Learning Driver Behavior Models for Predicting Urban Traffic Situations

Lernen von Fahrermodellen zur Prognose urbaner Verkehrssituationen

Der Technischen Fakultät der
Friedrich-Alexander-Universität
Erlangen-Nürnberg

zur Erlangung des

Doktorgrades Dr.-Ing.

vorgelegt von

Moritz Sackmann

aus Göttingen

Als Dissertation genehmigt
von der Technischen Fakultät der
Friedrich-Alexander-Universtität Erlangen-Nürnberg

Tag der mündlichen Prüfung	04.03.2024
Gutachter	Prof. Dr.-Ing. Jörn Thielecke
	Prof. Gustav Markkula, Ph.D.

Abstract

An automated vehicle needs to be able to predict the future evolution of a perceived traffic situation to safely and comfortably interact with surrounding vehicles. This work focuses on generating predictions by executing a traffic simulation. The advantage of this simulation-based prediction is that the predictions of all vehicles are constructed simultaneously and can interact with each other. Moreover, conditional predictions become possible, e.g., "How would the traffic situation evolve, if the automated vehicle merges in front of or behind another vehicle?"

The behavior model is crucial for the accuracy of the prediction. Therefore, this thesis investigates three approaches to learning a behavior model: Multi-Step Behavior Cloning, Reinforcement Learning, and Inverse Reinforcement Learning.

For Multi-Step Behavior Cloning, the behavior model is trained such that it selects an action sequence, and hence a trajectory, as similar as possible to human drivers, starting from the same initial situation. The training requires a differentiable simulation environment, which is introduced in this work.

In contrast, the training goal of Reinforcement Learning (RL) is to maximize a hand-defined reward function. With this, explicit goals can be formulated, such as avoiding collisions, remaining on the road, and maintaining safety distances. A modification of the method is proposed to represent different driving styles with one single behavior model, e.g., sporty or careful driving.

To model human driving with RL, the reward function must be adapted until the resulting trajectories are similar enough to human trajectories. This tedious procedure can be automatized with Inverse Reinforcement Learning (IRL). To this end, Adversarial Inverse Reinforcement Learning (AIRL) is employed. With the reconstructed reward function, the behavior model is trained in additional fictional critical situations to obtain a more robust model.

Finally, all trained models are compared under equal conditions in an untrained roundabout. The IRL algorithms achieve the best results with collision rates below 1% and root mean squared prediction errors (RMSE) below $22\,\mathrm{m}$. RL and IRL reduce the collision rate compared to Behavior Cloning, because they directly penalize collisions beyond the goal of pure imitation.

Zusammenfassung

Ein automatisiertes Fahrzeug muss die Entwicklung einer wahrgenommenen Verkehrssituation vorhersagen können, damit es sicher und komfortabel mit anderen Fahrzeugen interagieren kann. Diese Arbeit untersucht verschiedene Methoden, um Vorhersagen mit einer Simulation der Situation zu erzeugen. Der simulationsbasierte Ansatz ist vorteilhaft, weil die Vorhersagen aller Fahrzeuge gleichzeitig aufgebaut werden und aufeinander reagieren können. Außerdem werden bedingte Vorhersagen möglich, z.B. „Wie entwickelt sich die Situation, wenn sich das automatisierte Fahrzeug vor oder hinter einem anderen Fahrzeug einfädelt?“

Das Verhaltensmodell der simulierten Fahrzeuge hat entscheidenden Einfluss auf die Genauigkeit der Vorhersage. Daher befasst sich diese Dissertation mit drei Ansätzen, um ein Verhaltensmodell zu lernen: Multi-Step Behavior Cloning, Reinforcement Learning und Inverse Reinforcement Learning.

Bei Multi-Step Behavior Cloning wird das Verhaltensmodell so trainiert, dass es ausgehend von derselben Ausgangssituation eine möglichst ähnliche Aktionsfolge und damit Trajektorie wie ein menschlicher Fahrer wählt. Für das Training wird eine differenzierbare Simulationsumgebung benötigt, die in dieser Arbeit vorgestellt wird.

Im Gegensatz dazu ist das Trainingsziel bei Reinforcement Learning (RL) die Maximierung einer händisch definierten Belohnungsfunktion. So können explizite Ziele vorgegeben werden, z.B., dass Fahrzeuge Kollisionen vermeiden, auf der Fahrbahn bleiben und Sicherheitsabstände einhalten. Die Methode wird erweitert, um mit einem Verhaltensmodell unterschiedliche Fahrverhalten zu repräsentieren, z.B. sportlichere oder vorsichtigere Fahrer.

Um menschliches Fahrverhalten mit RL nachzubilden, muss die Belohnungsfunktion so lange angepasst werden, bis die resultierenden Trajektorien ähnlich wie echte Trajektorien aussehen. Dieser aufwändige Prozess wird von Methoden des Inverse Reinforcement Learning (IRL) automatisiert. Hierfür wird unter anderem Adversarial Inverse Reinforcement Learning (AIRL) verwendet. Mit der rekonstruierten Belohnungsfunktion wird das Verhaltensmodell außerdem in fiktiven kritischen Situationen trainiert, um eine höhere Robustheit des Modells zu erreichen.

Abschließend werden alle trainierten Modelle unter gleichen Bedingungen in einem untrainierten Kreisverkehr verglichen. Hierbei schneiden die IRL-Algorithmen bei $10\,\mathrm{s}$-Vorhersagen mit Kollisionsraten unter $1\ \%$ und Vorhersagefehlern (RMSE) unter $22\,\mathrm{m}$ am besten ab. RL und IRL verringern die Kollisionsrate im Vergleich zu Behavior Cloning, weil neben dem Ziel der Imitation des Verhaltens auch Kollisionen direkt bestraft werden.

Acknowledgements

I joined Audi in 2017 to write my master thesis. Originally, I did not plan to stay longer, but things turned out differently. This was mainly thanks to my later supervisor Uli, who took an early interest in my work and convinced me to stay to pursue a Ph.D. Thank you Uli for so much—for your trust, the freedom to explore the research direction that I considered the most promising, countless hours of discussions and so many good ideas.

I am also very grateful to my university advisor, Professor Jörn Thielecke. I was always impressed by your desire to understand every detail of an idea, which in turn always pushed me to dig deeper to have an answer to all of your questions. A big thank you also goes to Professor Gustav Markkula for co-examining my thesis.

Back to Audi—I encountered a great environment in the AI Lab with many other current and former Ph.D. students who supported and motivated each other. I am grateful for having met so many great people here: Henrik, our collaboration was highly motivating and lead to great results. Johannes and Michael, your dry humor was an important reason to spend not only the day, but also the evening in the office. Franzi, it was great to have you for discussions that went beyond my mathematical understanding! Pia and Sophie, the anecdotes from your user studies could fill a book and were always a welcome excuse to take a break. I loved the atmosphere in our Doktorandenbüro and hope we will revive this from time to time! Also Siedi, thank you for your vision of the AI Lab—this setup was ideal.

A big thanks goes to so many other colleagues from Audi for the great team spirit and for sharing enjoyable moments also outside of work. Specifically, I would like to thank Richard, for taking the initiative with the drone recordings; Stephi, for supporting us and the students; and Stefan, for pulling back the curtain and saying how things really are.

Equally, I am grateful for the colleagues that I met at the LIKE. Adam, Markus, Christian, Flo, Rossouw, Burak, Sebastian, Lukas—thank you for the great collaboration, critical feedback and encouragement! Also, thank you to all the students that I supervised and worked together with. I learned as much from you as you from me.

I am indebted to my friends from school and university who supported me throughout the highs and lows of this time, despite me often being far away and having little spare time. Further, I would like to thank my parents Silke and Michael and my sister Carlotta. You always supported me and your counsel has been pivotal in shaping my path. And finally, thank you to Nicola for always being there for me.

