill expectations and significantly differs from the expected behavior, the agent is either discarded or its training is resumed. In such a case, we assume that optimization is stuck in the local optimum and the agent could not explore more profitable actions. Additionally, such scenarios allow for close inspection of agent behavior patterns. Such inspection guides agent development and allows reward tuning and adjusting training scenario distributions. The test set comprises scenarios with varying numbers of lanes, objects, and movement patterns. A detailed description of the scenarios is described below (Sec. 6.4).

After all stages, the process calculates KPIs that can be compared across agents based on their values and performances. For ultimate comparison, we consider the mean sum of reward achieved by the agent in resimulated real scenarios, which states the final metric.

6.2 KPI

The following sections present evaluation results for all trained policies and a complete evaluation conducted for the final agent (PPO on MARWIL). The Tables 6.1, 6.2, and 6.3 consist of calculated KPIs defined in previous Section 6.1. Table 6.1 and 6.2 include the most important results regarding agent performance in natural conditions, as they refer to the resimulation of test drives. Table 6.1 includes metrics for policies trained only with offline learning methods, and Table 6.2 comprises adequate metrics for the final policy, baseline PPO agent, and the human driver. The last Table 6.3 provides a comparison between the final policy, MARWIL trained on the optimized dataset, and baseline PPO agent. KPIs are calculated based on 100 simulated episodes.

Beneath, I analyze the individual values of performance indicators and what influenced their outcome. Besides the KPIs, Section 6.4 presents the performance of the final agent in scripted scenarios, which tests the basic functionality of ACC.

KPI	**MARWIL on D_{opt}**	**BC on D_{opt}**	**MARWIL on D_{train}**	**BC on D_{train}**
Speed execution squared (4.2)	3.190	3.465	3.133	3.344
Squared acceleration (4.3)	-0.160	-1.200	-0.203	-0.267
Safety violation (4.4)	-0.581	-0.702	-1.545	-0.715
Terminal State (4.5)	-1.702	-1.702	-3.404	-1.915
Sum of rewards (6.1)	0.747	-0.140	-2.020	0.447
Average Speed [m/s]	20.254	15.808	22.031	20.529
AverageAcc [m/s^2]	-0.088	-0.099	-0.005	-0.069
AverageAbsAcc [m/s^2]	0.249	0.572	0.304	0.318
AverageAbsJerk [m/s^3]	0.572	0.447	0.761	0.776
DistanceTo FrontTarget [m]	85.621	114.268	69.835	83.875
AverageSpeed ToTarget [%]	0.958	0.803	1.017	0.965
HeavyBraking Event [%]	0.000	0.012	0.000	0.000
SafetyViolation [%]	0.052	0.042	0.119	0.067
Collisions [%]	0.255	0.213	0.489	0.277
CauseCollisions [%]	0.170	0.170	0.340	0.191
NotCauseCollisions [%]	0.085	0.043	0.149	0.085
Oscillation0n EmptyLane [m/s^2]	0.210	0.419	0.233	0.253

Table 6.1: Comparison of KPIs calculated for all offline trained agents for re-simulated logged episodes. It estimates the performance of a given agent in real conditions.

KPI	**PPO on MARWIL**	**MARWIL on D_{opt}**	**PPO**	**driver**
Speed execution squared (4.2)	3.846	3.190	3.860	3.819
Squared acceleration (4.3)	-0.367	-0.160	-0.662	-0.456
Safety violation (4.4)	-0.166	-0.581	-1.506	-0.313
Terminal State (4.5)	0.000	-1.702	-2.128	0.000
Sum of rewards (6.1)	3.313	0.747	-0.436	3.050
AverageSpeed [m/s]	18.341	20.254	21.679	21.285
AverageAcc [m/s^2]	-0.124	-0.088	-0.021	-0.005
AverageAbsAcc [m/s^2]	0.312	0.249	0.482	0.303
AverageAbsJerk [m/s^3]	0.513	0.572	0.884	0.306
DistanceToFrontTarget [m]	94.511	85.621	65.951	48.930
AverageSpeedToTarget [%]	0.860	0.958	1.003	1.000
HeavyBrakingEvent [%]	0.024	0.000	0.015	0.004
SafetyViolation [%]	0.008	0.052	0.116	0.010
Collisions [%]	0.021	0.255	0.319	0.000
CauseCollisions [%]	0.000	0.170	0.213	0.000
NotCauseCollisions [%]	0.021	0.085	0.106	0.000
Oscillation0nEmptyLane [m/s^2]	0.261	0.210	0.390	0.395

Table 6.2: KPIs calculated for all agents trained in online mode for re-simulated logged episodes and for drivers who collected those episodes.

6.3 Testing on Logs

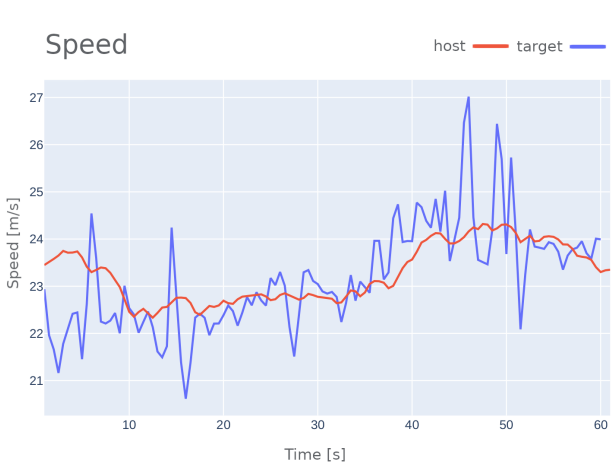
According to Table 6.2, the sum of rewards (6.1) indicates that the best performance under real conditions is achieved by an agent trained simultaneously on logs and simulations (PPO on MARWIL). The final agent achieved a value of the absolute average acceleration of $0.312m/s^2$, which is comparable to human driver $0.303m/s^2$. It is higher than MARWIL policy - $0.249m/s^2$ - which was directly trained on ground truth optimized trajectories. In terms of reward component c_1, the final agent scored -0.367, the human driver -0.456, and the MARWIL policy -0.160. Baseline PPO policy stands out from others with the highest value of $0.482m/s^2$ and accordingly c_1 =-0.662. The same regards to average absolute jerk since the latter is the first's derivative. The higher jerk may suggest that the agent frequently changes behavior from step to step, which can be seen in the attached figures below (6.4a, 6.4a, 6.6a). The outstanding metric is the distance to the front target, which suggest that trained agents tend to follow agent at a longer distance than the human driver. This may be due to the fact that the agents in the simulation get significant penalties for falling into a safe distance. The calculation of safe distance (2.19) depends on the speed of the host and leading vehicle. Therefore, if actors in simulations frequently brake, the agent will receive a penalty that encourages it to develop a safer policy.

A similar conclusion can be drawn based on average speed KPI. The final policy tends to drive more conservatively, sacrificing reward for speed in favor of avoiding penalties for safety violations and collisions. It could be confirmed by comparing rewards for safety violations and terminal states and corresponding metrics: Safety Violation and Cause Collisions. For Safety Violation, the PPO on MARWIL policy got a -0.166 score, and it drove only 0.8% steps closer to the following target than is recommended by safety distance. Most importantly, the final policy did not cause any collision in opposition to the PPO baseline and MARWIL policy. MARWIL policy may cause a crash due to the fact that it could not know how to react while the following vehicle is too close. Such a situation may occur when the driving trajectory drifts too far from familiar states, which is in the case of a near-collision situation (sec. 5.4).

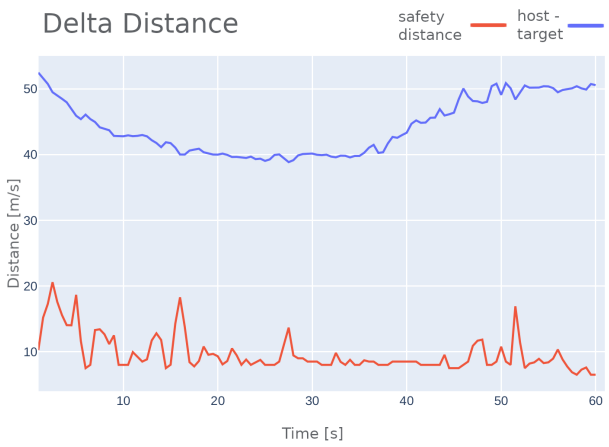
In Figure 6.1, an example driving log trajectory is presented. It is evident from the figures that the agent has acquired the skill of securely docking to a leading vehicle and following it. Furthermore, Figure 6.1a reveals that the variance of the host velocity is significantly lower than the target velocity reported by the perception system.

Figure 6.2 and 6.3 visualize the course of another driving log. It could be seen the outcome of actions performed by the final policy, and human driver, and calculated from the optimization process (Sec. 5.2).

The velocity profile of the human driver and the final policy appear similar in some respects. They both sequentially slow down, accelerate, and rapidly slow down to line up the target speed. However, the agent's actions lagged behind the driver, causing a delayed outcome. On the contrary, the optimization-based approach results in the most comfortable trajectory in terms of acceleration. It keeps an almost constant speed and does not respond rapidly to shifting target velocity.



(a) Velocity of host and target



(b) Offset between host and target and adequate safety distance.

Figure 6.1: Evaluation of final policy on resimulated driving log

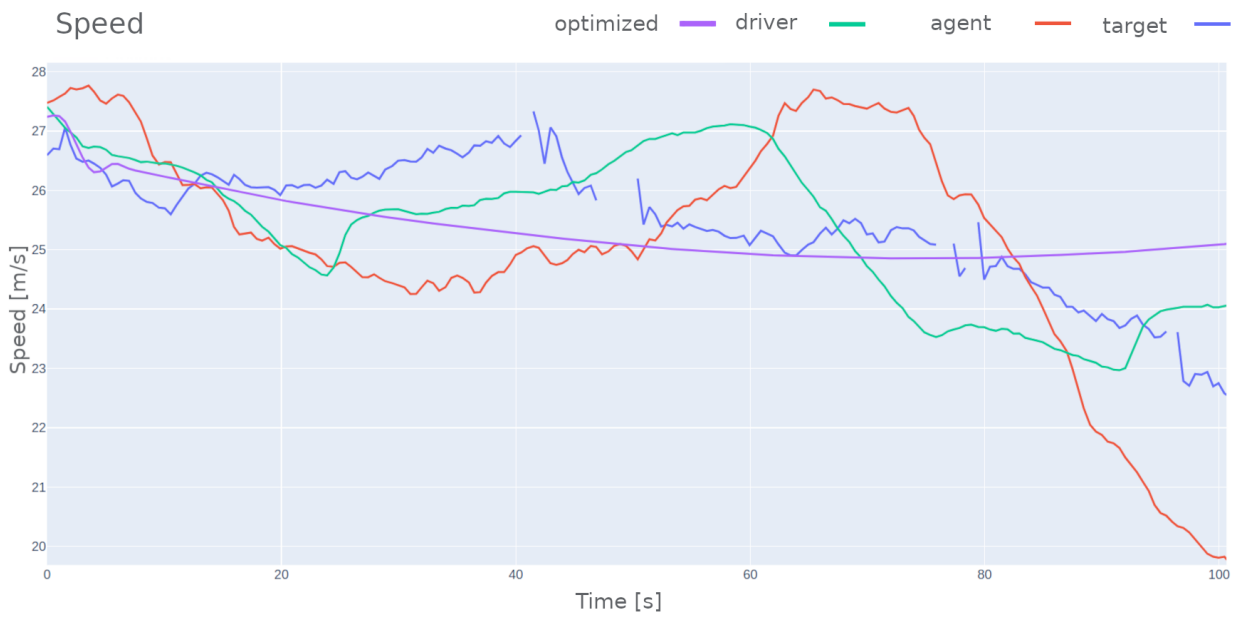


Figure 6.2: Comparison of executed velocity by different policies evaluated on resimulated log.

6.4 Closed Loop Testing in Simulation

The trained agents were evaluated on 100 simulated episodes and several predefined test scenarios. Table 6.3 presents the KPIs that were calculated based on these episodes.

The table indicates that the performance of the baseline PPO agent is better in simulation than the final and MARWIL policy. Compared with the final policy, it maximizes the reward for executing speed limits while paying less attention to the comfort of driving. It is confirmed by the value of the reward term squared acceleration c_1 and KPIs: average absolute acceleration and average absolute jerk. The final policy in simulation keeps a comparable distance to the front vehicle as other policies. However, it caused a lot of

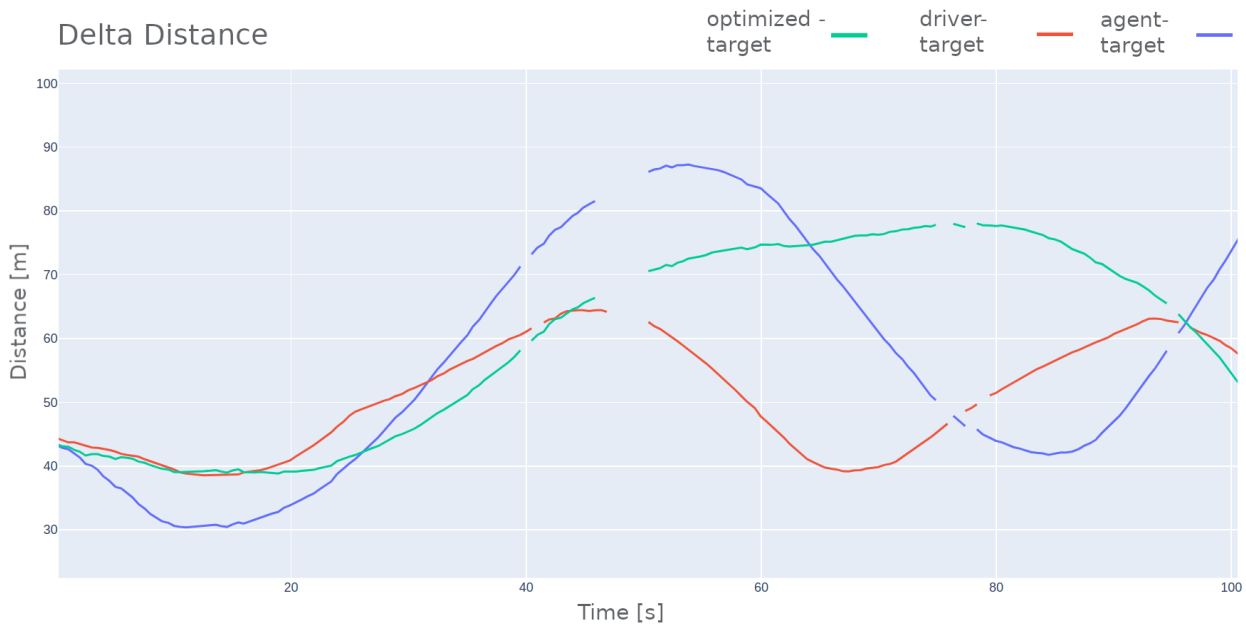


Figure 6.3: Comparison of distances to target during the evaluation of human driver (red), final policy (blue), and optimized solution on resimulated log (green).

collisions maximizing speed to target. Its speed was 140% of the speed of the following vehicles. This result is comparable to MARWIL policy which may indicate that training data did not contain part of the situation that often appeared in simulation. It probably may address problems with traffic jam situations that did not appear in logged driving.

The similarity between KPIs values for the final and MARWIL policy may suggest that the problem of catastrophic forgetting was alleviated. The final policy is driving faster than MARWIL, and average absolute acceleration metric 0.328m/s^2 is closer to MARWIL 0.218m/s^2 than to baseline PPO 0.518m/s^2. The policies that used real data for training have a similar ratio of collision (26% and 25%) which is much higher than the baseline PPO agent (3%). It may suggest that the pretrained policy was not able to explore the actions which cause heavy brakings to be generally more profitable since they preserve from collisions. The KPIs Heavy Braking Event shows that the final policy only started to explore this range of actions using it in 1% of steps. The MARWIL policy did not use any action in such range since the training dataset does not include any of them (Fig. 5.13).

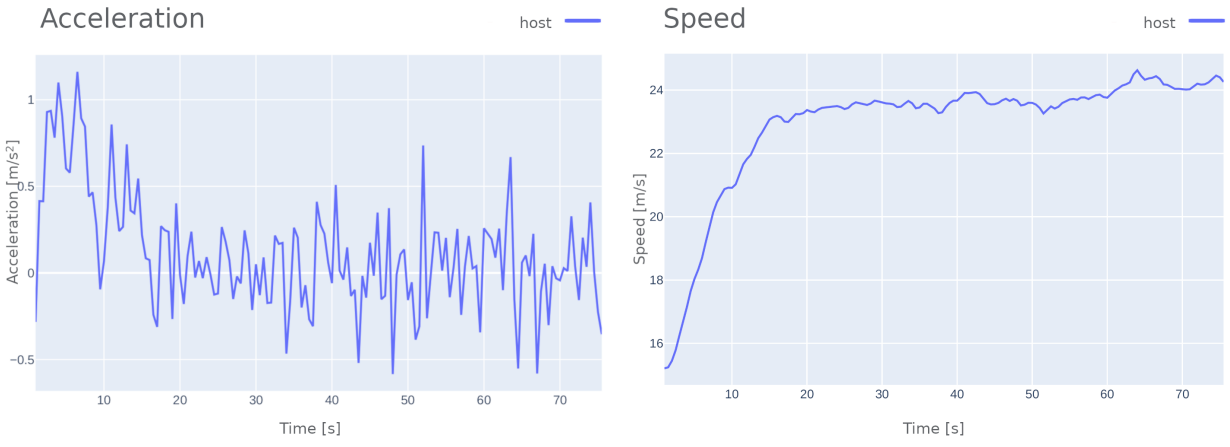
Testing policies on random scenarios are supplemented with testing on scripted scenarios that evaluate agents in typical road situations. Such an approach allows for a close inspection of agent behavior and comparing the policy outcome with expectations. Situations are diversified across a range of preset velocities, road geometries, and target behaviors. The following paragraphs explain the test case scenarios and expected agent behavior. The accompanying charts show the example trajectory of agents in the given scenarios.