Verden, March 2024 *Moritz Sackmann*

Contents

List of Abbreviations

Adam Adaptive Moment Estimation
AIRL Adversarial Inverse Reinforcement Learning

BC Behavioral Cloning

CA Constant Acceleration
CNN Convolutional Neural Network
CTRA Constant Turn Rate and Acceleration
CTRV Constant Turn Rate and Velocity
CV Constant Velocity
CVAE Conditional Variational Auto-Encoder

DFS DataFromSky
DQN Deep Q-Network

FLOP Floating Point Operation

GAE Generalized Advantage Estimate
GAIL Generative Adversarial Imitation Learning
GAN Generative Adversarial Network
GNN Graph Neural Network
GNSS Global Navigation Satellite System
GP Gaussian Process
GPU Graphics Processing Unit
GRU Gated Recurrent Unit

HMM Hidden Markov Model

IDM Intelligent Driver Model
INS Inertial Navigation System
IPPO Independent Proximal Policy Optimization
IRL Inverse Reinforcement Learning

LiDAR Light Detection and Ranging
LSTM Long Short-Term Memory

MARL Multi-Agent Reinforcement Learning
MOBIL Minimize Overall Braking Induced by Lane Change
MS Multi-Step

PDE Partial Differential Equation
PF Particle Filter
POMDP Partially Observable Markov Decision Process
POSG Partially Observable Stochastic Game
PPO Proximal Policy Optimization

REINFORCE REward Increment = Nonnegative Factor x Offset Reinforcement x Characteristic Eligibility
RL Reinforcement Learning
RMSE Root Mean Square Error
RNN Recurrent Neural Network

SAC Soft Actor Critic
SIMD Single Instruction, Multiple Data
SMAC StarCraft Multi-Agent Challenge
SVM Support Vector Machine

TRPO Trust Region Policy Optimization

UKF Unscented Kalman Filter

VGMM Variational Gaussian Mixture Model

WTA Winner-Takes-All

Symbols and Mathematical Notation

Symbol	Description
	General
$\lvert x \rvert$	Absolute value of x
$\lvert X \rvert$	Cardinality of set X
$\exp(x)$	Alternative notation for e^x
	Random variables
bold letters	Random variables or random vectors, e.g., $\mathbf{y}$
regular letters	Realization of random variable or random vector, e.g., y
$y \sim \mathbf{y}$	y is a realization of the random variable $\mathbf{y}$
$\Omega_{\mathbf{y}}$	Sample space of the random variable $\mathbf{y}$, i.e., $y \in \Omega_{\mathbf{y}}$
$p(y) = p(\mathbf{y} = y)$	Probability or density of random variable $\mathbf{y}$ assuming the value y
$\mathbf{E}\{\mathbf{y}\}$	Expected value of random variable $\mathbf{y}$
$p(y\lvert x)$	Conditional probability of y given x
$\mathbf{E}\{\mathbf{y}\lvert\mathbf{x}\}$	Conditional expected value
$\mathcal{N}(\mu, \sigma^2)$	Normal distribution with mean μ and standard deviation σ
$\mathcal{N}(\mu, \Sigma)$	Normal distribution with mean vector μ and covariance matrix Σ
$\mathcal{U}(a, b)$	Continuous uniform distribution on interval $[a, b]$
	Simulation properties
k	Timestep. Negative values indicate past, 0 is the prediction origin, positive values are in the future.
Δt	Time resolution of the prediction, typically $\Delta t = 0.2\,\mathrm{s}$
H	Prediction horizon, number of predicted steps.
N	Number of vehicles in predicted traffic situation
	Prediction formalism
$a : b$	Inclusive range $a, a+1, \ldots, b-1, b$
$a :$	Inclusive range with undefined end, i.e., $a, a+1, \ldots$
$: b$	Inclusive range with undefined beginning, i.e., $\ldots, b-1, b$
$\mathbf{y}_k^j$	Predicted future state of agent j at timestep k (random variable)
$\mathbf{y}^j = \mathbf{y}_{1:H}^j$	All predicted future states of agent j
$\mathbf{y} = \mathbf{y}_{1:H}^{1:N}$	Predicted future states of all agents at all timesteps

Symbol	Description
$\mathbf{x}_k^j$	Past state of agent j at timestep k
$\mathbf{x}^j = \mathbf{x}_{:0}^j$	Past states of agent j
$\mathbf{x} = \mathbf{x}_{:0}^{1:N}$	Past states of all vehicles in the traffic situation
$Y = \Omega_{\mathbf{y}}$	State space of predicted states, $y_k^j \in Y$ with $y_k^j \sim \mathbf{y}_k^j$
$\mathbf{m}$	Maneuver
$\mathcal{M}$	Map
$p(y^j \vert x, \mathcal{M})$	Probability density of future states of vehicle j, given the past states of all vehicles and the map.
$p(y \vert x, \mathcal{M})$	Probability density of the future states of all vehicles, given their past states and the map.
	Simulation components
$\mathbf{a}, a, A$	Action: Random variable, realization, action space
$\mathbf{o}, o, O$	Observation: Random variable, realization, observation space
$\pi(\mathbf{a} = a \vert \mathbf{o} = o)$	Stochastic policy: Conditional density of action, given observation
$\pi : O \to A, o \mapsto a$	Deterministic policy function, mapping observations to actions
π_θ	Policy neural network with parameters θ
W_k^j	External world state from perspective of vehicle j, subsuming all surrounding vehicle states $y_k^{1:N}$ and the map.
$\kappa : (y, a) \mapsto y$	Kinematic model, mapping the current kinematic state and action to the next state
$\lambda : (y_k^j, W_k^j) \mapsto o_k^j$	Observation model, mapping the current vehicle and world state to an observation
	Neural networks
$\ell(x)$	General loss function
$\nabla_\theta \ell(x)$	Gradient of loss function w.r.t. the neural network parameters θ
$f_2(x) = x^2$	Quadratic loss function
$f_H(x)$	Huber loss function
α	Learning rate for neural network optimization
	Reinforcement Learning
$\mathcal{R} : (y, a) \mapsto r$	Reward function, determining the reward for selecting action a in state y
$\mathcal{R}^j$	Reward function of agent j in a multi-agent environment

Symbol	Description
ω	Weight of reward terms, i.e., $\mathcal{R}(y,a) = \omega_1\mathcal{R}_1(y,a) + \omega_2\mathcal{R}_2(y,a) + \dots$
$r_k = \mathcal{R}(o_k, a_k)$	Reward at time step k
$\ell_{\text{RL}}(\theta)$	Pseudo-loss, used to estimate the RL policy gradient
τ^j	Trajectory $((y_0^j, o_0^j, a_0^j, r_0^j), (y_1^j, o_1^j, a_1^j, r_1^j), \dots)$ of states, observations, actions and rewards of agent j
$\tau \sim \pi_\theta$	Trajectory τ is obtained by executing the stochastic policy π_θ in the simulated environment
$\mathbf{E}_{\tau\sim\pi_\theta}\{f(x)\}$	Expected value of $f(x)$ when executing the stochastic policy π_θ repeatedly. Hereby, x is some value of the generated trajectories, e.g., the observation or reward.
σ_{min}	Lower boundary of policy action noise σ
$\mathcal{D}_{\text{RL}} = \{\tau^1, \tau^2, \dots\}$	Dataset of trajectories collected by executing the policy in the simulated environment
$\tau_{a:b}$	Snippet from trajectory τ from timestep a to b
$R(\tau_{a:b})$	Sum of rewards along trajectory between timestep a and b, possibly discounted by γ. $R(\tau_{a:b}) = \sum_{k=a}^{b} \gamma^{(k-a)} r_k$
$R(\tau) = R(\tau_{0:\infty})$	Return, i.e., sum of all rewards along the trajectory
γ	Discount factor, which decreases the value of future rewards
$\overline{R(\tau)}$	Median return of multiple trajectories, used to measure the performance of a policy
$V_\pi : o \mapsto \mathbb{R}$	Value function: expected cumulated rewards when following policy π after making observation o
$\hat{V}_\psi : o \mapsto \mathbb{R}$	Value network with parameters ψ; estimate of V_π
$\hat{A}_k^{(n)}$	n-step advantage estimate after trajectory step k
Ψ	Gradient weight, e.g., generalized advantage estimate Ψ^{GAE}
$\Psi_k^{\text{GAE}(\lambda,\gamma)}$	Generalized Advantage Estimate gradient weight, see (5.24)
$R_k^{\text{GAE}(\lambda,\gamma)}$	Generalized Advantage Estimate return, see (5.25)
$\text{clip}(x; l, u)$	Function that clips x if it exceeds the lower or upper bound l or u
	Inverse Reinforcement Learning
$\mathcal{D}_E$	Dataset of expert demonstrations
$\mathcal{D}_S$	Dataset of demonstrations from policy execution
$D_\phi : B \to [0,1]$	GAIL Discriminator neural network with parameters ϕ
	Kinematic model
p_x, p_y	Coordinates of vehicle center of gravity
ψ	Vehicle heading angle

Symbol	Description
δ	Vehicle steering angle
v	Speed
$a_{\text{lon}}, a_{\text{lat}}$	Longitudinal and lateral acceleration
β	Slip angle
l_r, l_f	Distance from rear or front axle to center of gravity

1 Introduction

For decades, vehicles have been travelling with little human intervention in the air, in space, on sea, on the moon and on Mars [Web14]. And yet, one of the most common transportation experiences to humans—driving a car—requires continuous control and supervision. In recent years, some companies have been experimenting with “Level 4” automated driving systems that automatically control a road vehicle [Ack21]. However, these systems are still subject of research and development and work only under specific conditions in limited areas [Ing23]. After at least thirty years of research [Pom89; DZ87], what makes the development of fully automated road vehicles so challenging?

A key reason lies in the immense diversity of driving situations: The fundamental task of reliably perceiving a traffic situation is still a subject of research [Van+18], involving the detection of the road layout and the position, heading, velocity and further object characteristics of other traffic participants such as cars, pedestrians, kids, animals, or cyclists. This poses tremendous demands on the perception system, especially under adverse conditions such as bad lighting or partial occlusion.

Then, after perceiving the traffic situation, the automated vehicle needs to interpret it to make a plan for its own future behavior. Consider for example the situation faced by the driver of the red vehicle in Figure 1.1. He might wonder: “Can I enter the roundabout before the white car? Or would this be impolite, because it forces the other vehicle to brake?”. Let us assume that the red vehicle is an automated vehicle. In order to reason about such questions, it needs a model of how its perceived environment will likely evolve in the next seconds. The better the model of the environment, the more likely is it that it can make a plan that it does not need to alter when it perceives the actual evolution of the situation.

Figure 1.1: Exemplary unclear right-of-way situation at a roundabout

What makes such predictions even more challenging is that they sometimes depend on how the automated vehicle itself behaves. For example, if the red vehicle accelerates and enters the roundabout, the white vehicle will likely slow down compared to driving unhindered. If the automated vehicle predicts the behavior of the white vehicle without considering its own planned trajectory, it would likely predict the white vehicle to drive unhindered, and draw the conclusion that it needs to stop at the entry. While the resulting behavior is safe, it might often be too cautious and unnecessarily impede the overall traffic flow. This phenomenon is known as the "freezing robot problem" [TK10] and occurs in highly interactive scenarios such as roundabouts or merging situations with high traffic density.

For this reason, this thesis focuses on approaches to generate interacting and conditional predictions of the trajectories of surrounding vehicles in the next 5 to $10\,\mathrm{s}$. Specifically, what sets the methods proposed in this work apart from many other approaches to trajectory prediction, are the following aspects:

- **Interacting predictions**: The predictions describe a coherent evolution of the traffic situation, where the predicted vehicles interact with each other. For example, the proposed methods can predict that one vehicle enters the roundabout as soon as it observes that a potentially conflicting vehicle is leaving the roundabout and a sufficiently large gap emerges. This is in contrast to many other approaches to trajectory prediction discussed in Section 2.4, which predict vehicle trajectories independently. The latter leads to potentially overlapping predicted trajectories where the predicted vehicles seem to "drive through each other".
- **Conditional predictions**: The predictions can be issued conditional on the future trajectory of one vehicle. This can be leveraged by a cooperative behavior planning system of an automated vehicle: It compares multiple future planned trajectories for itself. Conditioned on each plan, it predicts the evolution of the traffic situation. Ultimately, it selects the planned trajectory that is not only optimal for itself, but also for others.

To implement a prediction that fulfills these two points, it is realized as a microscopic traffic simulation, in which all vehicles from the situation are modeled as independent agents. The states of the vehicles are initialized according to the perceived traffic situation that shall be predicted. Then, each agent follows a behavior model, which instructs it how to act, depending on a simulated observation of its local environment. After processing the actions of all agents with a kinematic model, their states in the next time step are computed. By executing this scheme repeatedly, the trajectory predictions are constructed step by step. Thereby, the predicted vehicles can observe each other and therefore interact with each other. Conditional predictions can be realized by fixating the trajectory of one agent to a specific trajectory prior to executing the simulation of the surrounding vehicles.

This thesis is not the first work to propose such a *prediction-by-simulation* approach. Other notable works that have shaped the ideas in this work have been presented by Schulz et al. [Sch+18b; Sch+19] and a group of researchers from Stanford University [WRK16; Kue+17; Bha+18; Bha+19]. The key question that differentiates these works is the way in which the behavior of vehicles in the simulation is modelled. While [Sch+18b] formulates a set of rules to determine the behavior, all other works follow a data-driven approach and learn the behavior model from a dataset of real-world trajectories.

In the literature [Kue+17; SB18], the behavior model it is often referred to as the *policy*. It is the central component in the simulation, as it decides whether the predictions will be plausible and accurate. Thus, the central topic of this thesis is the investigation of different approaches to obtaining the policy. To do so in a general and scalable manner, this work is exclusively concerned with learning-based approaches, where the policy is implemented as a neural network. Hereby, ideas from the previously mentioned works are picked up and extended, as briefly discussed in the following section and explained in detail in the corresponding chapters of this work.

1.1 Outline

Chapter 2 expands on the fundamentals of driver behavior modeling. It contains an overview of potential applications and the challenges of trajectory prediction. After introducing a general problem formulation, it provides a survey of different approaches to trajectory prediction and motivates the choice of a simulation-based prediction approach.

The remaining content of this thesis can be illustrated with the "behavior triangle" depicted in Figure 1.2. There are three perspectives on the description of behavior:

- Firstly, behavior can be described through the underlying goals, assuming that an actor acts rationally to achieve his goals. The goals can be mathematically expressed as a cost or *reward function*. For example, it can be assumed that drivers want to maximize their progress along the road while driving safely and comfortably.
- Alternatively, behavior can be described as a function of the state and environment of an actor. This is the *policy*, which describes how the actor behaves, depending on his perceived state in the environment. For example, a set of rules or equations could describe how a driver steers his vehicle to remain in the center of the road and to maintain a comfortable distance to a preceding vehicle.
- And thirdly, behavior can be directly described through its concrete realization: This is the *trajectory* of an actor, which emerges as he interacts with the environment through his policy. Only this physical realization of the behavior, e.g., the position, velocity, acceleration and steering over time can be measured. Ultimately, the goal of this work is to obtain a prediction of the trajectory of other vehicles.

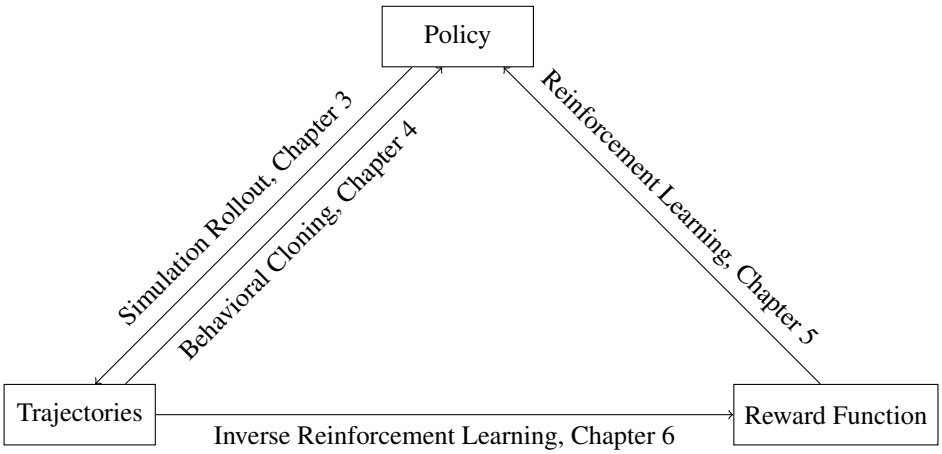


Figure 1.2: The "behavior triangle" sets the three perspectives on behavior discussed in this thesis into perspective: Concrete behavior (trajectories); functions that generate concrete behavior (policies); and goals (rewards) that can be used to derive policies or to explain concrete behavior. Each chapter mentioned in the figure establishes a connection between two of the blocks.