Carry out preset velocity The test case assumed driving on empty lanes. The agent was intended to accelerate smoothly to reach the preset velocity and keep it. Figure 6.4 shows the acceleration and speed profile of the agent driving on a single lane and trying to achieve a preset velocity of 24m/s. It started rapidly to increase velocity from 15m/s up to 23m/s. Then for the rest of the episode, we can see that the agent's

KPI	**PPO on MARWIL**	**MARWIL on D_{opt}**	**PPO**
Speed execution squared (4.2)	8.296	6.862	11.206
Squared acceleration (4.3)	-0.527	-0.186	-1.347
Safety violation (4.4)	-1.092	-0.807	-0.453
Terminal State (4.5)	-2.600	-2.500	-0.400
Sum of rewards (6.1)	4.077	3.369	9.006
AverageSpeed [m/s]	22.797	20.936	24.376
AverageAcc [m/s^2]	-0.003	-0.033	-0.009
AverageAbsAcc [m/s^2]	0.328	0.218	0.518
AverageAbsJerk [m/s^3]	0.551	0.582	0.932
DistanceToFrontTarget [m]	129.680	125.312	118.773
AverageSpeedToTarget [%]	1.412	1.299	1.094
HeavyBrakingEvent [%]	0.012	0.000	0.017
SafetyViolation [%]	0.050	0.053	0.025
Collisions [%]	0.330	0.320	0.040
CauseCollisions [%]	0.260	0.250	0.030
NotCauseCollisions [%]	0.070	0.070	0.010
0scillation0nEmptyLane [m/s^2]	0.287	0.190	0.455

Table 6.3: Comparison of KPIs calculated for Marwil and PPO agent trained only in simulation and agent trained with simulated and real data. KPIs are generated based on 100 simulated episodes.

acceleration fluctuated around 0m/s to fulfill the target speed precisely. It learned that it needed to drive at a preset velocity. However, it received less reward for driving slightly above the speed limit than for driving slightly below it. It resulted directly from shape of speed execution reward term c_0 (4.2), (Fig. 4.4).

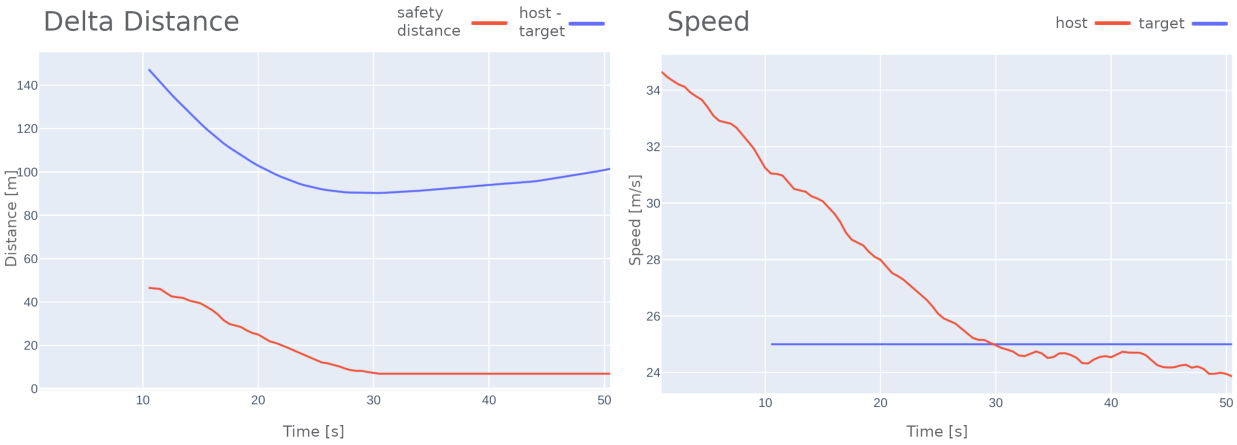


(a) Host acceleration

(b) Host velocity

Figure 6.4: Test case: carry out preset velocity.

Tracking vehicle with constant velocity The next test case type was to follow the target vehicle, which drove with constant velocity. It assumed that the agent should in the first place decrease the distance to the target. Then it should align velocity with the target and keep it until the end of the episode. Charts 6.5 show the offset between host and target and their velocity. When there is no data in the plot that regards the target, it means that it is not visible to the host since the gap is more expansive than 150m. Charts show that the host initially decreased the distance to the target as assumed. In the range of about 90m to the target, it started to adjust velocity. However, its acceleration oscillated, leading to slowing down and drifting apart the target. This is a unique scenario because the situation when the target is driving with constant velocity is very rare in the training dataset and in simulated scenarios. In the first source, there is always the sensor noise which affects the perceived target velocity. In simulation, the behavior model of simulated vehicles is prone to change velocity frequently.



(a) Offset between target and host.

(b) Velocity of leading and host vehicles

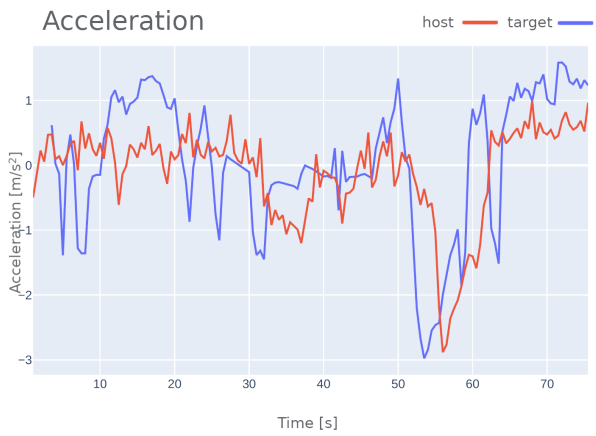
Figure 6.5: Test case: tracking vehicle with constant velocity.

Tracking vehicle with oscillating acceleration The typical scenario on the highway is when the leading target drives with varying velocities because of different reasons. In this scenario, the agent was supposed to satisfy the ACC objective while keeping the target within the sensor range. Figures 6.6 present the acceleration and speed of target and host vehicles. It is evident from the figures that the agent successfully complies with the objective by varying its acceleration less than the car in front throughout the entire episode.

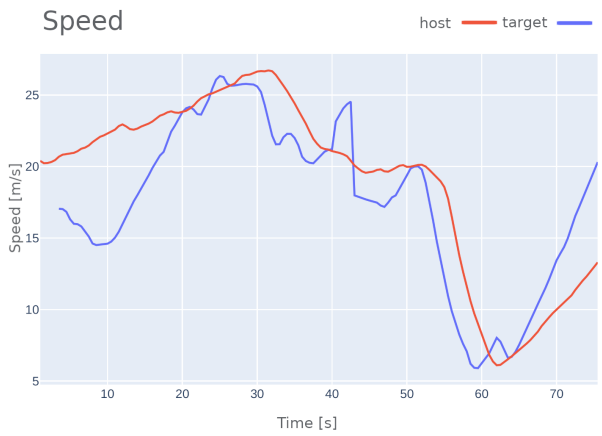
Static object on the road The scenario assumed driving on an initially empty highway. However, at a 250m distance from the host starting position, all lanes were occupied by stationary vehicles. The scenario tested whether the agent recognized the situation and started braking to avoid a collision. As can be seen in Figure 6.7, the host initially accelerated to reach preset velocity and, after perceiving static objects, started braking. It successfully avoids collision stopping 18m in front of obstacles.

6.5 Summary

The methodology presented in this study successfully produced the final policy (MARWIL on PPO) that generated the highest mean sum of rewards (6.1) among other agents during evaluation in resimulated sce-

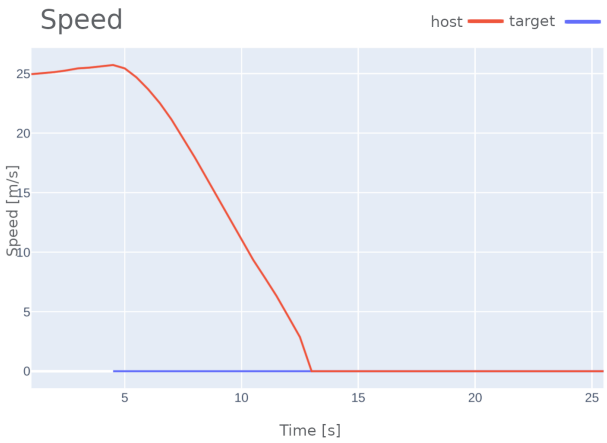


(a) Acceleration of leading and host vehicles.

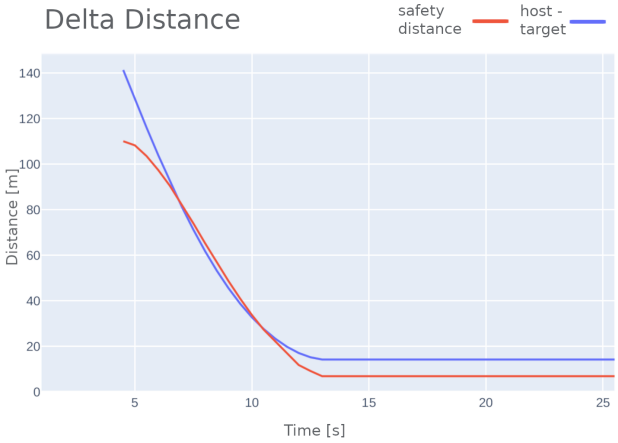


(b) Speed of leading and host vehicles.

Figure 6.6: Test case: tracking vehicle with oscillating acceleration



(a) Velocity of host and static objects.



(b) Delta distance between host and static objects

Figure 6.7: Test case: avoid collisions with static objects.

narios. The primary goal of the Adaptive Cruise Control system is to optimize both comfort and safety while simultaneously maximizing vehicle velocity. The final policy resulted in a lower average absolute acceleration and fewer safety violations compared to both a human driver and the baseline PPO driver. Inspection of the agent trajectories performed in resimulated real logs indicated that the driving behavior of the policy was quite similar to that of a human driver. However, it differed by keeping a higher degree of caution and prioritizing safety over speed maximization. Specifically, the policy maintained a greater distance from the leading vehicle than a human driver but had a similar rate of safety distance violations. The policy demonstrated the ability to adjust velocity to leading targets which velocity frequently changed. It learned to maintain a smaller acceleration variance than the leading vehicle. However, further training is required to eliminate action flickering, which results in unnecessary high jerks in consecutive steps, and other unwanted behaviors. During the evaluation, it was noticed that RL agents struggled with learning simple tasks such as maintaining a constant speed on an empty lane. In contrast, they easily cope with more advanced challenges

such as adjusting their speed to follow a target with varying velocities. This discrepancy could result from an overemphasis on randomized and complex scenarios during training, neglecting the essential basics. To enhance the training process, it is crucial to improve scenario distribution to include more straightforward scenarios.

Through the experimentation process, some constraints were discovered with respect to the suggested solution. In particular, it was observed that trajectory optimization (Sec. 5.2) could sometimes yield trajectories that differ significantly from the customary driving behavior exhibited by humans. Such divergence could have potentially detrimental effects on other drivers, leading to misunderstandings and disrupted collaboration between traffic participants due to the lack of accurate forecasting of the vehicle's trajectory. Nonetheless, it was noted that the trained policy performs optimally when it benefits from optimized trajectory paths. The optimization can then be adapted and utilized across the entire spectrum of executed trajectories during the training phases. This capacity allows for the unification of training samples, thus producing a near-optimal policy that closely approximates human behavior.

Another limitation regards the applicability of numerical optimization. The proposed solution may be only used when at least a basic environment transition model is available. Here it could be adopted since the model of the host movement is known (Sec. 2.1). However, the solution is limited because it is impossible to model the response of adjacent vehicles to optimized actions accurately. If we want to include modeling such responses we could use the behavior model included in the simulation. However, due to its stochasticity, it might be assumed several hypotheses of target responses. Therefore, from single action could result in the distribution of future states. To confine algorithm complexity, the presented solution is limited to preserving the original objects' trajectories. In case the environment model is known, the approach may be transformed to those represented in Model-Based Reinforcement Learning (for example [112, 111, 110]).

Moreover, the proposed solutions are expected to offer several benefits. One of the most significant advantages is that the approach eliminates the need to develop additional algorithms, which are commonly required in traditional ACC approaches. Specifically, our modular Neural Network natively incorporates the functionality of various algorithms such as trajectory prediction of adjacent cars, target selection, object lane assignment, and lane change prediction, among others. However, if there is a requirement for any of these predictions to be returned, the proposed architecture of the Neural Network is easily extendable with additional prediction heads. These heads can be trained in a supervised manner to return the desired outputs, providing flexibility and adaptability to our approach.

Chapter 7

Agent Explainability

As an addition to the evaluation process, I propose a new approach that can provide more transparency on the reasoning behind agent's decisions [41]. The method utilizes the trajectory samples (observation-action tuples) collected during the first evaluation phase. The method determines the extent to which a given input feature affects the agent's action and indicates the relationship between decisions, the importance of input features, and their values. The presented method allows for understanding the agent's behavior and determines whether the decision-making process is cohesive with human intuition or contradicts it.

7.1 Preliminaries

The decision module, which is based on an RL agent, is considered to be a black box because the agent's decision results from ANN inference which information flow is difficult to trace and comprehend. To understand why the agent selects a particular action, one needs to examine the flow of the input state through the ANN and induced neuron activation. The family of methods that incorporates that is called primary attribution. It allows for the examination of how a particular element of the input, layer, or individual neuron influences action selection. The name attribution refers directly to the magnitude of influencing.

To introduce the explainability into behavior planning agents, I incorporated the member of the primary attribution family, which is the method of Ingredient Gradients (IG) [120]. This method was applied to determine the significance of the neural network's input on its output. For calculation, IG requires baseline input o^{base} which is composed arbitrarily and should be neutral for the model. For example, if the model consumes images, the typical baseline is an image that contains all black or white pixels.

The process of the calculating attribution of input feature $g_i(o)$ starts by generating, in small steps α, a set of inputs by linear interpolation between the baseline o^{base} and the processed input $o = o_t$. Then it computes gradients between interpolated inputs and model outputs in order to, finally, approximate the integral with the Riemann Trapezoid rule as

$$g_i(o) = (o_i - o_i^{\text{base}}) \times \int_{\alpha=0}^{1} \frac{\partial f^{\text{ANN}}(o^{\text{base}} + \alpha \times (o - o^{\text{base}}))}{\partial o_i} d\alpha \tag{7.1}$$

where $f^{\mathrm{ANN}}(o)$ is function that represent ANN, i is feature dimension, $o = o_t$ is input observation, o^{base} is a baseline observation, and $\frac{\partial f^{\mathrm{ANN}}(o)}{\partial o_i}$ is gradient of $f^{\mathrm{ANN}}(o)$ along the i^{th} dimension. α denotes interpolation constant step.

The application of this method directly allows only for interpretation of what influenced the decision which was selected in a given state. However, understanding why action was selected in a single state does not really reflect the pattern of agent behavior. Moreover, similar states may trigger quite distinct actions, therefore, it is beneficial to understand the behavior pattern over multiple episodes. Given the above, I propose incorporating statistical analyses of attributation calculated for all encountered states in numerous episodes.

7.2 Experiments

The aim of this study is to develop an interpretation method of RL agents' decisions adequate for discrete and continuous action space. To showcase the effectiveness of the proposed algorithm, two agents were trained: one using a discrete action space (Maneuver Agent) [8] and the other using a continuous action space (ACC Agent).

The Maneuver agent states a part of the behavior module of the motion model, and it is responsible for selecting the appropriate maneuver to be executed (Fig. 2.3). Its discrete action space comprises six maneuvers: Follow Lane (FL), Prepare for Lane Change: Right (PLCR), Left: (PLCL), Lane Change: Right (LCR), Left (LCL), and Abort Maneuver. The agent's primary objective is to navigate efficiently on the multilane highway while maintaining a gentle driving style. The expected behaviors of the agent involve changing to the faster lane when it is safe and the host's velocity is lower than the preset speed limit. Additionally expected is returning to the right lane when it is possible without affecting fulfilling preset velocity. Depending on the selected action, the trajectory generation module (2.10) plans the trajectory to achieve the intended position on the road. In the case of lane change actions, the trajectory module plans a path that will take the vehicle to the center of the adjacent lane within a set amount of time. If the agent decides to abort the maneuver mid-change, the module plans a path to return the vehicle to its original lane. In other cases, the planned trajectory follows the currently occupied lane. However, if the prepare for lane change maneuver is selected vehicle switches on an adequate blinker to communicate its intentions. Such maneuvers may influence other traffic participants which for example may make room for the host's lane change maneuver.

In contrast, the ACC Agent is responsible for planning the continuous acceleration value when the higher-level agent selects the Follow Lane maneuver according to the objective specified in Section 2.1.