Chapters 3 to 6 can be interpreted as building the links between these three blocks, as indicated in the figure.

Specifically, Chapter 3 introduces the microscopic traffic simulation that executes the behavior policy. By executing the policy of all vehicles in the simulation, a prediction of their interacting future trajectories is created. Hence, this chapter forms the connection from the policy to trajectories in the behavior triangle. The simulation framework implemented in this thesis is used for training and evaluating all presented approaches to learning a policy. Moreover, the chapter describes the dataset of more than 4000 real-world trajectories that is used for training and evaluation in all experiments.

With this background, different approaches to learning a driver behavior policy are presented in Chapters 4 to 6. Chapter 4 introduces Behavioral Cloning (BC), a set of methods that learn a policy by directly imitating the behavior observed in a dataset of trajectories. This establishes the next connection in the behavior triangle—how to obtain a policy, given a set of trajectories. Fundamentally, the task is formulated as a supervised learning problem, with the idea of executing similar actions to the real-world drivers from the trajectory dataset, when confronted with similar situations. One novelty in this thesis is to execute the policy neural network in a differentiable traffic simulation, which allows to directly minimize the long-term trajectory prediction error. This is opposed to minimizing only the error of predicting the directly next acceleration and steering action of the policy, which has been proposed by [WRK16; Sch+19] and multiple other works discussed in this chapter.

Next, an entirely different approach is presented in Chapter 5: Reinforcement Learning (RL). Here, the goal is to find a policy that maximizes a manually defined reward function. In the

behavior triangle, this chapter establishes the link between rewards, which can be interpreted as the goals of a driver, and the policy, that describes how to maximize the rewards and thereby achieve the goals. The employed RL algorithm works by executing actions randomly in the simulation, assessing which actions lead to high cumulative rewards, and increasing the probability of selecting those actions in the next training epoch. The advantage of RL over BC is that the resulting behavior of the policy can be directly influenced by the reward function, e.g., to teach the policy to avoid collisions. Moreover, RL training happens entirely in the simulation environment and does not require any real-world training data, thereby enabling the training in a variety of situations for which no training data exists.

However, to accurately model the behavior of humans with RL, the goals of human drivers must be known and specified as a mathematical reward function. As this reward function is generally unknown and hard to model, Chapter 6 investigates two different approaches to automatically construct a reward function that best explains a set of real-world trajectories. This builds the final link in the behavior triangle, inferring the reward function from a trajectory dataset. With this reconstructed reward function, a RL algorithm can be employed to learn a policy that follows similar goals as human drivers, and hence produces trajectories that resemble trajectories driven by humans. The methods presented in [Kue+17; Bha+18; Bha+19] fall into this category. This thesis extends them with a different algorithm and problem setup that allows for better interpretability of the inferred reward function. Moreover, a training procedure that trains in real-world situations as well as simulated situations is proposed. The idea is to learn a policy that predicts real situations accurately, but is also able to handle situations for which real-world training data is hard to obtain, for example near-collisions.

Finally, a comparison of the different approaches to learning a policy in terms of plausibility, accuracy and ability to generalize to untrained situations is made in Chapter 7. The comparison is carried out under equal conditions, e.g., by training on the same dataset, using the same neural network architecture for the policy, and evaluating on the same dataset. Chapter 8 concludes the results and discusses limitations and opportunities for future research.

1.2 Contributions

Many ideas lined out in this work have been published in [**Sac+20b**; **Sac+21**; **Kon+21**; **Sac+22a**; **Sac+22b**]. Throughout this work, all connected publications are printed in bold letters, with a comprehensive list on page 213f. This thesis aims to set the central publications into perspective, extend the ideas, and to compare the performance of the different methods under equal conditions. The main contributions are:

- **Multi-step training** (Chapter 4, [**Sac+20b**; **Sac+21**]): Other works in BC train a policy by minimizing the prediction error of the next action (acceleration, steering), based on

the current local environment of a vehicle. Policies trained in this fashion are instable, because they are not confronted with the consequences of their actions during training. For example, they tend to drift off the road and are unable to recover from this during long-term predictions. Instead, this work proposes multi-step training, where the policy is trained to directly minimize the multi-step trajectory prediction error.

- **Differentiable simulation environment** (Chapters 3 to 4, [**Sac+20b**; **Sac+21**]): To realize multi-step training, the gradient of the trajectory prediction error must be computed with respect to the policy neural network parameters. This means that not only the policy, but the full simulation environment needs to be implemented in a framework that is capable of automatic differentiation. Hence, no standard simulation environment such as Carla [Dos17] can be used. Moreover, the simulation environment that is implemented as a part of this thesis is capable of simulating more than $2\,000\,\mathrm{s}$ of vehicle trajectories in $1\,\mathrm{s}$ of computation time on a single CPU core. This allows for fast training times of one to a few hours for any method presented in this work.
- **Advances in Multi-Agent Reinforcement Learning (MARL) for traffic situation modeling** (Chapter 5, [**Kon+21**; **Sac+22a**]): The previously mentioned multi-step training approach to learning a policy is limited in that its goal is to imitate the ground truth trajectories. No secondary goals can be formulated, e.g., to teach the policy to avoid collisions under any circumstances. Moreover, multi-step training is restricted to situations for which ground truth trajectories have been recorded. To rectify these deficiencies, a MARL approach to learning a policy is proposed. The training takes place in simulated traffic situations with the goal of maximizing a manually specified reward function. To learn a policy that is capable of handling the interaction between vehicles, e.g., right-of-way situations, the single-agent Proximal Policy Optimization algorithm [Sch+17a] is modified to learn from the experience of all vehicles in the simulated traffic situation. The same policy neural network is used for controlling every agent in the traffic situation, a technique which is known as parameter sharing.
- Extending MARL to model **heterogeneous behavior** (Section 5.5, [**Sac+22a**]): While parameter sharing is a straightforward approach to learning a policy that models the behavior of multiple agents, it implies that all agents behave equally when confronted with the same situations. This is unrealistic when modeling human drivers with different driving styles. To fix this, an approach to learning a flexible policy that can exhibit different behaviors, depending on special preference inputs, is proposed.
- **Reconstructing the reward function** using Adversarial Inverse Reinforcement Learning (AIRL) [FLL18] (Chapter 6, [**Sac+22b**]): The previously mentioned MARL approaches maximize a manually specified reward function. As the reward function of human drivers is unknown, this limits their usefulness for issuing accurate predictions of a real-world traffic situation. To still be able to model real-world driver behavior using the MARL idea, the AIRL [FLL18] algorithm is employed to reconstruct the

reward function from the dataset of real-world trajectories. To this end, the single-agent AIRL algorithm is modified to handle the multi-agent problem formulated in this thesis.
- **Training in fictional situations** (Section 6.4.2): Similar to most real-world datasets, the dataset used in this thesis contains many regular driving situations, but relatively few rare events such as near-collisions. It is shown that additional training in fictional situations, which are set up to include many critical driving situations, leads to more robust policies with reduced collision rates. For this, the AIRL algorithm is modified to reconstruct the reward function on the real-world dataset, but to train the policy in complementary fictional situations.
- **Comparison under equal conditions** (Chapter 7): Finally, an extensive comparison of different variants of all presented algorithms (single- and multi-step BC, Generative Adversarial Imitation Learning (GAIL), AIRL) to learning a driving policy is carried out. This allows for a direct comparison of the accuracy and robustness of the policies as well as the training duration.

1.3 Publications

The central ideas lined out in this work have been published in [**Sac+20b**; **Sac+21**; **Sac+22a**; **Sac+22b**] based on my own contributions as the lead author. Further, [**Kon+21**] was published as the result of my supervision of a master thesis.

As the publications share a common simulation core that has been improved and extended over time, all experiments presented in this thesis have been repeated with the most recent implementation of the simulation. This is required for a fair comparison of the performance of the different approaches under equal conditions in Chapter 7. Hence, most experimental results of this thesis have not been published previously, but are in line with the results in the associated publications.

The relation between the content of this thesis and previous publications is:

- Chapter 2 contains an extensive literature review, previously unpublished.
- Chapter 3 describes the simulation environment used in this thesis, first described in [**Sac+21**] and used and extended in [**Sac+22a**; **Sac+22b**].
- Chapter 4 describes the idea of Behavioral Cloning and multi-step training. It is based on [**Sac+20b**; **Sac+21**] and contains previously unpublished experiments on causal confusion.
- Chapter 5 focuses on Reinforcement Learning. The idea of applying Multi-Agent Reinforcement Learning was published as the result from a master thesis that I supervised [**Kon+21**]. An improved learning algorithm that is used in this thesis and an approach to modelling heterogeneous driver behavior were later published in [**Sac+22a**]. This thesis

further introduces a previously unpublished cost function and multiple experiments to gain insight into the functioning of the RL algorithms.
- Chapter 6 is based on [**Sac+22b**].
- Chapter 7 compares all approaches to behavior modeling presented in this thesis and was not previously published.

Further publications which I coauthored, and which do not play a major role in this thesis, are:

- as the result of my supervision of different master theses [**Vog+20**; **Lee+21**],
- as the result of my co-supervision of a master thesis [**Rad+23**],
- as the result of my co-supervision of my "successor" as a PhD student, also working on the topic of interacting predictions, [**Kon+23**],
- as the result of my collaboration with a colleague, whose research as the lead author focuses on the planning task of automated vehicles, [**Bey+20**; **Bey+21a**; **Bey+21b**].

2 Fundamentals of Driver Behavior Modeling

This chapter serves as a literature review on systems for modeling driver behavior to motivate the architectural design decisions behind the proposed prediction approaches in the following chapters. First, Section 2.1 introduces the central applications of driver behavior models, derives the underlying assumptions of the proposed prediction approaches and formulates requirements. Next, Section 2.2 discusses the core challenges of trajectory prediction. A general problem formulation is stated in Section 2.3, and related works that address this task are outlined in Section 2.4. Thereby, a special emphasis is laid on the representation of the uncertainty in Section 2.5 and on the representation of the environment as the input of the prediction model in Section 2.6.

2.1 Background: Applications

To motivate research on driver behavior modeling, this section introduces the major applications of driver behavior models: Behavior planning for an automated vehicle and behavior simulation for the development and validation of automated driving functions.

2.1.1 Behavior Planning

The structure of an automated driving system can be separated into three consecutive parts shown in Figure 2.1: “Perception - Cognition - Execution” (K.-H. Siedersberger, personal

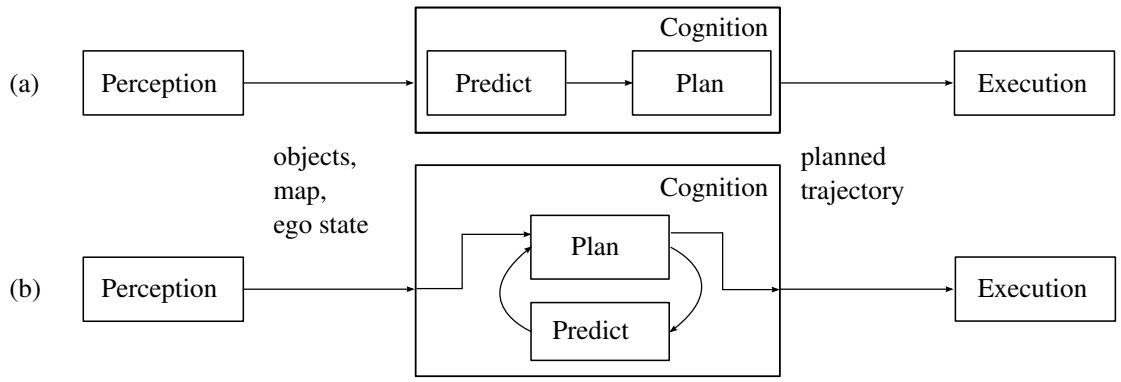


Figure 2.1: Behavior planning according to the Perception-Cognition-Execution scheme.
(a) Sequential prediction and planning block.
(b) Holistic planning and prediction block.

communication, October 2023). A model of the environment that contains all surrounding objects perceived by current and past sensor inputs and the ego state is constructed in the perception stage. This also includes external data, e.g., from digital maps. Based on this information, the cognition system searches for a trajectory that balances certain criteria, e.g., low jerk, low acceleration, progress along the track, as well as staying on track and avoiding collisions. Finally, the execution block is a controller that selects the appropriate actions (e.g., steering and acceleration) to realize the planned trajectory. The cognition stage can be executed in two different ways, depicted in Figure 2.1:

(a) **Sequential** planning: First predicting the evolution of the traffic situation, and then planning a trajectory. This implies that the prediction of surrounding vehicles disregards the planned actions of the ego vehicle, prohibiting maneuvers that require cooperation of other vehicles.
(b) **Holistic** planning: Selecting a candidate trajectory for the ego first, and then predicting how this would influence the other vehicles. This cycle repeats until a sufficiently good trajectory is found. The resulting trajectory is the plan. The task of predicting the situation conditioned on the own plan is referred to as *conditional inference* [Tol+21] or *hypothetical inference* [TS19].

The "freezing robot problem" [TK10] illustrates the functional differences between the approaches: Consider a highly interactive traffic situation, for example, when two lanes merge into one. Being unable to conceive cooperation of surrounding vehicles, an automated vehicle that implements a sequential approach might get stuck at high traffic densities. A holistic approach is able to resolve these situations, as it can consider merging into a gap even if this forces the rear vehicle to brake. As it models the influence of the ego on the rear vehicle, it can plan cooperatively and minimize the discomfort not only for itself, but also for the other vehicle. However, this ability comes at the high cost of having to predict the same situation repeatedly, for each evaluated ego candidate trajectory. Practical implementations of behavior planning algorithms such as [**Bey+20**; **Bey+21a**; **Bey+21b**; Hub+17] often evaluate thousands of candidate trajectories, rendering this an expensive design choice.

As behavior planning is beyond the scope of this work, in the remainder an agnostic position is assumed, requiring the developed behavior models to be compatible with either approach. In the following, the combined prediction and planning module is denoted as the *cognition* module. To be able to implement a holistic cognition module, the prediction model must be capable of hypothetical inference, i.e., being able to consider the influence of the future trajectory of the ego vehicle on the predictions of surrounding vehicles.

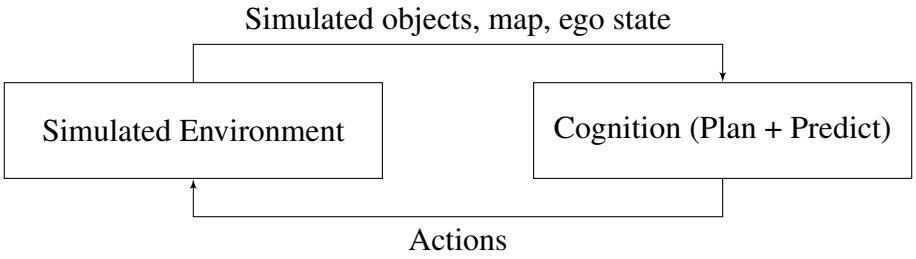


Figure 2.2: Simulating the inputs for the cognition block and reacting to the planned actions

2.1.2 Behavior Simulation

One way to test an automated vehicle is to perform real-world test drives. However, this is expensive, time-consuming, and errors are often not reproducible. Multiple authors argue that billions of test kilometers are required to statistically assure that automated vehicles drive more safely than human drivers [WW16; KP16]—and changes to the system require the testing to start over again.

A realistic simulation offers the potential to substantially reduce the real world testing effort [Win+16, Ch. 8]. Figure 2.2 depicts the sub-problem of simulating the environment of the cognition module. The environment simulator generates the perception-outputs, i.e., the map, ego state and other objects, and determines the simulated ego states according to the actions by a kinematic model.