The training of both agents utilized the same simulation (Sec. 2.4) and highway environment as described in Section 4.1. However, the observation space and ANN architecture differ slightly from the previous experiments. The actual ANN architecture used for this research case is presented in Figure 7.1.

The ANN consumes observation from observation space $o_t \in \Omega^{\mathrm{acc}}$ for ACC agent and $o_t \in \Omega^{\mathrm{man}}$ in case of Maneuver agent. Both observation spaces include observation of host, adjacent vehicles, and road $\Omega = \Omega_{\mathrm{host}} \times \Omega_{\mathrm{objects}} \times \Omega_{\mathrm{road}}$. The host's observation space includes longitudinal velocity (v_s), the degree to which it adheres to the speed limit ($v_{\mathrm{s\_limit\_exec}} = v_s / v_{s,\mathrm{max}}$), longitudinal acceleration

(a_s) and selected action in previous time step (ACC agent $a_{s,target,t-1}$; Maneuver agent u_{t-1}). For the Maneuver agent, additional information is included, such as the lateral position (d), velocity (v_d) and acceleration (a_d), rotation towards the center of the lane (ζ_{fcs}) and information whether it is safe to change lanes accordingly to RSS rules ($u_{\text{LCL avail}}, u_{\text{LCR avail}}$). $\Omega^{\text{acc}}_{\text{host}} = \{v_s, v_{\text{s\_limit\_exec}}, a_s, a_{s,target,t-1}\}$ $\Omega^{\text{man}}_{\text{host}} = \{d, v_s, v_d, a_s, a_d, v_{\text{s\_limit\_exec}}, \zeta_{\text{fcs}}, u_{t-1}, u_{\text{LCL avail}}, u_{\text{LCR avail}}\}$

The second observation space Ω_{objects} presents the perceived vehicles and consists of relative to the host information about longitudinal and lateral distance (s, d), velocity (v_s, v_d), and acceleration (a_s, a_d)). It also consists data about target's rotation (ζ_{vcs}) and its dimensions (w, l). $\Omega_{\text{objects}} = \{s_{\text{obj}}, d_{\text{obj}}, v_{s,\text{obj}}, v_{d,\text{obj}}, a_{s,\text{obj}}, a_{d,\text{obj}}, \zeta_{\text{vcs,obj}}, w_{\text{obj}}, l_{\text{obj}}\}$

The Ω_{road} represents the road geometry by encoding up to six visible marker lines with information about delimiter type (c_{type}), curvature ($\kappa = \frac{1}{r}$), rotation in the host's position (ζ_{wcs}), sensing range (Δ_r), and lateral distance from the host to the marking line (d_{road}). $\Omega_{\text{road}} = \{c_{\text{type}}, \kappa, \zeta_{wcs}, \Delta_r, d_{\text{road}}\}$

The input data is encoded by the feed-forward layers in the perception module (Fig. 7.1). The encoded in the latent state features are then passed through three residual blocks [121], the outputs of which are sent to the Control Module. For the ACC agent, the ANN generates parameters for the Gaussian distribution, precisely the mean value μ and the natural logarithm of standard deviation $\ln(\sigma)$. The mean value is obtained through the application of the hyperbolic tangent (tanh) activation function, while the natural logarithm of standard deviation is a trainable input-independent parameter. For the Maneuver agent, the output from the residual blocks is passed through the fully connected layer. Any maneuvers that are deemed unsafe according to the rules described in [57] and implemented in [8] are masked out. The masked-out output is then converted to probability distribution by the softmax function.

The reward functions vary for both agents since their objectives are different. The ACC agent reward R_{acc} is a weighted sum of the following terms:

- **Speed execution** $c_0^{\text{acc}}(v^{\text{vcs}}, v_{\max})$ - calculated as a ratio of host absolute velocity and speed limit - forces agent to maximize speed limit (4.2).
- **Squared acceleration** $c_1^{\text{acc}}(a^{\text{vcs}}) = (a^{\text{vcs}})^2$ - the squared value of acceleration - negative reward promotes a smooth ride
- **Jerk absolute** $c_2^{\text{acc}}(a^{\text{vcs}}) = |\frac{\partial a^{\text{vcs}}(t)}{\partial t}|$ - the value of absolute jerk - also promotes comfort driving experience
- **Safety violation** $c_3^{\text{acc}}(s^{\text{v}}_{\text{host}}, s^{\text{v}}_{\text{target}})$ - a negative reward for being too close to other vehicles (4.4). Distance is calculated based on Responsibility Sensitive Safety assumptions [57]
- **Terminal State** $c_4^{\text{acc}}(\Delta s, v^{\text{vcs}})$ - a reward for causing a collision or speeding too much (4.5).

$$\begin{aligned} R_{\text{acc}}(s^{\text{v}}_{\text{host}}, s^{\text{v}}_{\text{target}}) = 0.07c_0^{\text{acc}}(v^{\text{vcs}}, v_{\max})^2 - 0.02c_1^{\text{acc}}(a^{\text{vcs}}) - 0.0001c_2^{\text{acc}}(a^{\text{vcs}}) + \\ - 0.2c_3^{\text{acc}}(s^{\text{v}}_{\text{host}}, s^{\text{v}}_{\text{target}}) - 10c_4^{\text{acc}}(\Delta s, v^{\text{vcs}}) \end{aligned} \tag{7.2}$$

The reward function for Maneuver agent R_{man} consists of following terms:

- **Speed execution** $c_0^{\text{man}}(v^{\text{vcs}}, v_{\max})$ calculated as a ratio of host velocity and speed limit - forces agent to maximize speed limit, encourages to overtaking slower cars (defined accordingly to (4.2))

- **Squared acceleration** $c_1^{\text{man}}(a^{\text{vcs}}) = (a^{\text{vcs}})$ - a reward for braking events, incentives for smooth driving

- **Sequence Maneuver Execution** $c_2^{\text{man}}(u_t, u_{t-1})$ - a reward for inconsistency in selecting maneuvers, this term has a non-zero value when the agent selects a different action than selected before. It reduces the action flickering problem

$$c_2^{\text{man}}(u_t, u_{t-1}) = \begin{cases} 1, & \text{if } u_{t-1} \neq u_t \\ 0, & \text{otherwise,} \end{cases} \tag{7.3}$$

- **Collision** $c_3^{\text{man}}(\Delta s)$ - a negative reward for causing collision (4.5)

- **Right Lane available** $c_4(u_{\text{LCR avail}})$ - non-zero when the right lane is available and the agent can change it. It promotes gentleness on the road by releasing the left lane for faster vehicles.

$$c_4^{\text{man}} = u_{\text{LCR avail}} \tag{7.4}$$

- **Being overtaken by right** $c_5^{\text{man}}(u_{\text{LCL avail}}, v_{s,t}, v_{s,\text{obj right},t})$ - This reward is non-zero when the agent is slower than vehicles in the right lane. The agent should change the lane to the right to allow faster vehicles to drive on the left.

$$c_5^{\text{man}}(u_{\text{LCL avail}}, v_s, v_{s,\text{obj right},t}) = \begin{cases} 1, & \text{if} \neg u_{\text{LCL avail}} \wedge v_s < v_{s,\text{obj right},t} \\ 0, & \text{otherwise,} \end{cases} \tag{7.5}$$

where $v_{s,\text{obj right},t}$ is the longitudinal velocity of the vehicle on the right adjacent lane.

- **Over taking right** $c_6^{\text{man}}(v_s, v_{s,\text{objright},t})$ - This term rewards agents for overtaking other cars while driving on the left lane.

$$c_6^{\text{man}}(v_s, v_{s,\text{obj right},t}) = \begin{cases} 1, & \text{if } v_s > v_{s,\text{obj right},t} \\ 0, & \text{otherwise,} \end{cases} \tag{7.6}$$

where $v_{s,\text{obj right},t}$ is the longitudinal velocity of the vehicle on the right adjacent lane.

$$\begin{aligned} R_{\text{man}}(s_t) =& 0.03c_0^{\text{man}}(v^{\text{vcs}}, v_{\max})^3 - 0.0005c_1^{\text{man}}(a^{\text{vcs}}) - 0.0001c_2^{\text{man}}(u_t, u_{t-1}) - c_3^{\text{man}}(\Delta s) + \\ & - 0.03c_4^{\text{man}}(u_{\text{LCR avail}}) - 0.01c_5^{\text{man}}(u_{\text{LCL avail}}, v_{s,\text{objright},t}) + 0.07c_6^{\text{man}}(v_s, v_{s,\text{obj right},t}) \end{aligned} \tag{7.7}$$

training parameter	Maneuver	ACC
gamma γ	0.9985	0.998
lambda λ	0.95	0.95
batch mode	complete episodes	complete episodes
train batch size	1024	50000
SGD minibatch size	512	20000
shuffle sequences	True	True
number of SGD iterations	3	3
grad clip	3.0	3.0
KL coefficient	0.0	0.0
clip parameter	0.2	0.2
entropy coefficient	0	0
learning rate	4.5e-5	1e-4

Table 7.1: Hyperparameters of PPO algorithm used for training ACC and Maneuver Agents. The training sessions were performed with the usage of RLlib [107].

7.3 Training

Both the ACC and Maneuver agents were trained for over 30 million steps. During the training process, the agents started with random actions as a result of the neural networks being initialized randomly and gradually improved their performance over time. The agents with the highest mean sum of rewards (6.1) were selected for further investigation. Additionally, to ensure that the agents behaved as expected, they were tested with predefined test scenarios as described in Section 6.2. In addition to the predefined scenarios described above, I have also created typical scenarios for testing the Maneuver agent. The performance of agents during training may be observed through observation of how agents handle predefined scenarios.

For example, a test scenario for the ACC agent involved a single-lane road with both a host vehicle and a front target that accelerated and decelerated interchangeably. The goal of the agent was to follow the target while minimizing acceleration oscillation. Initially, the agent struggled to track the target and often either lost track of it or accelerated too quickly and crashed into it. However, over the course of training, the agent learned how to track the target and maintained a smooth driving experience by minimizing acceleration.

Another example regards the Maneuver agent. It was tested on a scenario involving a host vehicle and a slower target car driving in the rightmost lane of a three-lane road. The agent's task was to overtake the target vehicle and return to the rightmost lane. At the beginning of training, the agent followed the target vehicle and changed lanes to the leftmost lane because it learned this was the fastest lane. However, towards the end of the training, the agent learned to change to the center lane and return to the right lane after overtaking the target vehicle. It could learn it due to incorporating three terms in the reward function $(c_4^{\text{man}}, c_5^{\text{man}}, c_6^{\text{man}})$, as specified in (7.7).

Other testing scenarios and how agents execute them are comparable to those outlined in Section 6.4.

7.4 Collecting Neural Activations

Agents with the highest mean sum of rewards were evaluated in randomly generated simulation scenarios. The process generated 5 hours of driving experience for the Maneuver agent and 3.5 hours for the ACC agent, which corresponds to over 240,000 simulation steps. To ensure temporal independence for statistical analysis, every tenth sample was selected from this set for further consideration. The sample included ANN inputs (state observation $o_{s,t}$) and agent decisions. The decisions are stored in the form of action values u_t for the ACC agent and probabilities of selecting a particular action for the Maneuver Agent. The attributions of each input value were calculated using the Integrated Gradients method (7.1). As a baseline input o^{base}, there was defined an observation of a state that represents a three-lane highway with no other vehicles besides the host in its default state (maximum legal velocity, zero acceleration). The Captum library [122], which provides an implementation of various interpretability and explainability methods for PyTorch models, was used for the calculation. The results of the attributions calculation, associated input features, and ANN's decisions were subjected to further inspection through statistical analysis.

7.5 Statistical Analysis

The statistical analysis consists of two parts performed using Minitab software [123]. The first part focuses on the examination of the level of significance of the attributation values and on the analysis of their distribution. The second one investigates the relationships between attributation values and input features.

The first part begins with the identification of elements with statistically significant values of attributation distribution. That identification regards action selected from the Maneuver Agent's actions space and overall action distribution of the ACC agent. The elements at which significant levels were determined are subject to analysis of variance (ANOVA). For this analysis, attributation data were divided into six groups according to the type of maneuver. Additionally, to simplify the analysis, attributations related to objects and roads were summed up in accordance with matching components (for example, components related to road lanes' marking types or objects' velocities). For each resulting parameter, a t-test was performed with the null hypothesis $H_0 : \mu = 0.03$ and alternative hypothesis $H_1 : \mu > 0.03$. The significance level of all tests was set to 0.05. Based on the t-test results, parameter distribution was categorized as significant when the mean value is higher than μ. In the case of the Maneuver Agent, it was conducted by distinguishing between different maneuvers. In the end, Welch's ANOVA [124] was performed for results that are significantly based on the t-test. Such analysis informs which parameters are significantly more important than others regarding available maneuvers. Additionally, all samples were divided into groups using Games Howell [125] (Tab. 7.4). To visualize distinguished results, the standard deviation was calculated for these samples and 95%-confidence intervals for their means which gives 95% assurance that the expected value is within these intervals regarding the dispersion of data.

The second part of the analysis relies on the examination of the linear and monotonic relationship between feature attributation and the probability of selecting a given maneuver. A Pearson correlation [126] was applied to study linear correlation and Spearman's rank correlation coefficient Rho [127] to examine a monotonic correlation. Correlations were calculated for the attributation of all input features concerning

the probability of selecting a particular maneuver. Additionally, for the ACC agent, mutual information was calculated for action and features-attributation (Tab. 7.5).

An analysis based on a Pearson correlation begins with the calculation of the p-value and identification of whether the correlation is significant at 0.05 α-level. The p-value indicates if the correlation coefficient is significantly different from 0. If the coefficient tends to 0, it means that there absence of a linear relationship in the sample population. Afterward, the Pearson correlation coefficients were calculated with

$$\rho_{x,y} = \frac{\mathrm{cov}(x,y)}{\sigma_x \sigma_y} \tag{7.8}$$

where x is given input feature and y an attribution of that featuer. cov is the covariance and σ_x, σ_y are the standard deviation of x and y.

The coefficients were further analyzed in order to determine the strength and correlation direction (Tab. 7.2). The coefficients were within the range of -1 to $+1$. The larger the absolute value of the coefficient, the stronger the linear relationship between the samples was. It was assumed that there is a weak correlation if the absolute value of the coefficient is lower than 0.4. If this value is within the range of 0.4-0.8, the correlation is moderate, and large if higher than 0.8. To investigate the direction of the dependence, the sign of the coefficient need to be examined. The positive sign means that compared variables tend in the same direction and the line that represents the correlation slopes upward. The coefficient values below 0 mean that the values of compared variables tend in different directions, and the correlation line tilts downward.

Due to the fact that Pearson correlation indicates only linear correlation, Spearman's rank correlation coefficient Rho was utilized to detect monotonic correlations, which are not linear (Tab. 7.3). The calculation of Spearman Rho correlation relies on calculating the rank of the raw data and applying the correlation formula:

$$r_s = \rho_{r(x),r(y)} = \frac{cov(r(x), r(y))}{\sigma_{r(x)} \sigma_{r(y)}} \tag{7.9}$$

where $\rho_{r(x),r(y)}$ denotes the usual Pearson correlation coefficient, but applied to the rank variables $r(x), r(y)$, $cov(r(x), r(y))$ are the covariance of the rank variables, $\sigma_{r(x)}\sigma_{r(y)}$ are the standard deviations of the rank variables. Only if all n ranks are distinct integers, it can be computed using a formula $r_s = 1 - \frac{6 \sum d_i^2}{n(n^2-1)}$) where: $d_i = r(x_i) - r(y_i)$ is the difference between the two ranks of each observation, n is the number of observations.

Spearman Rho correlation coefficient describes the direction and strength of the monotonic relationship, and its coefficients are interpreted analogously to Pearson's results.

7.6 Results

In this study, the statistical analysis results were analyzed with a systematic approach. Firstly, I examined the boxplots (Figure 7.2, Appendix A.1) that visually represent the distribution of attributation for each input signal in a specific maneuver. These plots determine the extent to which a given feature contributes to selecting a particular maneuver. For instance, Figure 7.2 depicts the distribution of attributation for the feature $d \in \Omega_{\mathrm{host}}^{\mathrm{man}}$, which represents the lateral distance of the host-vehicle from the center of the driving lane.

The findings indicated that the middle 50% of the distributions (box), mean (dot), and median (horizontal line in the box) of attributation values are significantly higher for maneuvers related to Lane Change. Moreover, attributation values greater than 0 for Follow Lane and Abort maneuvers are deemed outliers (star), consistent with drivers' intuition. These observations affirm that the ANN performs as intended, at least in this specific domain.

Other results, presented in Figure A.6, demonstrate that the agent emphasizes the information about fulfilling preset velocity while selecting the maneuver related to the changing lane to the left. It indicates that the agent reasonably connected the signal of preset velocity with the actions that allow for maximizing it and fulfilling the controller objective. Because usually changing the lane to the left allows for overtaking slower vehicles and driving faster.

Other results which are cohesive but rather contradict intuition are presented in Figures A.6 and A.7. Boxplots demonstrate that the agent pays more attention to information that a given maneuver is safe while selecting the opposite action. Figure A.6 shows that agents focus more on the availability of lane change left maneuver while deciding to change the lane to the right. FigureA.6 shows the opposite result.