Today, popular automated driving simulation frameworks such as CARLA [Dos17; 22] and SUMO [DLR; Lop+18] use manually formulated rule-based models to simulate the behavior of surrounding vehicles. In a simulator study, Rock et al. [Roc+22] found that participants rate the plausibility and interactivity of such models poorly. However, the development of highly automated driving functions, especially cooperative behavior planners, requires realistic behavior models of surrounding vehicles to ensure that the performance of the system in the simulation is representative for its performance in the real world.

To this end, behavior models similar to those introduced in this work are used for simulating the behavior of surrounding vehicles to compare the robustness of different behavior planning algorithms to internal prediction model errors [**Bey+21a**]. Another model is used in [**Bey+21b**] as both, the internal prediction model of the behavior planner and the simulation model for the evaluation. Other recent works that target the realistic simulation of driver behavior with learned models are presented in [Suo+21; Ber+21].

2.1.3 Related Tasks

Besides behavior planning and simulation, driver behavior models can be used for multiple other applications, such as:

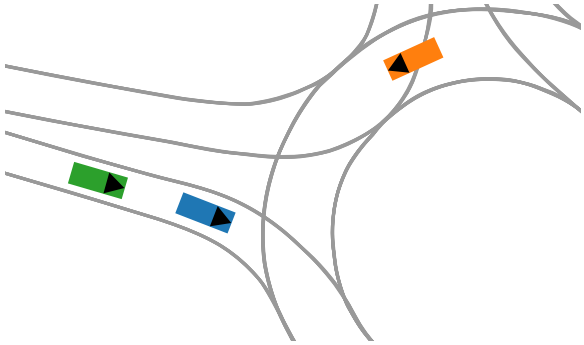


Figure 2.3: Exemplary traffic situation

- Intention Estimation: By predicting actions for multiple future options and comparing them to the actually performed action of the vehicle, the intention of a driver can be estimated. For example, [Sch+18c] combines a behavior model with a multiple model unscented Kalman filter to estimate whether a vehicle is driving straight through an intersection, or turning left or right.
- Risk Assessment [LVL14]: When the behavior model is employed to predict the future trajectories of surrounding vehicles, these trajectories can be used to estimate the criticality of a situation and possibly execute emergency maneuvers or issue a takeover request to the driver.
- Ego behavior prediction for human-controlled vehicles: Significant mismatches between executed and predicted actions can be used to identify edge cases in which the model needs to be improved or where the driver makes a mistake.

2.2 Core Challenges of Trajectory Prediction

Consider the situation depicted in Figure 2.3, consisting of three agents, Blue, Green, and Orange. Many challenges arise when predicting the future evolution of the situation:

1. **Influence of road geometry**: The future movement of all vehicles is typically restricted by the road geometry, which therefore needs to be considered in the prediction.
2. **Kinematic feasibility**: Vehicle trajectories are subject to kinematic constraints [Kon+15]. Making use of this knowledge can improve the prediction performance [JDZ21].
3. **Multimodality**: Which path will Orange take? It is unclear whether it continues driving in the roundabout, or whether it leaves at the next exit.
4. **Traffic rules**: If Orange continues in the roundabout, Blue and Green should respect its right of way.
5. **Interaction**: Nevertheless, Blue could also decide to squeeze into the roundabout before Orange, thereby forcing Orange to brake. Influences between vehicles can sometimes

be considered unidirectional, e.g., Blue influences Green, but not vice versa. However, symmetric relations also exist, e.g., Orange influences Blue, but Blue also influences Orange. This entanglement means that predictions cannot be made in sequence, e.g., first Blue, then Green. Rather, predictions need to be able to influence each other.

6. **Driver characteristics**: Beyond the uncertainty induced by the previous points, different driver characteristics, e.g., aggressiveness or attentiveness [Sad+18], have an influence on the future trajectory.
7. **Vehicle characteristics**: Similar to the previous point, power, mass and shape of vehicles can have an influence on the trajectory.
8. **Combinatorics**: The multimodality due to the road network and different possible sequences of maneuver execution lead to a rapid growth of possible logical traffic evolutions, when multiple vehicles are present in a situation.
9. **Traffic participant class**: Moreover, the type of the traffic participant, e.g., car, truck, bicycle, pedestrian, has a strong influence on the prediction.
10. **Aleatoric uncertainty** [HW21; Var+22]: Even if one driver is perfectly characterized, some uncertainty remains due to the inherent stochasticity of driver behavior, e.g., because of jitter in the reaction time.

Many of the above points are highly dependent on the interaction between vehicles. Thus, this thesis is exclusively concerned with a holistic prediction of traffic situations rather than separate predictions of individual vehicles. This is in contrast to most prediction approaches presented in Section 2.4, but enables interaction-aware driver behavior models that can be used for both, prediction (including hypothetical inference) and simulation of surrounding vehicles. The models proposed in this work address points 1 to 6. Building on the concepts introduced in this work, further extensions are conceivable to cover the remaining points, but they fall outside the scope of this work.

Following most other works on trajectory prediction, this thesis excludes the perception task from the prediction problem. This is advantageous, as it allows for the development of the prediction module separately from the perception module. This enhances the versatility of the approach and makes it compatible with various environment perception strategies. For this reason, the proposed prediction approaches operate on detected objects and not directly on sensor data. Moreover, the proposed models heavily rely on map information, which is assumed to be available.

2.3 General Problem Formulation

Predicting the future evolution of a traffic situation can be formulated as a probabilistic prediction problem, as for example proposed in [TS19; Rhi+19; Cas+20b; BDK20; Suo+21].

To this end, the future traffic situation is modeled as the realization y of a random variable $\mathbf{y}$ and the goal is to obtain the conditional probability density[1]

$$p(y|x, \mathcal{M}), \tag{2.1}$$

or an approximation thereof. To accommodate all approaches that will be discussed in the next sections into a common framework, let $\mathbf{x}$ be a representation of all dynamic information available at the prediction origin, e.g., the current and past states of all traffic participants, and let $\mathcal{M}$ contain all static information, e.g., the map. The random variable

$$\mathbf{y} \doteq \begin{pmatrix} \mathbf{y}_1^1 & \mathbf{y}_1^2 & \cdots & \mathbf{y}_1^N \\ \mathbf{y}_2^1 & \mathbf{y}_2^2 & \cdots & \mathbf{y}_2^N \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{y}_H^1 & \mathbf{y}_H^2 & \cdots & \mathbf{y}_H^N \end{pmatrix} \doteq \mathbf{y}_{1:H}^{1:N} \tag{2.2}$$

characterizes the positions of all N vehicles, at all H future timesteps. $\mathbf{y}_k^j$ is the random variable of the position or the full state of the vehicle j at timestep k.

To simplify the notation, the absence of a superscript or subscript denotes all agents or timesteps, i.e., $\mathbf{y}_k = \mathbf{y}_k^{1:N}$ and $\mathbf{y}^j = \mathbf{y}_{1:H}^j$. Similarly, $\mathbf{x}_{-P:0}$ describes the states of all traffic participants from the past timestep $k = -P$ to the prediction origin at the current timestep $k = 0$. $\mathbf{x}_k = \mathbf{x}_k^{1:N}$ is the joint random variable of all agent states at timestep k.

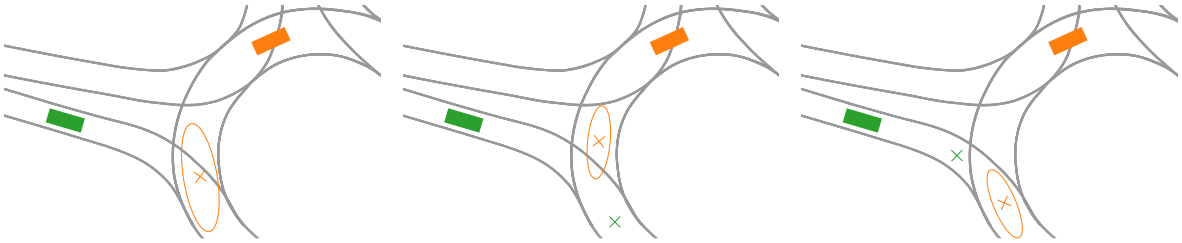
Clearly, the density (2.1) cannot be expressed in closed form as its sample space $\Omega_{\mathbf{y}} = \mathbb{R}^{N \times H}$ is high dimensional with many interdependencies between the components that arise as an effect of the challenges outlined in the previous section—kinematic constraints, map constraints, and interaction between vehicles. Thus, practical implementations of prediction approaches simplify the problem by exploiting its structure in different ways, as described in the following section.

An additional challenge arises when the situation prediction is conditioned on the fixed future of one or more vehicles, typically the automated vehicle a. This problem of hypothetical or conditional inference [TS19; Tol+21] can be formulated as

$$p(y^{1:N \setminus a}|x, \mathcal{M}, \mathbf{y}^a = y^a), \tag{2.3}$$

where $1:N \setminus a$ denotes the set of all vehicles except for vehicle a. Such conditional predictions are essential for the holistic behavior prediction and planning approach from Section 2.1.1. The concept is illustrated in Figure 2.4.

[1] This work follows the convention to abbreviate $p(\mathbf{y} = y|\mathbf{x} = x)$ as $p(y|x)$. Occasionally, the long form is used to highlight the importance of certain variables.



(a) Unconditional prediction **(b)** Green's plan: enter roundabout **(c)** Green's plan: yield to Orange

Figure 2.4: Illustration of hypothetical inference: The green vehicle is the ego. **(a)** Green's plan is unknown at prediction time, leading to large uncertainty about the future position of Orange as indicated by the confidence ellipse. If it is known that green either enters the roundabout **(b)** or stops at the entrance **(c)**, this reduces the uncertainty in the prediction of Orange.

2.4 Approaches to Trajectory Prediction

Numerous approaches to trajectory prediction exists. The central differentiator between them is the structure that is imposed upon the prediction problem (2.1). In line with Lefèvre et al. [LVL14], this section mainly differentiates between physics-based approaches, maneuver-based approaches and interaction-aware approaches.

Physics-based approaches assume that the future vehicle motion exclusively depends on its past motion. In contrast, maneuver-based approaches focus on recognizing executed maneuvers and predict their continuation, sometimes assisted by map knowledge. Finally, interaction-aware approaches attempt to model the interaction between vehicles in the prediction.

This thesis is mainly concerned with interaction-aware trajectory prediction via a rollout of a driver behavior model, as described in Section 2.4.3.2. However, to shed light on the large corpus of works concerned with trajectory prediction, the central ideas of other approaches are discussed in the following. For other perspectives on the field, the reader is referred to the recent surveys on motion prediction by Brown et al. [BDK20], Karle et al. [Kar+22], Mozaffari et al. [Moz+22] and Huang et al. [Hua+22] and the literature review in the dissertations by Schulz [Sch21] and Wissing [Wis20].

Domains Beyond physics-based models, most approaches to trajectory prediction are domain specific. The two key domains, highway and urban settings, can further be divided:

Car-following has been studied at least since 1950 [TK13b, Ch. 10] with the goal of understanding traffic phenomena, such as the formation of traffic jams [THH00]. As these approaches only consider the interaction between the target vehicle and its preceding vehicle in a single-lane scenario, they are of limited use for the development of highly automated vehicles. However, they can be adapted to highway scenarios by comparing the feasible

acceleration on each lane, and switching lanes to minimize the overall braking induced by lane changes (MOBIL) [KTH07]. Many recent approaches directly predict driver behavior on a multi-lane highway, e.g. [Kue+17; Len+17; Die+19]. Apart from these general models, some works focus on specific scenarios, such as merging [YL13; Bou+20] or predicting imminent lane changes [Sch+15a; Woo+17; MKA20; DBU21].

The urban domain requires higher modeling effort due to complex road geometries, additional traffic rules and different types of traffic participants. Many works in this domain focus on particular settings, such as intersections [Lie+12; SH14; Li+22] and roundabouts [Zha+17; ZWN20]. These two scenarios are especially interesting, because they involve a high degree of interaction between drivers. A wide range of models focuses on pedestrian prediction [Rid+18], which is not separately addressed in the following review for brevity. Examples of general urban prediction models are [CLU18; Sch+18b; Gao+20].

2.4.1 Physics-Based Models

Simple and popular physics-based models are the Constant Velocity (CV) and the Constant Acceleration (CA) model, as well as the Constant Turn Rate and Velocity (CTRV) and Constant Turn Rate and Acceleration (CTRA) model. They are often employed for short term predictions, e.g., in tracking problems [SRW08]. As these models only consider the current kinematic state of the vehicle, but neglect any environmental information, e.g., road boundaries, their performance rapidly degrades for long-term predictions [Dju+20]. Most physics-based models address none of the challenges outlined in Section 2.2, except for the kinematic feasibility.

Formally, these models factorize the prediction problem (2.1) as

$$p(y|x,\mathcal{M}) = \prod_{j=1}^{N} p(y^j|x^j) \tag{2.4}$$

by assuming exclusive dependence of the future of each vehicle y^j on its past x^j, and ignoring any interaction or map information. The prediction of each individual vehicle

$$p(y^j|x^j) = p(y_0^j|x^j) \cdot \prod_{k=1}^{H} p(y_k^j|y_{k-1}^j) \tag{2.5}$$

is typically further simplified by modelling y_k^j as the full target vehicle state, and recurrently executing a physics-based, Markovian state-transition model $p(y_k^j|y_{k-1}^j)$. The structure is depicted in Figure 2.5. It is assumed that the initial state y_0^j is contained in x_0^j. The formulation (2.5) naturally lends itself to uncertainty propagation similar to a Kalman filter prediction step, used by Ammoun et al. [AN09], who predict Gaussian uncertainty ellipses for each trajectory

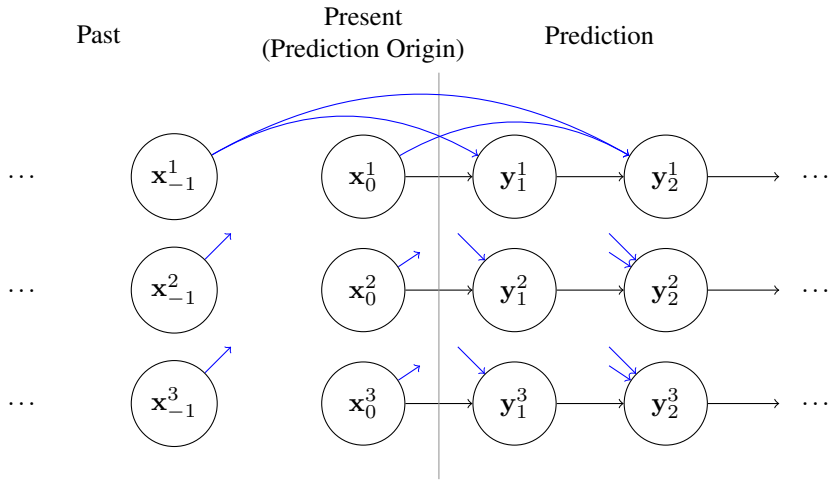


Figure 2.5: Structure of physics-based prediction approaches: The next state of each vehicle is predicted on the basis of all preceding states. The states of different vehicles do not influence each other. Commonly, the Markov assumption is made, which means that each state exclusively depends on its preceding state. In this case, the connections indicated by the blue arrows are omitted. For clarity, only the start and end of the arrows are shown for vehicle 2 and 3.

point. Uncertainty propagation in nonlinear systems can be performed using the unscented transform, which is implicitly linearizes the system with a deterministic set of points around the state estimate. This is commonly used inside the prediction step of the Unscented Kalman Filter (UKF), e.g., by Tran et al. [TF13]. Alternatively, uncertainty propagation in highly nonlinear systems with non-Gaussian random densities can be simulated via Monte Carlo methods, by predicting the evolution of a set of stochastically sampled particles, for example used by Althoff et al. [AM11].

Some approaches also propose to learn a prediction model using the independent prediction problem formulation (2.4). This enables the use of more complex, non-Markovian models. For example, Wiest et al. [Wie+12] describe the past speed and heading angle as a Chebyshev polynomial, and learn the parameters of a Variational Gaussian Mixture Model (VGMM) that predicts the polynomial coefficients of the future trajectory, given the past trajectory coefficients, thereby sidestepping the Markovian assumption (2.5) and allowing the model to leverage long-term trends in the trajectory for the prediction. Another example is Zyner et al. [ZWN20], who predict a Gaussian mixture density of future positions, given a sequence of past states (position, speed and heading). By using a Recurrent Neural Network (RNN), which aggregates information in a hidden state, the Markov assumption is bypassed. As the training data is limited to few similar sized intersections, the model implicitly learns the road layout, thereby allowing multiple distinct path modes to emerge in the prediction, e.g., turning right or going straight at the intersection. This implicit representation of a map is made explicit by the approaches in the next category, maneuver-based models.