After inspecting boxplots, I explored the correlation between attributions and the input feature values, examining this relationship from two perspectives. Firstly, I investigated strong correlations and compared them against human intuition. The analysis revealed, for example, that when considering the selection of the Follow Lane maneuver, the agent places less emphasis on the longitudinal velocity value v_s as the velocity increases. Instead, it focuses more on the parameter that indicates whether the velocity limit $v_{s\_limit\_exec}$ has been met. This tendency was evident from the Spearman-Rho correlation analysis, while the Pearson correlation did not exhibit this relationship. Furthermore, these findings could be confirmed by inspecting the scatterplots of v_s and $v_{s\_limit\_exec}$ attributions, as depicted in Figure 7.4.

I concluded that the agent's behavior is similar to that of human drivers, as individuals tend to focus less on absolute speed and more on driving at a legal velocity when accelerating. When speeding up, human drivers first concentrate on gaining speed and then they shift their attention to assessing whether they comply with the speed limit.

Regarding the ACC agent, I observed a moderate correlation between the attribution of acceleration and the acceleration action value. The analysis indicated that the agent emphasizes acceleration value more as it increases, which is a desirable correlation. However, I believe that the values of attribution should be higher. In Figure 7.5, it could be observed at least three distinct correlation patterns that are linked to factors not examined in this study. Therefore, conducting further research to investigate these factors is recommended.

Furthermore, the findings suggest that the agent prioritizes other vehicles' positions when braking, as illustrated in Figure 7.5. This observation underscores the importance of the agent's perception of its environment in driving safety and efficiency.

The research process continued, with the goal of identifying the optimal location for a strong correlation with human cognitive processes. For instance, the hypothesis was that drivers compare the longitudinal distance to the target vehicle with their velocity, resulting in a strong correlation between the attribution of objects' position relative to longitudinal velocity. However, the analysis indicated only a weak correlation contrary to the assumptions.

Host	**Follow Lane**	**PLCL**	**PLCR**	**LCL**	**LCR**	**Abort**
d	0.008	-0.017	-0.036	-0.862	-0.707	-0.001
v_s	0.024	-0.221	0.080	0.040	0.255	0.162
$v_{\text{s\_limit\_exec}}$	-0.125	0.334	0.031	-0.071	0.096	0.113
v_d	0.002	0.018	-0.003	-0.915	-0.677	-0.006
a_s	-0.013	0.053	-0.035	-	-0.100	0.029
a_d	-0.008	-0.020	0.024	0.309	-0.104	-0.025
ζ_{fcs}	0.002	0.010	0.000	-0.893	-0.668	-0.003

Table 7.2: Table with values of Pearson corellation for attributation of $v_{\text{s\_limit\_exec}}$ with respect to values of hosts's features.

Host	**Follow Lane**	**PLCL**	**PLCR**	**LCL**	**LCR**	**Abort**
d	-0.019	0.012	-0.019	-0.681	-0.823	-0.116
v_s	0.065	-0.217	0.038	0.114	0.076	0.182
$v_{\text{s\_limit\_exec}}$	-0.003	0.497	0.036	-0.107	0.188	0.135
v_d	0.014	0.015	0.013	-0.706	-0.775	-0.081
a_s	-0.022	-0.107	-0.014	-0.025	-0.037	0.171
a_d	0.005	0.029	0.017	0.191	-0.341	-0.042
ζ_{fcs}	0.012	0.013	0.016	-0.696	-0.753	-0.084

Table 7.3: Values of Spearman Rho correlation for attributation of $v_{\text{s\_limit\_exec}}$ with respect to values of features which describe host state.

In addition to explaining agent behavior, such a broad analysis of feature values and their attributations allows for inspecting any potential error in ANN or implementation. For example, reviewing the scatterplots (e.g., Figure 7.4), which illustrate the value of attribution concerning the input feature values, I noticed that one feature $d \in \Omega_{\text{host}}^{\text{man}}$ was normalized to the range (-2,0) instead of (-1,1). This observation allowed for fixing the function that normalizes the agent's observations and for conducting experiments on the correct implementation.

The second discovery regards the ANN architecture. It was noticed that the absence of attribution for every sample in a specific input feature region might indicate the issue of vanishing gradients in the model. This type of problem is heavily detected but necessary for accurate model training. In the current case, the incorrect implementation of tensors concatenation in the perception module precludes the model from passing gradients through it, thereby preventing the agent from utilizing a portion of the input knowledge. This finding enabled fixing the implementation and conducting more successful training.

action	**number of samples**	**mean**	**grouping**				
LCL	202	0.2515	A				
PLCL	7226	0.07468		B			
LCR	162	0.02159			C		
Abort	911	0.00946			C		
Follow Lane	9519	0.002283				D	
PLCR	4070	0.000718					E

Table 7.4: Grouping information using the Games-Howell Method and 95% Confidence for $v_{\mathrm{s\_limit\_exec}}$. Means that do not share a letter are significantly different.

Feature	**MI**
$a_{s,\mathrm{ego}}$	1.658
$a_{s,target,t-1}$	1.074
$g(a_{s,\mathrm{obj}})$	0.621
$g(a_{s,target,t-1})$	0.350
$g(v_{s,\mathrm{ego}})$	0.273
$g(d_{\mathrm{road}})$	0.272
$g(v_{s\_limit\_exec})$	0.248
$g(s_{\mathrm{range}})$	0.244
$g(s_{\mathrm{obj}})$	0.196
$g(d_{\mathrm{obj}})$	0.159

Table 7.5: Values of highest Mutual information for the action of ACC agent with respect to values of the host's features and attributation.

7.7 Application

The presented method can provide a valuable contribution to the understanding of RL agents that make decisions based on the distribution generated by ANNs. Specifically, it enables the identification of the input features that have the most significant influence on the agent's decisions and the examination of the correlation between the importance of a given input feature and its value. Additionally, it facilitates the detection of errors present in the model or in the input data. For instance, it may reveal the vanishing gradient problem, where critical information is ignored, or incorrect data normalization that causes a wrong data distribution in the charts. These issues were addressed by improving the model architecture and the normalization implementation, respectively. The application of the presented method can increase the safety and predictability of the entire system, particularly in the case of automated vehicle motion planning, which may enhance the reliability of machine learning applications for OEMs and consumers. Furthermore, the method's results may be utilized to improve the ANN architecture or to enhance the training process. It may be accomplished by tuning the reward function to represent the controller objectives better or redesigning the ANN modules that neglected the input features. For example, if the analysis reveals that the agent does not pay enough attention to other objects, we may propose to incorporate a term based on the time-to-collision metric in the reward function. To address this issue, it could be used a proven architecture such as the one presented in [91].

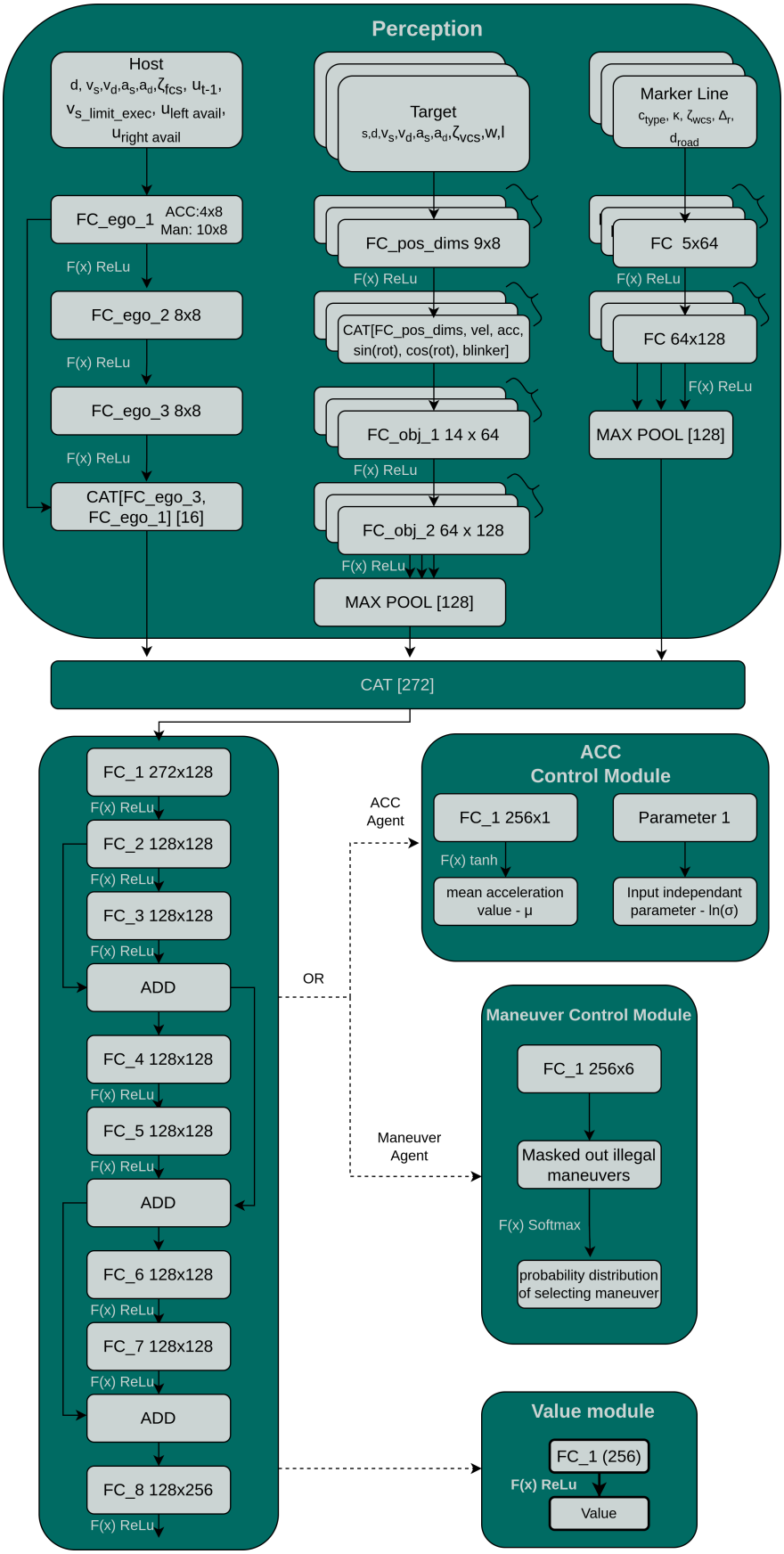


Figure 7.1: ANN architecture: FC_id - fully connected layers with bias; CAT - operation of features concatenation, { means that layers have shared weights (ex. each target is processed by the same layer).

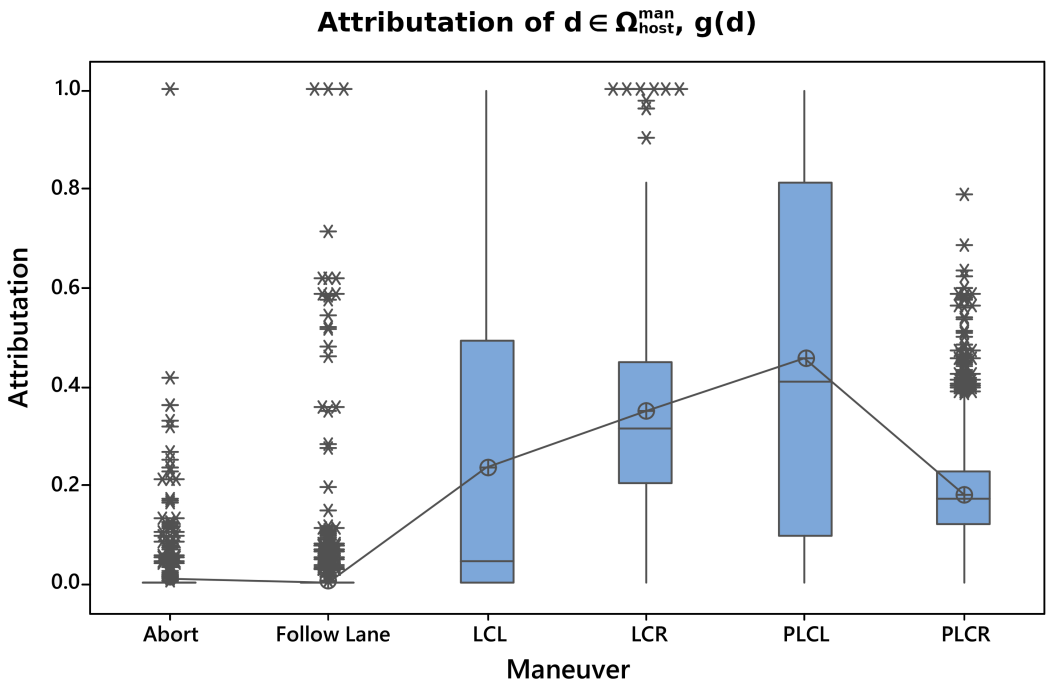


Figure 7.2: Distributions of attributation values for one of host parameters ($d \in \Omega_{\text{host}}^{\text{man}}$) for all maneuver types.

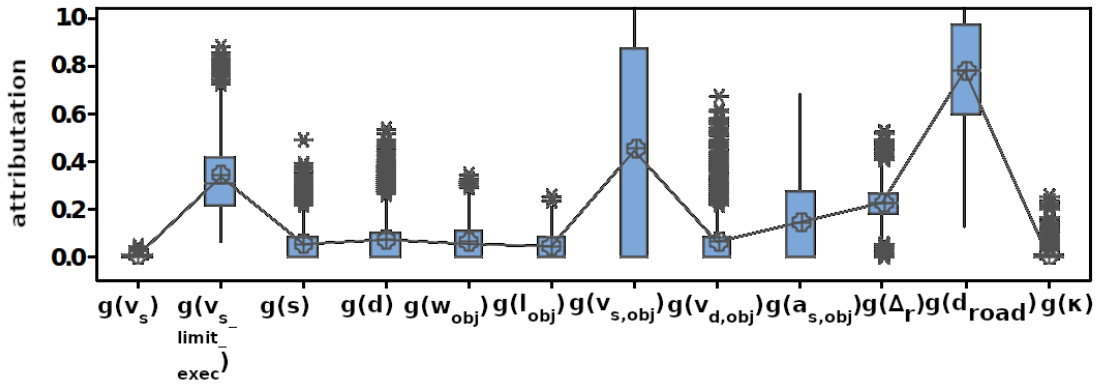


Figure 7.3: Distributions of attributation values for all ANN inputs - attributations of the same group of parameters for multiple objects are summed.

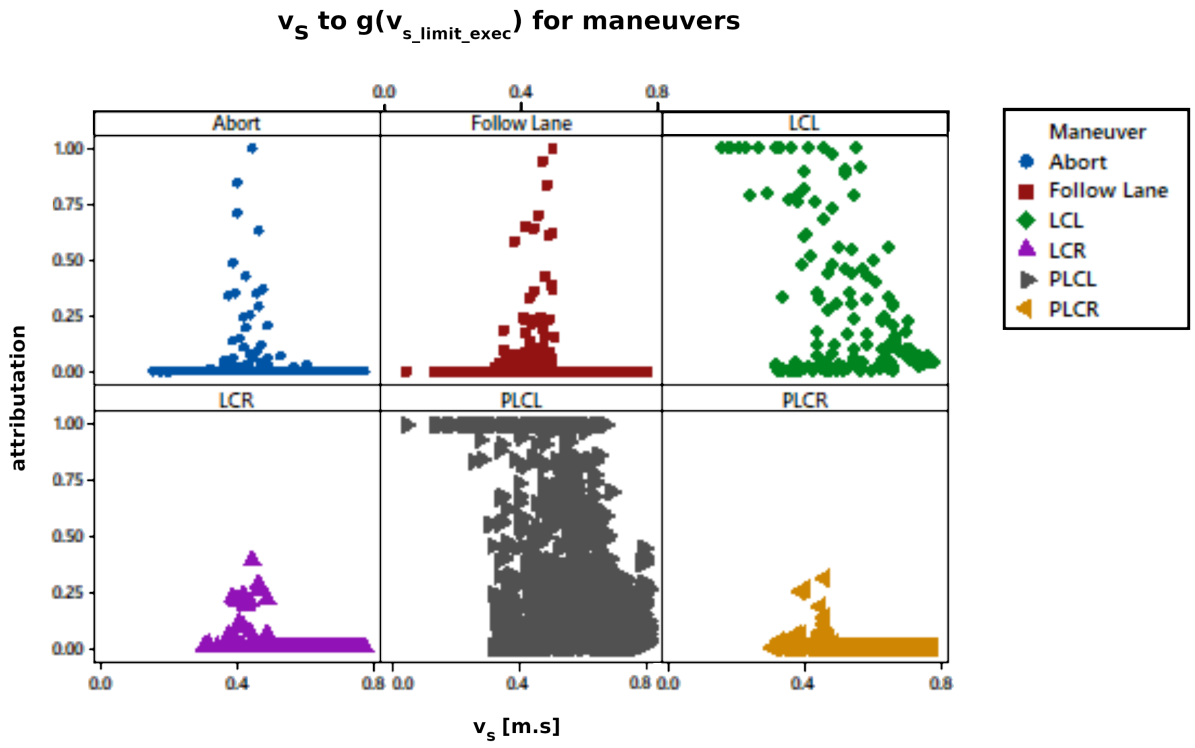


Figure 7.4: Scatterplot shows comparison between host's longitudinal velocity v_s and attribution values of $v_{s\_limit\_exec}$ for all maneuvers

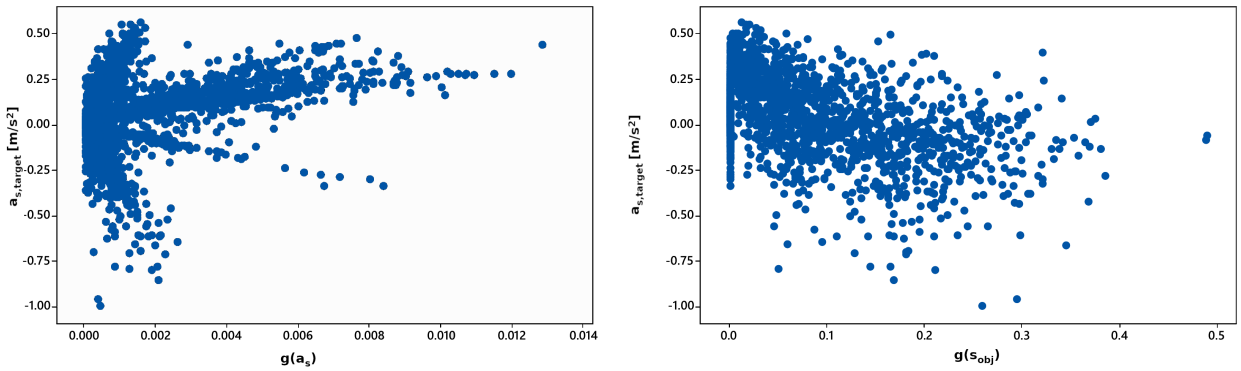


Figure 7.5: Scatterplots presents the correlation between acceleration $a_s \in \Omega^{\text{acc}}_{\text{host}}$ of continuous agent and attribution of its acceleration value and position of other objects $s_{\text{obj}} \in \Omega^{\text{acc}}_{\text{obj}}$.

Chapter 8

Conclusions

8.1 Summary

The thesis presents an approach for developing a behavior policy for an Adaptive Cruise Control driving mode that is efficient in a natural environment. The policy is integrated into the motion planning architecture of automated vehicles. The modularity of architecture supports the transparency requirements of vehicles' OEMs. The behavior module receives data from the perception module about traffic situations and user preferences. Based on that it returns the directives to subsequent modules. The behavior policy predicts the target acceleration which should be achieved by vehicle in 0.5s. The acceleration value is utilized by the trajectory module that generates a continuous reference trajectory that is executed by the control module.

The presented approach benefits from the capabilities of Offline and Online Reinforcement Learning algorithms and combines them to alleviate the accompanying disadvantages. The Online RL methods rely on interaction with a simulated environment to collect experiences and evaluations of performed actions in the form of rewards. They utilized that knowledge to optimize the behavior policy which is supposed to select actions that maximize the the sum of collected rewards. Utilization of simulation results in a policy that handles a broad distribution of situations. However, this approach may not be feasible in real conditions due to the fact that policy is optimized towards the simulated domain which may deviate from real environment. On the contrary, the Offline RL method uses only data collected in natural conditions, usually by a human expert. The optimization on such data leads to a policy that can handle real-world situations. However, this approach covers a narrower distribution of cases than simulation can produce, providing only limited experience to the policy.

The proposed combination of methods includes training ANN-based policy on a dataset using Offline RL algorithms and then fine-tuning policy in an online simulation with real data support. The dataset (Sec. 4.2) is created from human driving recordings according to the required format by the algorithm (SARS' trajectories). Being aware of the suboptimality of human actions, the collected dataset is optimized with numeric optimization (Sec. 5.2). The original and optimized datasets were used for initial agent training with the MARWIL and BC algorithms. The resulting agents were evaluated in resimulated driving logs where their performance was tested under natural circumstances. Evaluation were summed up in the form

of proposed KPIs which are shown in Table 6.1 in Section 6.2. The best performant policy, which was the MARWIL policy trained on optimized data, was then fine-tuned in a simulated environment (Sec. 4.1).

The subsequents training was necessary to train policy with the situations not included in training data. Due to the fact that such fine-tuning can lead to catastrophic forgetting of knowledge inferred during offline training, the resimulation of driving logs was incorporated as a part of the learning curriculum. The resimulation relied on replaying the scenarios included in dataset but agent might control its own trajectory by selecting actions. The trajectories of detected by perception module objects remained original. In that setup the policy could still optimize its behavior in a vast range of simulated scenarios as well as real situations.

In order to compare the effectiveness of the proposed solution, a baseline Reinforcement Learning training of ACC policy was conducted in simulation, as outlined in Section 5.1. The results from training policies were compared and evaluated using the approach described in Section 6.1, as presented in Section 6.2.

The agents' evaluation based on real data revealed that the agent trained accordingly to the proposed method surpassed the human driver to a certain extent. Its actions distinguished itself by minimizing absolute acceleration and maintained a wider separation from the leading vehicle. However, it was observed that the human driver moved faster and closer to targets than the trained agents. Further close inspection of several resimulated episodes showed that the behavior pattern of our policy closely resembled that of human driving but with a slight delay and slower reaction times. These findings may suggest that our proposed solution is a promising approach to developing behavior policy. However further extensive trainings, with a carefully selected distribution of scenarios, may be required to exceed the performance of human drivers in all scenarios.

As a supplement to the evaluation process, the work presents a novel approach for the explainability of the RL agent whose actions are derived from ANN inference (Sec. 7). The method requires collecting a massive set of observation-action pairs from agent interactions in the testing environment. Based on that it calculates the significance rate of input features for all pairs and conducts a statistical analysis to identify the meaningful correlation between actions and the observation's component. Such analysis enables interpreting the ANN-based RL agent and investigating whether the agent behavior pattern matches human intuition. The statistical techniques utilized in this approach enable the examination of overall behavior across multiple episodes instead of a single action in a specific situation. Moreover, the method can help detect errors in the ANN model, such as the vanishing gradient problem or concealed implementation errors of observation encoding or processing.

8.2 Prove Thesis

The demonstrated evaluation results (Sec. 6.2) indicated that the presented approach satisfies the expectations expressed in the presented thesis (Sec. 1.3) and control objectives (Sec. 5.2). First of all, it was stated that the presented methodology of combining real data with simulated ones in the training process increases the performance of policy in the natural environment compared to baseline training in simulation. The thesis was supported by comprehensive testing which results are presented in Section 6.2 and particularly in Tables 6.2 and 6.3. Calculated metrics and conducted tests showed that it is possible to achieve the human-like performance of trained agents with the presented approach. The final policy resembles the behavior pattern of a

human driver. It slightly outperforms comfort and safety compared to human driving. However, it developed a much safer policy that tends to prioritize safety over speed maximization. It is evident from Table 6.2 that baseline policy trained only in the simulation performed worse in resimulated real scenarios.

Additionally, the comparison of KPIs in Table 6.3 showed the superiority of the final policy over the MARWIL policy trained purely on static data. This indicated that the proposed approach allows for further policy optimization, which broadens the known scenario distribution and simultaneously did not lead to catastrophic forgetting of knowledge obtained during offline training. Therefore, the evaluation results supported the hypothesis that proposed combination of offline and online reinforcement learning techniques using real data could lead to better policy efficiency under real-world conditions, outperforming pure simulation-based approaches.

8.3 Future Work

The behavior model training will be continued to improve policy and achieve a stable and precise solution. The study was limited to the initially available data, and the conducted evaluation suggested increasing the portfolio of simulated scenarios. Therefore various data will be collected including urban environments. Besides, more focus will be put on a randomized scenario generation process. A portfolio of scenarios will be specified in more detail, and agent performance will be tracked separately for each of them, enabling the balancing of scenario distribution for training to achieve comparable performance in all situations. Additionally, a continuous learning process will be employed to extend the portfolio of training scenarios according to the evaluation results and further project requirements, as RL policy optimization is a joint optimization of policy and training data.

Furthermore, the objective is to decrease the sim2real gap by improving the realism of the simulation. It is evident that the RL agent can exploit specific simulation aspects in its favor, and if the expected behavior does not manifest in reality, the agent may select unfeasible actions. Thus, we will initiate the improvement of the behavior model of simulated agents. To achieve this, we may utilize driving data to imitate natural driving behaviors. Additionally, we are developing sensor models to distort perception intentionally. As initial work [128] has demonstrated, it is promising to introduce measurement errors, such as inaccuracies in position, velocity, and dimension, as well as false positive and false negative object detections.

Depending on the project's progression and future requirements, we may consider utilizing our modular ANN to address traditional ACC problems such as target selection, object trajectory prediction, lane assignment, and intention prediction. The ANN presented in Section 4.3 has been designed to be extended with additional prediction heads. The transformer encoder in the perception module is capable of comprehending the relationships between the input objects. Furthermore, additional supervised learning of individual modules would support the policy in gaining a better understanding of the situation.

Appendix A

Agent Explainability

A.1 Maneuver agent: boxplots

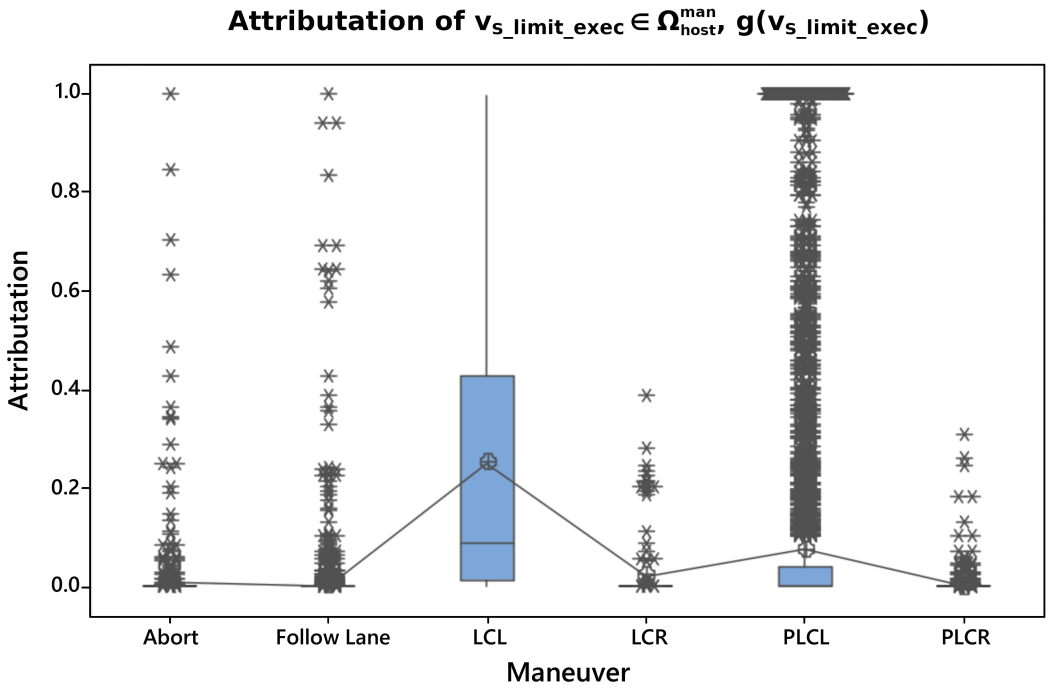


Figure A.1: Attributation distribution of $v_{s\_limit\_exec}$ input feature for all types of maneuvers.

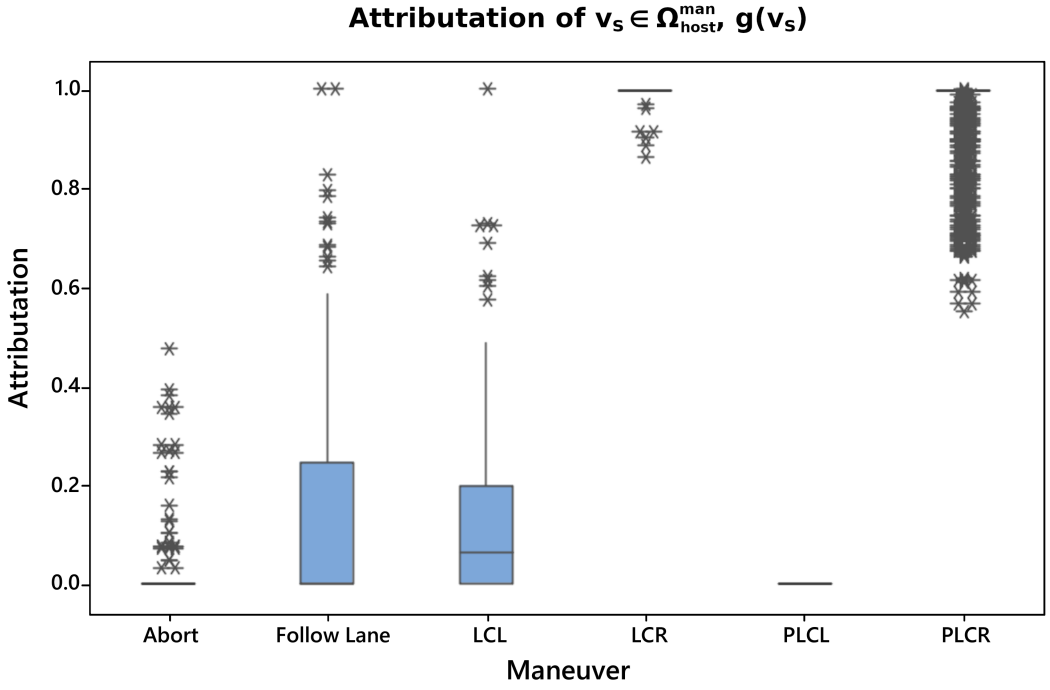


Figure A.2: Attributation distribution of v_s input feature for all types of maneuvers.

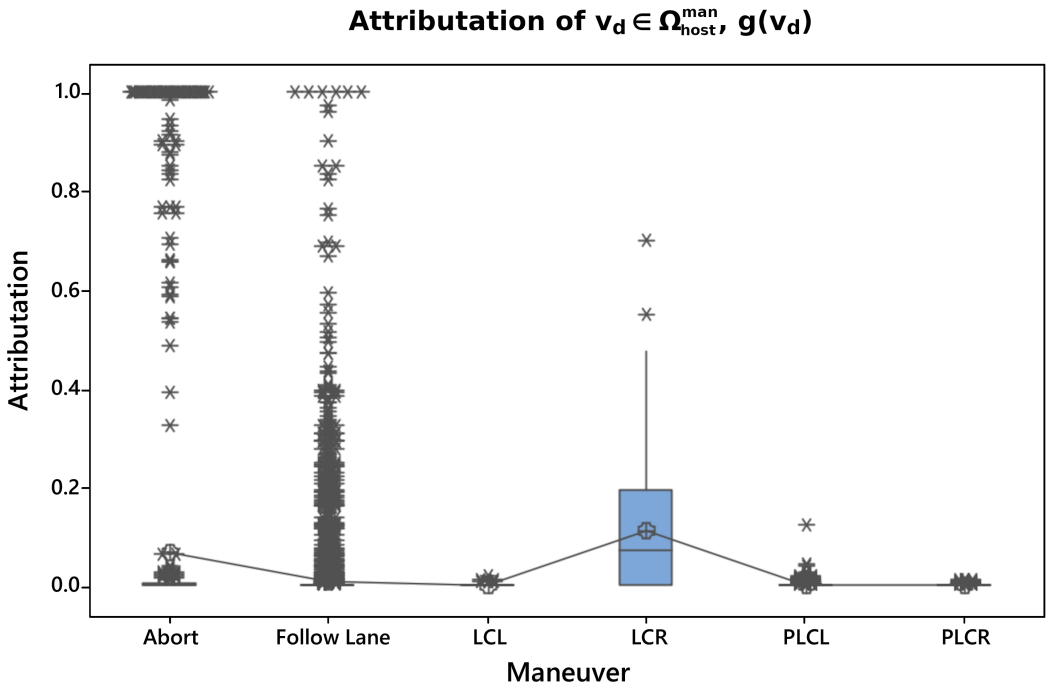


Figure A.3: Attributation distribution of v_d input feature for all types of maneuvers.

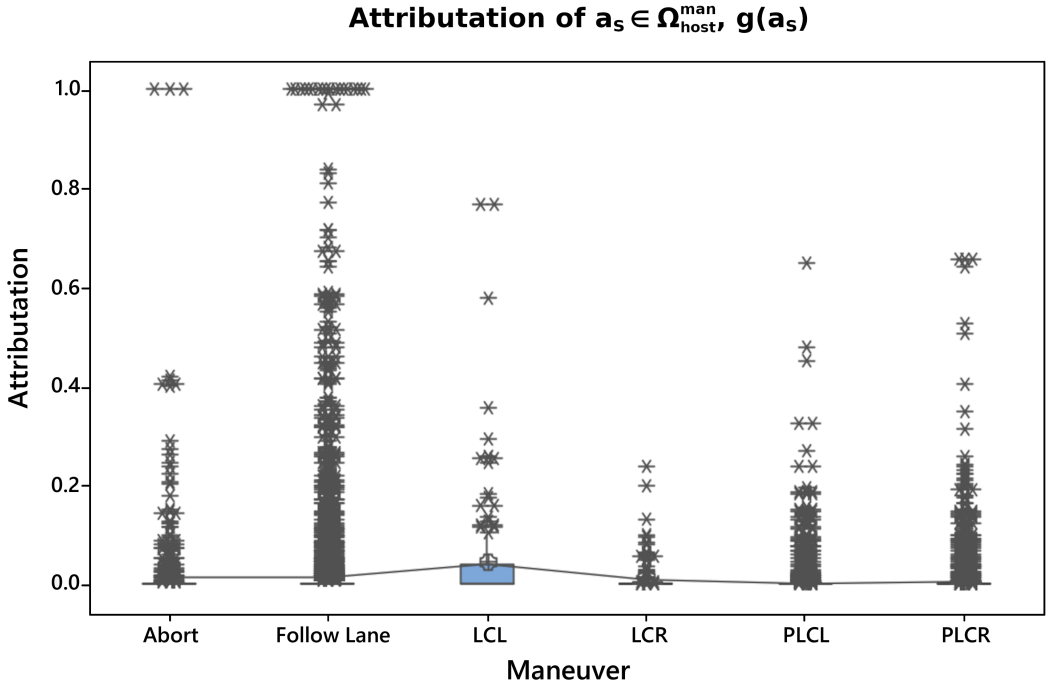


Figure A.4: Attributation distribution of a_s input feature for all types of maneuvers.

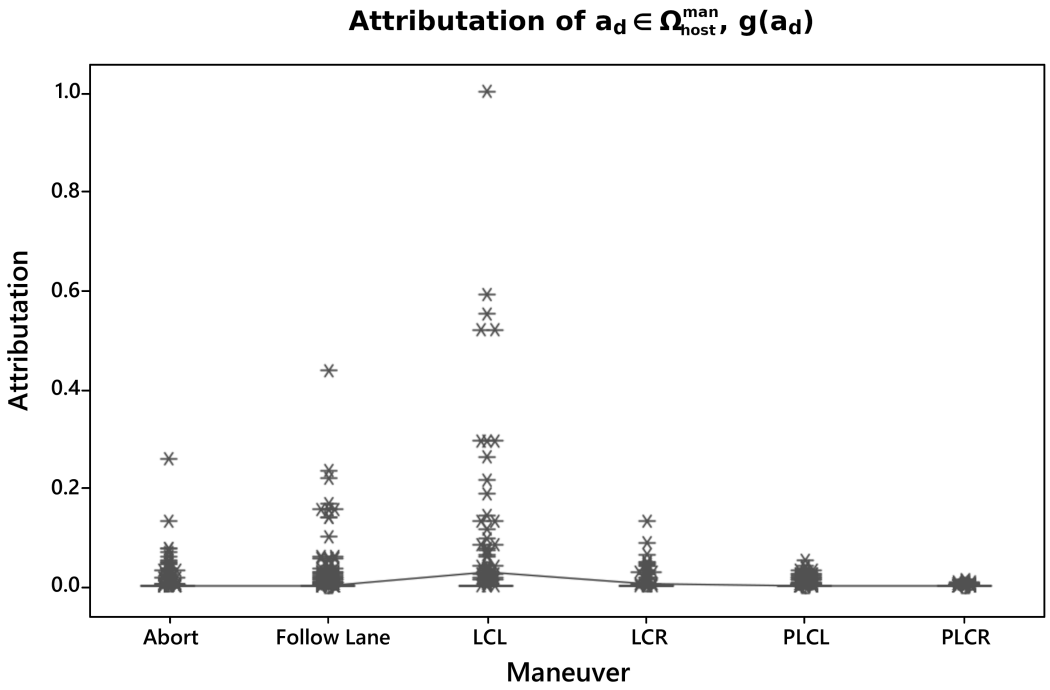


Figure A.5: Attributation distribution of a_d input feature for all types of maneuvers.

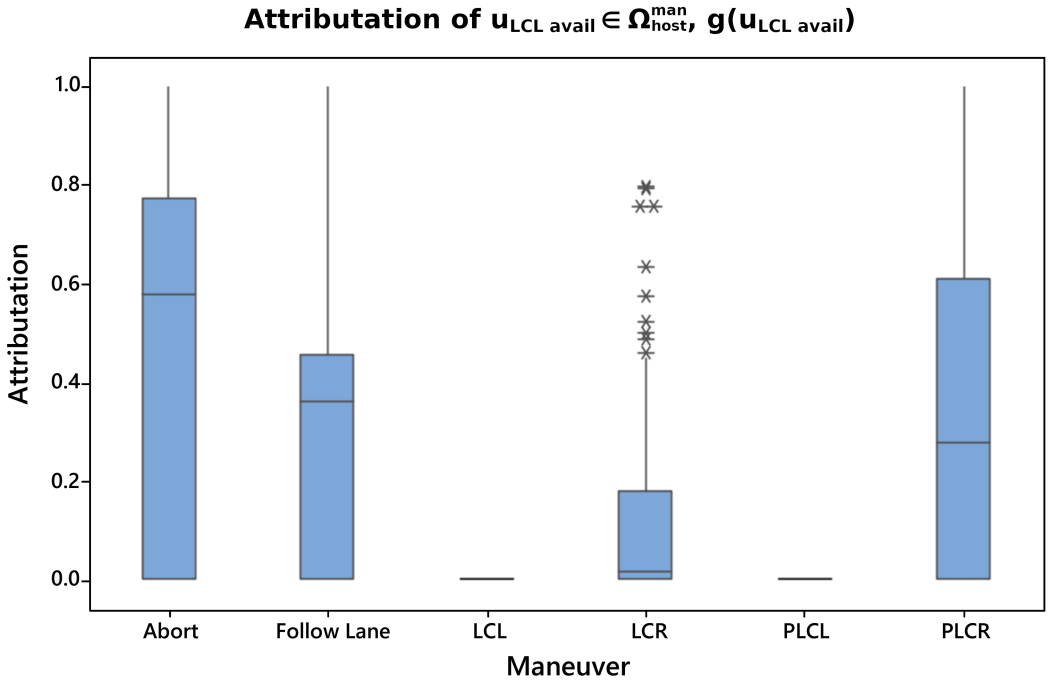


Figure A.6: Attributation distribution of u_{LCLavail} input feature for all types of maneuvers.

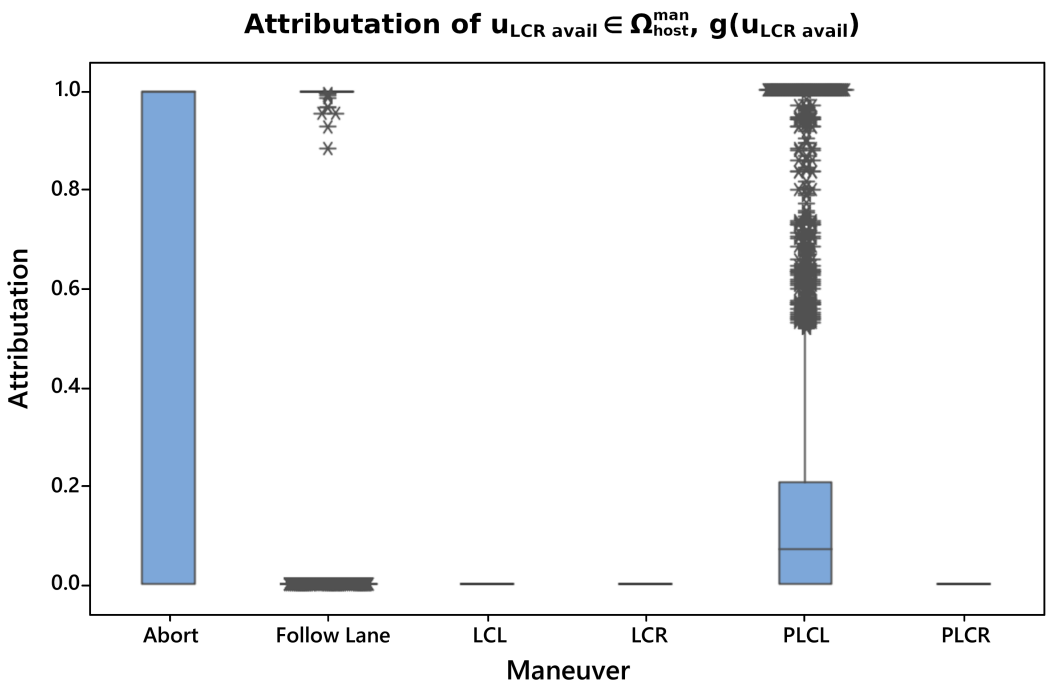


Figure A.7: Attributation distribution of u_{LCRavail} input feature for all types of maneuvers.

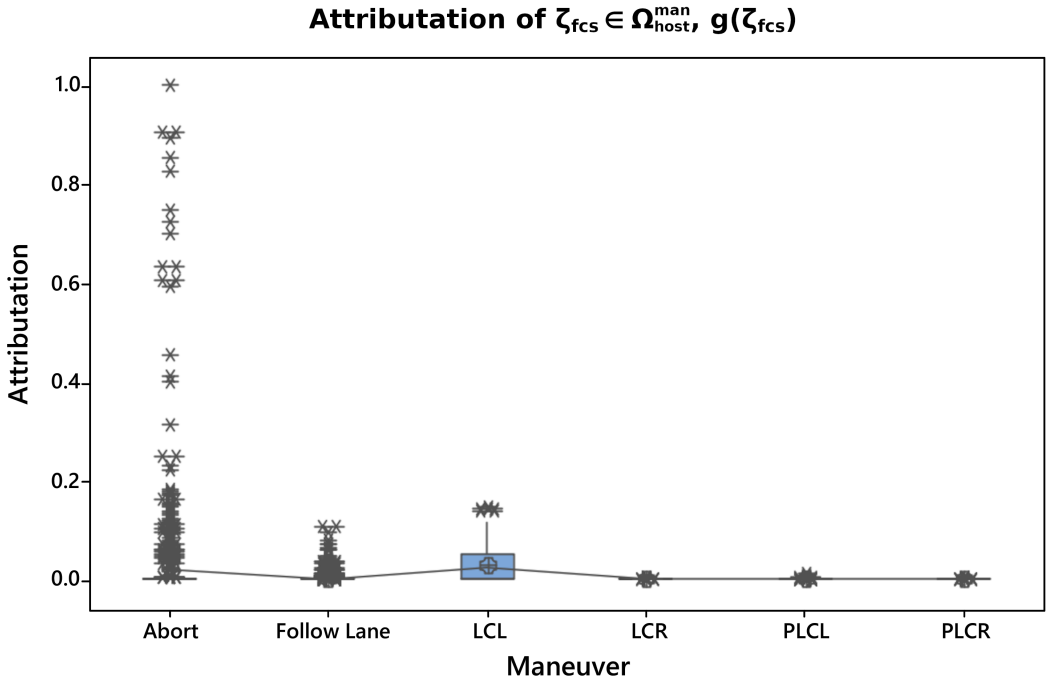


Figure A.8: Attributation distribution of ζ_{fcs} input feature for all types of maneuvers.

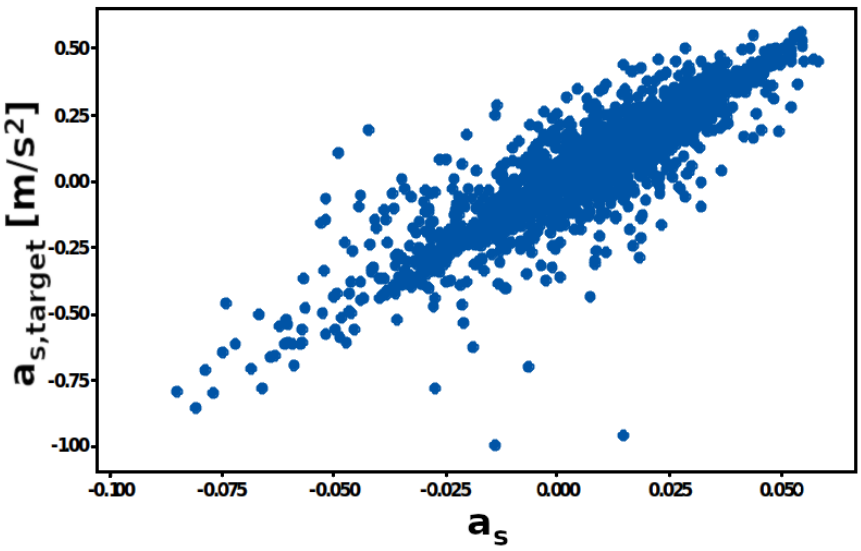


Figure A.9: Scatterplot of target acceleration $a_{s,\text{target}}$ vs normalized host acceleration a_s in current time step.

A.2 Acc agent: scatterplots

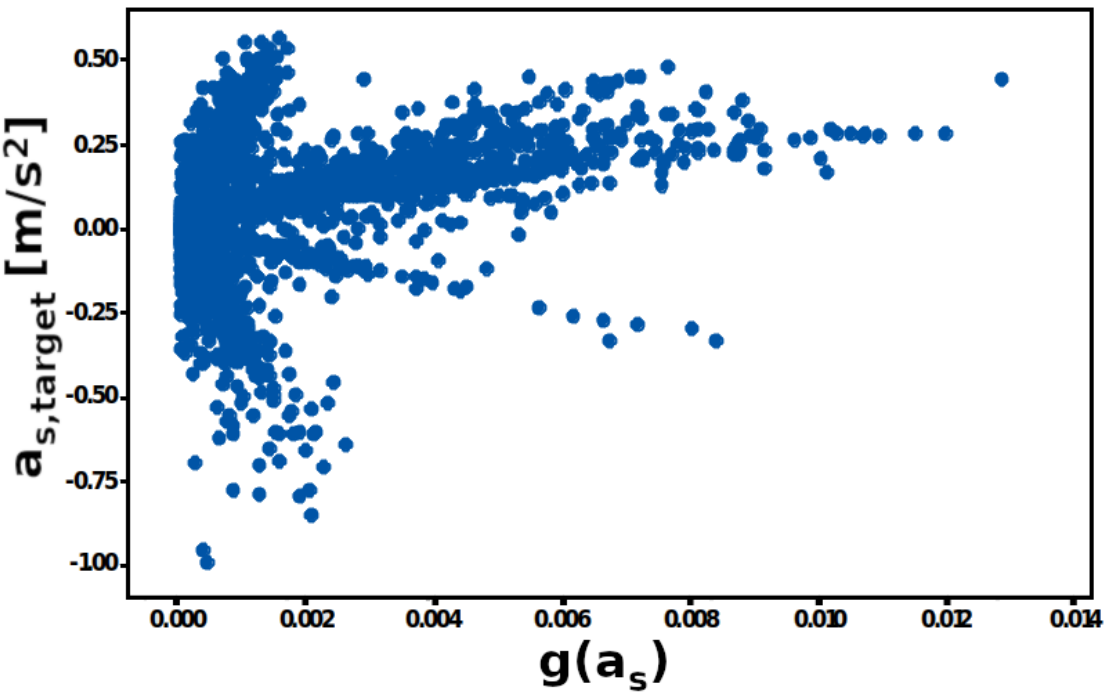


Figure A.10: Scatterplot of target acceleration $a_{s,\text{target}}$ vs attributation of host acceleration $g(a_s)$.

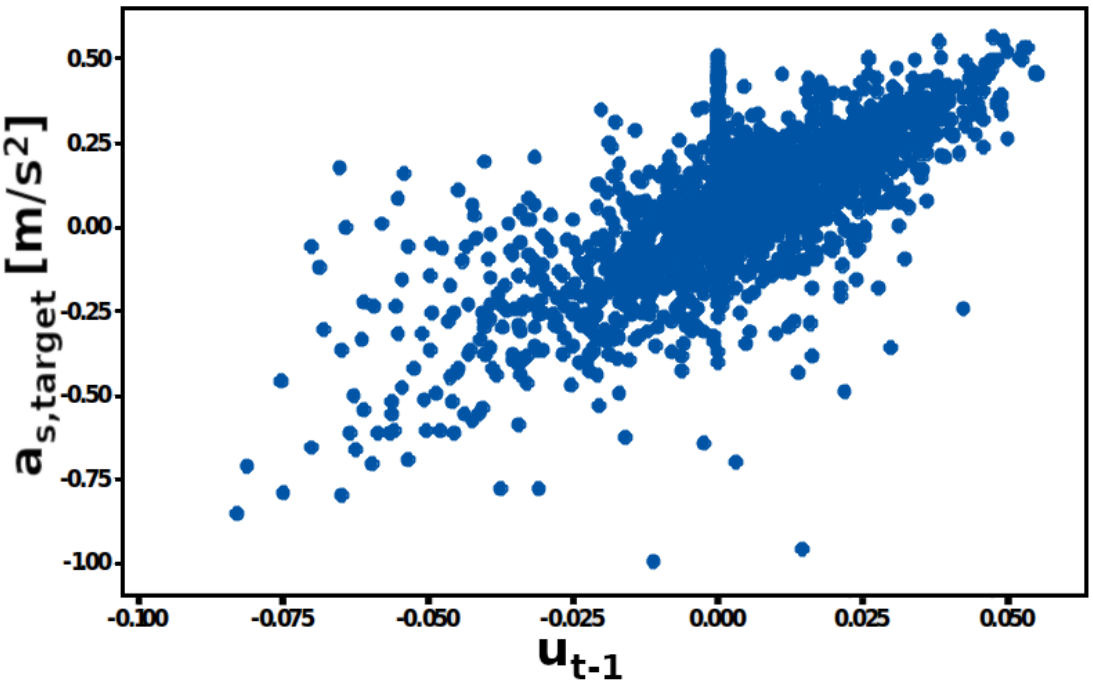


Figure A.11: Scatterplot of target acceleration $a_{s,\text{target}}$ vs action selected in previous time step u_{t-1}.

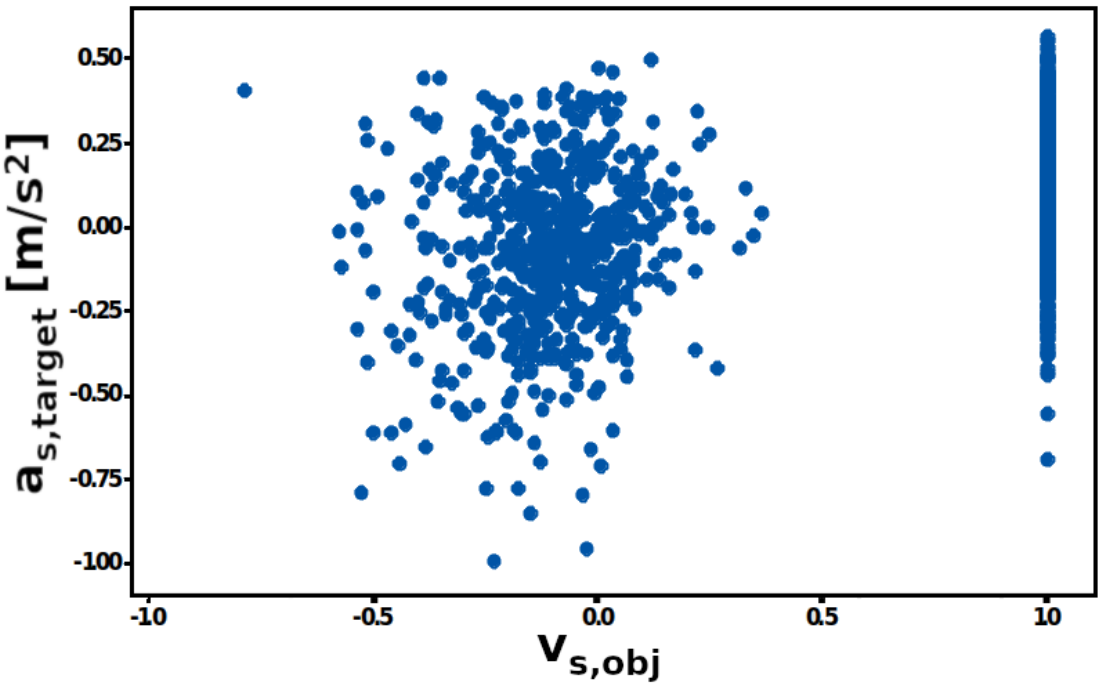


Figure A.12: Scatterplot of target acceleration $a_{s,\text{target}}$ vs objects longitudinal velocity $v_{s,\text{obj}}$. Values of velocity are normalized in range $[-1, 1]$.

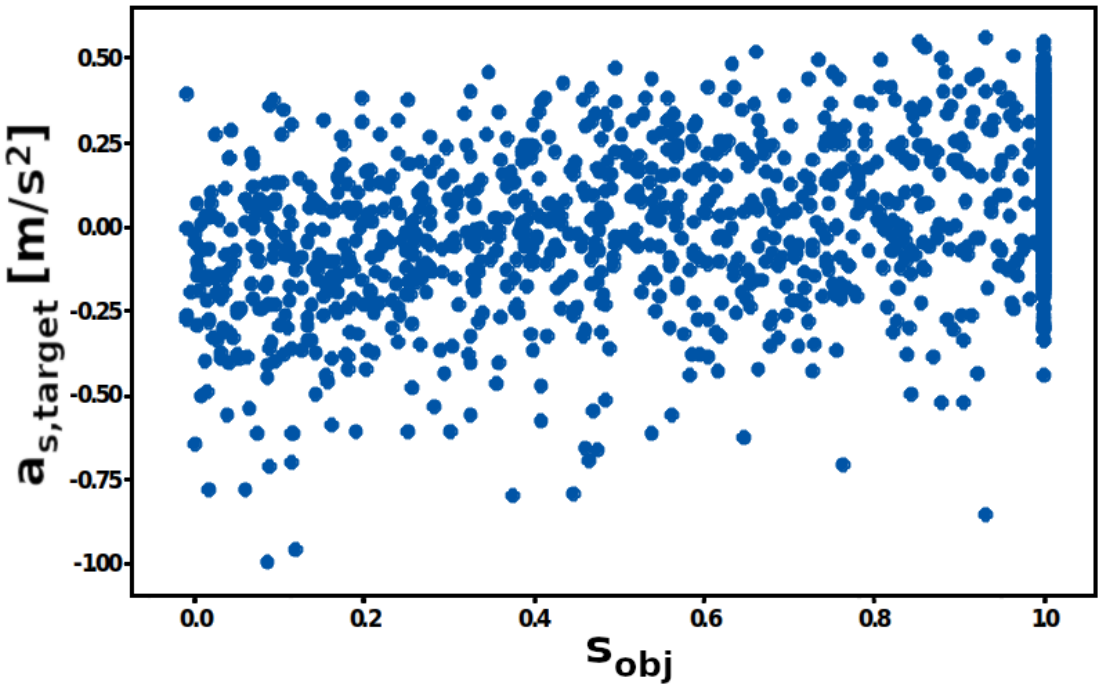


Figure A.13: Scatterplot of target acceleration $a_{s,\text{target}}$ vs objects' longitudinal position s_{obj}.

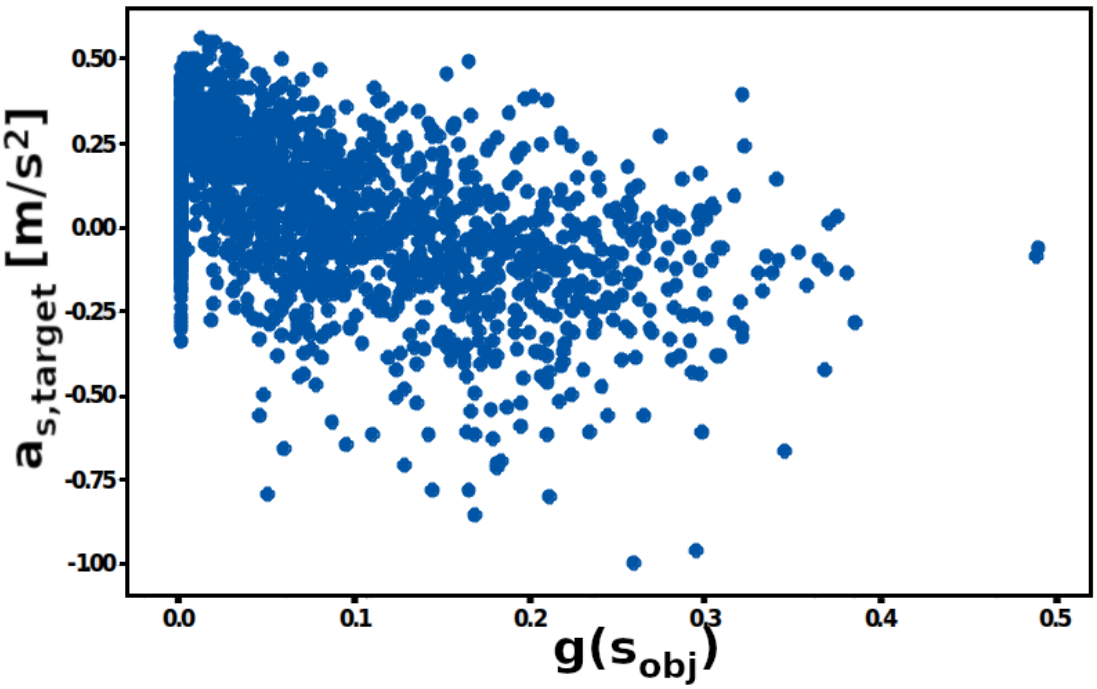


Figure A.14: Scatterplot of target acceleration $a_{s,\mathrm{target}}$ vs attributation of objects' longitudinal position $g(s_{\mathrm{obj}})$

Bibliography

[1] F Frenet. Sur les courbes à double courbure. *Journal de mathématiques pures et appliquées*, 17:437–447, 1852.

[2] On-Road Automated Driving (ORAD) committee. *Taxonomy and Definitions for Terms Related to Driving Automation Systems for On-Road Motor Vehicles*, apr 2021.

[3] Nikodem Pankiewicz, Tomasz Wrona, Wojciech Turlej, and Mateusz Orłowski. Promises and challenges of reinforcement learning applications in motion planning of automated vehicles. In Leszek Rutkowski, Rafał Scherer, Marcin Korytkowski, Witold Pedrycz, Ryszard Tadeusiewicz, and Jacek M. Zurada, editors, *Artificial Intelligence and Soft Computing*, pages 318–329, Cham, 2021. Springer International Publishing.

[4] Pin Wang and Ching Yao Chan. Formulation of deep reinforcement learning architecture toward autonomous driving for on-ramp merge. In *IEEE Conference on Intelligent Transportation Systems, Proceedings, ITSC*, volume 2018-March, pages 1–6. Institute of Electrical and Electronics Engineers Inc., 3 2018.

[5] Daniel Chi Kit Ngai and Nelson Hon Ching Yung. A multiple-goal reinforcement learning method for complex vehicle overtaking maneuvers. In *IEEE Transactions on Intelligent Transportation Systems*, volume 12, pages 509–522, 6 2011.

[6] Konstantinos Makantasis, Maria Kontorinaki, and Ioannis Nikolos. A Deep Reinforcement-Learning-based Driving Policy for Autonomous Road Vehicles. 7 2019.

[7] B. Ravi Kiran, Ibrahim Sobh, Victor Talpaert, Patrick Mannion, Ahmad A.Al Sallab, Senthil Yogamani, and Patrick Perez. Deep reinforcement learning for autonomous driving: A survey. *IEEE Transactions on Intelligent Transportation Systems*, pages 1–18, 2021.

[8] Mateusz Orłowski, Tomasz Wrona, Nikodem Pankiewicz, and Wojciech Turlej. Safe and goal-based highway maneuver planning with reinforcement learning. In Andrzej Bartoszewicz, Jacek Kabziński, and Janusz Kacprzyk, editors, *Advanced, Contemporary Control*, pages 1261–1274, Cham, 2020. Springer International Publishing.

[9] Dean a Pomerleau. ALVINN: an autonomous land vehicle in a neural network (Technical Report CMU-CS-89-107). *Advances in Neural Information Processing Systems 1*, pages 305–313, 1989.

[10] Urs Muller, Jan Ben, Eric Cosatto, Beat Flepp, and Yann L. Cun. Off-road obstacle avoidance through end-to-end learning. In Y. Weiss, B. Schölkopf, and J. C. Platt, editors, *Advances in Neural Information Processing Systems 18*, pages 739–746. MIT Press, 2006.

[11] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D. Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, Xin Zhang, Jake Zhao, and Karol Zieba. End to end learning for self-driving cars. *CoRR*, abs/1604.07316, 2016.

[12] Voot Tangkaratt, Ning Xie, and Masashi Sugiyama. Conditional density estimation with dimensionality reduction via squared-loss conditional entropy minimization. *CoRR*, abs/1404.6876, 2014.

[13] Mayank Bansal, Alex Krizhevsky, and Abhijit Ogale. ChauffeurNet: Learning to Drive by Imitating the Best and Synthesizing the Worst. 2019.

[14] Matt Vitelli, Yan Chang, Yawei Ye, Maciej Wołczyk, Błażej Osiński, Moritz Niendorf, Hugo Grimmett, Qiangui Huang, Ashesh Jain, and Peter Ondruska. SafetyNet: Safe planning for real-world self-driving vehicles using machine-learned policies. 2021.

[15] A. Shaout and M.A. Jarrah. Cruise control technology review. *Computers Electrical Engineering*, 23(4):259–271, 1997.

[16] Colleen Serafin. Driver preferences and usability of adjustable distance controls for an adaptive cruise control (acc) system. 1996.

[17] Yinglong He, Biagio Ciuffo, Quan Zhou, Michail Makridis, Konstantinos Mattas, Ji Li, Ziyang Li, Fuwu Yan, and Hongming Xu. Adaptive cruise control strategies implemented on experimental vehicles: A review. *IFAC-PapersOnLine*, 52(5):21–27, 2019. 9th IFAC Symposium on Advances in Automotive Control AAC 2019.

[18] Changwoo Park and Hyeongcheol Lee. A study of adaptive cruise control system to improve fuel efficiency. *Avestia Publishing International Journal of Environmental Pollution and Remediation*, 5:1929–2732, 2017.

[19] Taku Takahama and Daisuke Akasaka. Model predictive control approach to design practical adaptive cruise control for traffic jam. *International Journal of Automotive Engineering*, 9(3):99–104, 2018.

[20] Mario Zanon, Janick V. Frasch, Milan Vukov, Sebastian Sager, and Moritz Diehl. *Model Predictive Control of Autonomous Vehicles*, pages 41–57. Springer International Publishing, Cham, 2014.

[21] Li Hua Luo, Hong Liu, Ping Li, and Hui Wang. Model predictive control for adaptive cruise control with multi-objectives: Comfort, fuel-economy, safety and car-following. *Journal of Zhejiang University: Science A*, 11:191–201, 3 2010.

[22] Mostafa Al-Gabalawy, Nesreen S. Hosny, and Abdel-hamid S. Aborisha. Model predictive control for a basic adaptive cruise control. *International Journal of Dynamics and Control*, 9(3):1132–1143, Sep 2021.

[23] Nagayasu Maruyama and Hiroshi Mouri. A proposal for adaptive cruise control balancing followability and comfortability through reinforcement learning. *ROBOMECH Journal*, 9(1):22, Dec 2022.

[24] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic policy gradient algorithms. In Eric P. Xing and Tony Jebara, editors, *Proceedings of the 31st International Conference on Machine Learning*, volume 32 of *Proceedings of Machine Learning Research*, pages 387–395, Bejing, China, 22–24 Jun 2014. PMLR.

[25] Meixin Zhu, Yinhai Wang, Ziyuan Pu, Jingyun Hu, Xuesong Wang, and Ruimin Ke. Safe, efficient, and comfortable velocity control based on reinforcement learning for autonomous driving. *Transportation Research Part C: Emerging Technologies*, 117:102662, 2020.

[26] U.S. Department of Transportation Federal Highway Administration. Next generation simulation (ngsim) vehicle trajectories and supporting data. (ngsim) vehicle trajectories and supporting data, 2016. `http://doi.org/10.21949/1504477`.

[27] Lokesh Chandra Das and Myounggyu Won. Saint-acc: Safety-aware intelligent adaptive cruise control for autonomous vehicles using deep reinforcement learning. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 2445–2455. PMLR, 18–24 Jul 2021.

[28] Seungwuk Moon, Kyongsu Yi, and HyongJin Kang. Multi-vehicle adaptive cruise control with collision avoidance in multiple transitions. *IFAC Proceedings Volumes*, 42(15):304–311, 2009. 12th IFAC Symposium on Control in Transportation Systems.

[29] Seungwuk Moon, Hyoung-Jin Kang, and Kyongsu Yi. Multi-vehicle target selection for adaptive cruise control. *Vehicle System Dynamics*, 48(11):1325–1343, 2010.

[30] Jun Yao, Guoying Chen, and Zhenhai Gao. Target vehicle selection algorithm for adaptive cruise control based on lane-changing intention of preceding vehicle. *Chinese Journal of Mechanical Engineering*, 34(1):135, Dec 2021.

[31] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, 1995.

[32] Sajjad Mozaffari, Eduardo Arnold, Mehrdad Dianati, and Saber Fallah. Early lane change prediction for automated driving systems using multi-task attention-based convolutional neural networks. *CoRR*, abs/2109.10742, 2021.

[33] Donghan Lee, Youngwook Paul Kwon, Sara McMains, and J. Karl Hedrick. Convolution neural network-based lane change intention prediction of surrounding vehicles for acc. In *2017 IEEE 20th International Conference on Intelligent Transportation Systems (ITSC)*, pages 1–6, 2017.

[34] Neha Konakalla, Avrum Noor, and Josh Singh. Cnn, cnn encoder-rnn decoder, and pretrained vision transformers for surrounding vehicle lane change classification at future time steps. 2022.

[35] Mustafa Yildirim, Sajjad Mozaffari, Lucy McCutcheon, Mehrdad Dianati, and Alireza Tamaddoni-Nezhad Saber Fallah. Prediction based decision making for autonomous highway driving. *2022 IEEE 25th International Conference on Intelligent Transportation Systems (ITSC)*, pages 138–145, 2022.

[36] Zhenpei Yang, Yuning Chai, Dragomir Anguelov, Yin Zhou, Pei Sun, Dumitru Erhan, Sean Rafferty, Henrik Kretzschmar, Ut Austin, and Google Brain. SurfelGAN: Synthesizing Realistic Sensor Data for Autonomous Driving.

[37] Xinlei Pan, Yurong You, Ziyan Wang, and Cewu Lu. Virtual to Real Reinforcement Learning for Autonomous Driving. 2017.

[38] Haoyi Niu, Jianming Hu, Zheyu Cui, and Yi Zhang. DR 2 L: Surfacing Corner Cases to Robustify Autonomous Driving via Domain Randomization Reinforcement Learning.

[39] Guan Wang, Haoyi Niu, Desheng Zhu, Jianming Hu, Xianyuan Zhan, and Guyue Zhou. ModEL: A Modularized End-to-end Reinforcement Learning Framework for Autonomous Driving.

[40] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, second edition, 2018.

[41] Nikodem Pankiewicz and Paweł Kowalczyk. Attributation analysis of reinforcement learning-based highway driver. *Electronics*, 11(21), 2022.

[42] Takashi Nagata and Masayoshi Tomizuka. Engine torque control based on discrete event model and disturbance observer. In *ASME International Mechanical Engineering Congress and Exposition*, volume 43106, pages 43–52, 2007.

[43] Takashi Nagata and Masayoshi Tomizuka. Robust engine torque control by discrete event disturbance observer. *IFAC Proceedings Volumes*, 41(2):9473–9478, 2008.

[44] Junqing Wei, Jarrod M. Snider, Tianyu Gu, John M. Dolan, and Bakhtiar Litkouhi. A behavioral planning framework for autonomous driving. In *IEEE Intelligent Vehicles Symposium, Proceedings*, pages 458–464. Institute of Electrical and Electronics Engineers Inc., 2014.

[45] Bo Yang, Ishan Khatri, Michael Happold, and Chulong Chen. Adcnet: End-to-end perception with raw radar adc data. *ArXiv*, abs/2303.11420, 2023.

[46] Li Wang, Jun Tang, and Qingmin Liao. A study on radar target detection based on deep neural networks. *IEEE Sensors Letters*, 3(3):1–4, 2019.

[47] Bence Major, Daniel Fontijne, Amin Ansari, Ravi Teja Sukhavasi, Radhika Gowaikar, Michael Hamilton, Sean Lee, Slawomir Grzechnik, and Sundar Subramanian. Vehicle detection with automotive radar using deep learning on range-azimuth-doppler tensors. In *2019 IEEE/CVF International Conference on Computer Vision Workshop (ICCVW)*, pages 924–932, 2019.

[48] Felix Nobis, Maximilian Geisslinger, Markus Weber, Johannes Betz, and Markus Lienkamp. A deep learning-based radar and camera sensor fusion architecture for object detection. *2019 Sensor Data Fusion: Trends, Solutions, Applications (SDF)*, Oct 2019.

[49] Jingwei Zhang, Ming Zhang, Zicheng Fang, Yulong Wang, Xian Zhao, and Shiliang Pu. Rvdet: Feature-level fusion of radar and camera for object detection. In *2021 IEEE International Intelligent Transportation Systems Conference (ITSC)*, pages 2822–2828, 2021.

[50] Traffic AI - Simteract - simteract.com/traffic-ai/.

[51] OpenStreetMap contributors. Planet dump retrieved from https://planet.osm.org . `https://www.openstreetmap.org`, 2017.

[52] Philip Polack, Florent Altché, Brigitte d'Andréa Novel, and Arnaud de La Fortelle. The kinematic bicycle model: A consistent model for planning feasible trajectories for autonomous vehicles? In *2017 IEEE intelligent vehicles symposium (IV)*, pages 812–818. IEEE, 2017.

[53] Jason Kong, Mark Pfeiffer, Georg Schildbach, and Francesco Borrelli. Kinematic and dynamic vehicle models for autonomous driving control design. In *2015 IEEE Intelligent Vehicles Symposium (IV)*, pages 1094–1099. IEEE, 2015.

[54] Zhenhui Yao, Ric Mousseau, and Ben G Kao. A powertrain model for real-time vehicle simulation. In *Department of Mechanical, Industrial and Manufacturing Engineering, The University of Toledo, Ford Research Company. DSC North America Proceedings s*, volume 16, 2003.

[55] Alexey Dosovitskiy, Germán Ros, Felipe Codevilla, Antonio M. López, and Vladlen Koltun. CARLA: an open urban driving simulator. *CoRR*, abs/1711.03938, 2017.

[56] N. Koenig and A. Howard. Design and use paradigms for gazebo, an open-source multi-robot simulator. In *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)*, volume 3, pages 2149–2154 vol.3, 2004.

[57] Shai Shalev-Shwartz, Shaked Shammah, and Amnon Shashua. On a Formal Model of Safe and Scalable Self-driving Cars. 8 2017.

[58] Richard Bellman. A markovian decision process. *Indiana Univ. Math. J.*, 6:679–684, 1957.

[59] K.J Åström. Optimal control of markov processes with incomplete state information. *Journal of Mathematical Analysis and Applications*, 10(1):174–205, 1965.

[60] Richard Bellman. On the Theory of Dynamic Programming. *Proceedings of the National Academy of Sciences*, 38(8):716–719, 1952.

[61] Sylvain Gelly and David Silver. Monte-carlo tree search and rapid action value estimation in computer go. *Artificial Intelligence*, 175(11):1856–1875, 2011.

[62] Christopher J. C. H. Watkins and Peter Dayan. Q-learning. *Machine Learning*, 8(3):279–292, May 1992.

[63] Christopher J C H Watkins and Peter Dayan. Technical note q,-learning. 8:279–292, 1992.

[64] Tommi Jaakkola, Michael Jordan, and Satinder Singh. Convergence of stochastic iterative dynamic programming algorithms. *Advances in neural information processing systems*, 6, 1993.

[65] Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8(3-4):229–256, 1992.

[66] Vijay Konda and John Tsitsiklis. Actor-critic algorithms. *Advances in neural information processing systems*, 12, 1999.

[67] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. *arXiv preprint arXiv:1511.05952*, 2015.

[68] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017.

[69] John Schulman, Sergey Levine, Philipp Moritz, Michael I. Jordan, and Pieter Abbeel. Trust region policy optimization. *CoRR*, abs/1502.05477, 2015.

[70] Faraz Torabi, Garrett Warnell, and Peter Stone. Behavioral cloning from observation. *IJCAI International Joint Conference on Artificial Intelligence*, 2018-July(July):4950–4957, 2018.

[71] Qing Wang, Jiechao Xiong, Lei Han, Peng Sun, Han Liu, and Tong Zhang. MARWIL Exponentially Weighted Imitation Learning for Batched Historical Data. In S Bengio, H Wallach, H Larochelle, K Grauman, N Cesa-Bianchi, and R Garnett, editors, *Advances in Neural Information Processing Systems 31*, pages 6288–6297. Curran Associates, Inc., 2018.

[72] Doina Precup, Richard Sutton, and Satinder Singh. Eligibility traces for off-policy policy evaluation. *Computer Science Department Faculty Publication Series*, 06 2000.

[73] Philip S. Thomas, Georgios Theocharous, and Mohammad Ghavamzadeh. High confidence off-policy evaluation. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, AAAI'15, page 3000–3006. AAAI Press, 2015.

[74] Scott Fujimoto, David Meger, and Doina Precup. Off-policy deep reinforcement learning without exploration. *CoRR*, abs/1812.02900, 2018.

[75] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *CoRR*, abs/1801.01290, 2018.

[76] Aviral Kumar, Justin Fu, George Tucker Google Brain, and Sergey Levine. Stabilizing Off-Policy Q-Learning via Bootstrapping Error Reduction.

[77] Michael Bain and Claude Sammut. A framework for behavioural cloning. In *Machine Intelligence 15*, 1995.

[78] Yevgen Chebotar, Ankur Handa, Viktor Makoviychuk, Miles Macklin, Jan Issac, Nathan Ratliff, and Dieter Fox. Closing the Sim-to-Real Loop: Adapting Simulation Randomization with Real World Experience. *Proceedings - IEEE International Conference on Robotics and Automation*, 2019-May:8973–8979, oct 2018.

[79] Dan Hendrycks, Nicholas Carlini, John Schulman, and Jacob Steinhardt. Unsolved Problems in ML Safety. sep 2021.

[80] Quan Vuong, Sharad Vikram, Hao Su, Sicun Gao, and Henrik I. Christensen. How to pick the domain randomization parameters for sim-to-real transfer of reinforcement learning policies? mar 2019.

[81] Xue Bin Peng, Marcin Andrychowicz, Wojciech Zaremba, and Pieter Abbeel. Sim-to-real transfer of robotic control with dynamics randomization. *CoRR*, abs/1710.06537, 2017.

[82] OpenAI, Ilge Akkaya, Marcin Andrychowicz, Maciek Chociej, Mateusz Litwin, Bob McGrew, Arthur Petron, Alex Paino, Matthias Plappert, Glenn Powell, Raphael Ribas, Jonas Schneider, Nikolas Tezak, Jerry Tworek, Peter Welinder, Lilian Weng, Qiming Yuan, Wojciech Zaremba, and Lei Zhang. Solving rubik's cube with a robot hand. *CoRR*, abs/1910.07113, 2019.

[83] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

[84] Kanishka Rao, Chris Harris, Alex Irpan, Sergey Levine, Julian Ibarz, and Mohi Khansari. RL-CycleGAN: Reinforcement Learning Aware Simulation-To-Real.

[85] Hado Van Hasselt, Arthur Guez, and David Silver. Deep Reinforcement Learning with Double Q-learning.

[86] Katie Kang, Suneel Belkhale, Gregory Kahn, Pieter Abbeel, Sergey Levine, and Berkeley Ai Research. Generalization through Simulation: Integrating Simulated and Real Data into Deep Reinforcement Learning for Vision-Based Autonomous Flight.

[87] Natasha Jaques, Asma Ghandeharioun, Judy Hanwen Shen, Craig Ferguson, Agata Lapedriza, Noah Jones, Shixiang Gu, and Rosalind Picard. Way Off-Policy Batch Deep Reinforcement Learning of Implicit Human Preferences in Dialog.

[88] Scott Fujimoto, Herke Van Hoof, and David Meger. Addressing Function Approximation Error in Actor-Critic Methods. 2018.

[89] Yarin Gal and Zg201@cam Ac Uk. Dropout as a Bayesian Approximation: Representing Model Uncertainty in Deep Learning Zoubin Ghahramani. 2016.

[90] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. OpenAI Gym. *CoRR*, abs/1606.01540, 2016.

[91] Oriol Vinyals, Igor Babuschkin, Wojciech M. Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H. Choi, Richard Powell, Timo Ewalds, Petko Georgiev, Junhyuk Oh, Dan Horgan, Manuel Kroiss, Ivo Danihelka, Aja Huang, Laurent Sifre, Trevor Cai, John P. Agapiou, Max Jaderberg, Alexander S. Vezhnevets, Rémi Leblond, Tobias Pohlen, Valentin Dalibard, David Budden, Yury Sulsky, James Molloy, Tom L. Paine, Caglar Gulcehre, Ziyu Wang, Tobias Pfaff, Yuhuai Wu, Roman Ring, Dani Yogatama, Dario Wünsch, Katrina McKinney, Oliver Smith, Tom Schaul, Timothy Lillicrap, Koray Kavukcuoglu, Demis Hassabis, Chris Apps, and David Silver. Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature 2019 575:7782*, 575:350–354, 10 2019.

[92] Krzysztof Czarnecki. Automated driving system (ads) high-level quality requirements analysis - driving behavior comfort, 07 2018.

[93] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805, 2018.

[94] Charles Ruizhongtai Qi, Hao Su, Kaichun Mo, and Leonidas J. Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. *CoRR*, abs/1612.00593, 2016.

[95] Peter W. Battaglia, Jessica B. Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinícius Flores Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, Çaglar Gülçehre, H. Francis Song, Andrew J. Ballard, Justin Gilmer, George E. Dahl, Ashish Vaswani, Kelsey R. Allen, Charles Nash, Victoria Langston, Chris Dyer, Nicolas Heess, Daan Wierstra, Pushmeet Kohli, Matthew M. Botvinick, Oriol Vinyals, Yujia Li, and Razvan Pascanu. Relational inductive biases, deep learning, and graph networks. *CoRR*, abs/1806.01261, 2018.

[96] Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. Neural message passing for quantum chemistry. *CoRR*, abs/1704.01212, 2017.

[97] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9:1735–1780, 11 1997.

[98] Jiyang Gao, Chen Sun, Hang Zhao, Yi Shen, Dragomir Anguelov, Congcong Li, and Cordelia Schmid. Vectornet: Encoding hd maps and agent dynamics from vectorized representation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

[99] Shapefiles - ArcGIS - https://doc.arcgis.com/en/arcgis-online/reference/shapefiles.htm.

[100] ESRI Shapefile Technical Description - https://www.esri.com/content/dam/esrisites/sitecore-archive/Files/Pdfs/library/whitepapers/pdfs/shapefile.pdf.

[101] Matthias Fey and Jan E. Lenssen. Fast graph representation learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.

[102] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[103] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. *CoRR*, abs/2010.11929, 2020.

[104] Marcin Andrychowicz, Anton Raichuk, Piotr Stańczyk, Manu Orsini, Sertan Girgin, Raphaël Marinier, Leonard Hussenot, Matthieu Geist, Olivier Pietquin, Marcin Michalski, Sylvain Gelly, and Olivier Bachem. What matters for on-policy deep actor-critic methods? a large-scale study, 2 2022.

[105] Ustawa z dnia 20 czerwca 1997 r. prawo o ruchu drogowym1. *Dz.U.2023.1047*, 2023.1047, 2023-08-07.

[106] Christopher Berner, Greg Brockman, Brooke Chan, Vicki Cheung, Przemyslaw Debiak, Christy Dennison, David Farhi, Quirin Fischer, Shariq Hashme, Christopher Hesse, Rafal Józefowicz, Scott Gray, Catherine Olsson, Jakub Pachocki, Michael Petrov, Henrique Pondé de Oliveira Pinto, Jonathan Raiman, Tim Salimans, Jeremy Schlatter, Jonas Schneider, Szymon Sidor, Ilya Sutskever, Jie Tang, Filip Wolski, and Susan Zhang. Dota 2 with large scale deep reinforcement learning. *CoRR*, abs/1912.06680, 2019.

[107] Eric Liang, Richard Liaw, Robert Nishihara, Philipp Moritz, Roy Fox, Ken Goldberg, Joseph Gonzalez, Michael Jordan, and Ion Stoica. RLlib: Abstractions for distributed reinforcement learning. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 3053–3062. PMLR, 10–15 Jul 2018.

[108] Nikodem Pankiewicz, Wojciech Turlej, and Mateusz Orlowski. Determining a driving trajectory as training data for a machine learning based adaptive cruise control, April 13 2023. US Patent App. 17/938,232.

[109] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach,

Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of Go with deep neural networks and tree search. *Nature 2016 529:7587*, 529(7587):484–489, 1 2016.

[110] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, Jan 2016.

[111] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy P. Lillicrap, Karen Simonyan, and Demis Hassabis. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *CoRR*, abs/1712.01815, 2017.

[112] Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, Timothy P. Lillicrap, and David Silver. Mastering atari, go, chess and shogi by planning with a learned model. *CoRR*, abs/1911.08265, 2019.

[113] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C J Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020.

[114] Aviral Kumar, Joey Hong, Anikait Singh, and Sergey Levine. Should i run offline reinforcement learning or behavioral cloning? In *International Conference on Learning Representations*, 2022.

[115] Joeri R. Hermans, Gerasimos Spanakis, and Rico Möckel. Accumulated gradient normalization. In Min-Ling Zhang and Yung-Kyun Noh, editors, *Proceedings of the Ninth Asian Conference on Machine Learning*, volume 77 of *Proceedings of Machine Learning Research*, pages 439–454, Yonsei University, Seoul, Republic of Korea, 15–17 Nov 2017. PMLR.

[116] Nikodem Pankiewicz, Wojciech Turlej, Mateusz Orłowski, and Tomasz Wrona. Highway pilot training from demonstration. In *2021 25th International Conference on Methods and Models in Automation and Robotics (MMAR)*, pages 109–114, 2021.

[117] Eric Wiewiora. *Reward Shaping*, pages 863–865. Springer US, Boston, MA, 2010.

[118] Dario Amodei, Chris Olah, Jacob Steinhardt, Paul F. Christiano, John Schulman, and Dan Mané. Concrete problems in AI safety. *CoRR*, abs/1606.06565, 2016.

[119] Joar Skalse, Nikolaus H. R. Howe, Dmitrii Krasheninnikov, and David Krueger. Defining and characterizing reward hacking, 2022.

[120] Mukund Sundararajan, Ankur Taly, and Qiqi Yan. Axiomatic Attribution for Deep Networks. *34th International Conference on Machine Learning, ICML 2017*, 7:5109–5118, 3 2017.

[121] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[122] Narine Kokhlikyan, Vivek Miglani, Miguel Martin, Edward Wang, Bilal Alsallakh, Jonathan Reynolds, Alexander Melnikov, Natalia Kliushkina, Carlos Araya, Siqi Yan, and Orion Reblitz-Richardson. Captum: A unified and generic model interpretability library for pytorch, 2020.

[123] Minitab, LLC. Minitab.

[124] H. Liu. *Comparing Welch's ANOVA, a Kruskal-Wallis Test, and Traditional ANOVA in Case of Heterogeneity of Variance*. Virginia Commonwealth University, 2015.

[125] Derek C. Sauder and Christine E. DeMars. An updated recommendation for multiple comparisons. *Advances in Methods and Practices in Psychological Science*, 2(1):26–44, 2019.

[126] David Freedman, Robert Pisani, and Roger Purves. Statistics (international student edition). *Pisani, R. Purves, 4th edn. WW Norton & Company, New York*, 2007.

[127] Jerrold H Zar. Spearman rank correlation. *Encyclopedia of Biostatistics*, 7, 2005.

[128] Wojciech Turlej. High-level sensor models for the reinforcement learning driving policy training. *Electronics*, 12(1), 2023.