2.4.2 Maneuver-Based Models

The idea of maneuver-based models is to determine the probability of different maneuvers m^j, i.e.,

$$p(m^j|x^j, \mathcal{M}). \tag{2.6}$$

Typically, the set of maneuvers $\Omega_{\mathbf{m}^j}$ is restricted to few discrete elements, for example, going straight through an intersection, turning left, or turning right [TF13; TF14; Rei22]. In urban scenarios, the map $\mathcal{M}$ usually contains important information for the classification. As a consequence, each vehicle often has an individual set of possible maneuvers $\Omega_{\mathbf{m}^j}$ as opposed to a universal set of maneuvers $\Omega_{\mathbf{m}}$ that is valid for all vehicles.

A common method to realize the maneuver classification (2.6) is the use of Hidden Markov Models (HMMs) [SH14; DRT18]. For this, a maneuver conditional generative model $p(x^j|m^j, \mathcal{M})$ is constructed using the Baum-Welch algorithm. The learned model allows for a direct estimation of the likelihood of an observed trajectory belonging to each maneuver class. Alternatively, the classification can be performed using a discriminative Long Short-Term Memory (LSTM) neural network trained to directly predict the probability of a maneuver, e.g., [Zyn+17; **Vog+20**]. While both, HMM and LSTM enable updating of the estimated maneuver probabilities as new observations arrive, [Zyn+17] and [**Vog+20**] alternatively experiment with single-shot classification based on a fixed number of historical observations using standard feedforward neural networks. In a similar spirit, Zhao et al. [Zha+17] use a Support Vector Machine (SVM) to predict the probability of a driver leaving a roundabout at the next exit.

Optionally, the future trajectory of a vehicle can be predicted according to a maneuver-conditional prediction model

$$p(y^j|x^j, m^j, \mathcal{M}). \tag{2.7}$$

The prediction scheme follows the independent modelling of all vehicles as in (2.4). Moreover, these approaches naturally handle the multimodality induced by the road network via the maneuver class $\mathbf{m}^j$. The resulting full prediction

$$p(y^j|x^j, \mathcal{M}) = \sum_{m \in \Omega_{\mathbf{m}^j}} p(y^j|x^j, \mathbf{m}^j = m, \mathcal{M}) p(\mathbf{m}^j = m|x^j, \mathcal{M}) \tag{2.8}$$

is obtained by marginalizing out the maneuvers, which leads to a mixture of trajectories of likely maneuvers, e.g., [Wie+13; Xie+18]. Alternatively, (2.7) can be used to simply predict the trajectory of the most likely maneuver, e.g., [TF13; DRT18].

To realize the conditional maneuver prediction (2.7), [DRT18] extend the idea of [Wie+12], and use a VGMM to predict the coefficients of a Chebyshev polynomial that represents the future trajectory, given the past coefficents. To model maneuver dependency, one VGMM is

learned per maneuver. A similar idea is proposed in [Rei22], where $\mathbf{m}$ represents uncertain target positions that are heuristically generated from a map. The maneuver-conditional trajectory prediction (2.7) is implemented using Bézier curves that smoothly connect the current vehicle state to its target positions.

Instead of separately detecting the maneuver and predicting the conditional future trajectory, the problem can also be addressed jointly. For this, [Xie+18] implement an Interacting Multiple Model Kalman filter, which combines maneuver-specific prediction models with a Markov chain that models the transition probabilities between different maneuvers. Similarly, [TF13] learn the parameters of one Gaussian Process (GP) per maneuver, which is then converted to a stepwise, maneuver-conditional transition model that can be employed in an UKF. At execution time, the likelihood of the observation stemming from each maneuver specific GP is evaluated, and the trajectory is predicted according to the most likely maneuver GP. In a later work, [TF14] instead use a particle filter for the trajectory prediction, which represents the maneuver-induced multimodality via a set of particles.

Rehder et al. [RK15; Reh+18] pick up the idea of marginalization (2.8) by representing the uncertainty about the goal that a pedestrian wants to achieve in the conditional variable $\mathbf{m}^j$. In [RK15], the authors describe goals in an unstructured map as a Gaussian mixture and update the estimated likelihood of each component with a particle filter, as new observations become available. In [Reh+18], the goal is predicted from a sequence of camera images, using a combination of a Convolutional Neural Network (CNN) and a LSTM network. Interestingly, both works proceed to use a probabilistic planning algorithm approach towards the estimated goals to implement the goal-conditional trajectory prediction (2.7). In the context of pedestrian prediction, another goal-directed approach to trajectory prediction has been explored by Particke et al. [Par+18b], who construct multiple potential fields towards potential goal points and use a multi-hypotheses Kalman filter to estimate the intention of pedestrians and predict their future trajectories.

A deep learning approach to the maneuver-based vehicle trajectory prediction is proposed by Zyner et al. [ZWN20], who train a network to directly predict the multimodal distribution $p(y^j|x^j)$. The network outputs are the parameters of a Gaussian mixture density that models the future positions of the target vehicle. Reversing the logic of most other works in this section, the authors consolidate the resulting density in a second step to predict the underlying maneuver. For this task, a clustering algorithm that estimates $p(m^j|y^j)$ is employed.

Separately predicting individual driver maneuvers does not allow for a coherent long-term prediction of a situation, as the interaction between vehicles is disregarded. In a highway setting, Deo et al. [DRT18] thus implement a maneuver-based prediction of multiple trajectories for each individual vehicle, but filter out unlikely combinations of predicted trajectories of different vehicles prior to outputting the predictions. The likeliness of a pair of trajectory

predictions is determined inversely proportional to the minimum distance between the trajectories to reflect that drivers are collision averse. This notion of implausible combinations of predicted trajectories is a primitive form of interaction between predictions and leads to the category of interaction-aware models.

2.4.3 Interaction-Aware Models

All interaction-aware models share one central property: Apart from the past states x^j of the target vehicle, they make use of additional information concerning surrounding vehicles contained in x. Typically, map information $\mathcal{M}$ is also factored in. Thus, these approaches have access to all information that is required to make a holistic situation prediction rather than individual vehicle predictions.

Since the proposal of the category of interaction-aware approaches by [LVL14] in 2014, many new works have been published. This encourages a further differentiation between passive, reactive and proactive interaction awareness in this subsection.

Passive approaches use the information contained in x and $\mathcal{M}$ to make predictions for each vehicle individually, neglecting potential future interaction between the predicted trajectories. This thesis follows the terminology of Tolstaya et al. [Tol+21], who label these approaches as passive behavior prediction. To understand the implications, consider again the situation in Figure 2.3. The dynamic state of Orange indicates that Blue should wait at the entry until Orange has passed. However, it is currently unclear when this will be: Orange could either leave the roundabout, enabling Blue to enter right away. Orange could also remain in the roundabout, thereby forcing Blue to wait until it has passed. The future trajectory of Blue is clearly highly dependent on Orange; neglecting this influence forces a diligent predictor to specify a large uncertainty about Blue's future position. This illustrates that passive approaches are incapable of issuing coherent long-term predictions of the situation, because the prediction of one vehicle does not react to the prediction of another vehicle.

Reactive approaches enable interacting predictions by formulating a single-step prediction model that is executed repeatedly with small timesteps, e.g., $\Delta t = 0.2\,\mathrm{s}$. Starting from the prediction origin, $k = 0$, the information from the previous prediction step is an input to the prediction of the next timestep. Thus, predictions of individual vehicles are able to react to each other, which explains the name of this category. In the above example, Blue would enter the roundabout as soon as it perceives that Orange leaves it, or when Orange has driven past Blue. Formulating the single-step prediction model is a challenging task and lies at the heart of this thesis. It can be interpreted as a reactive control policy that is applied by every agent in a simulation of the situation to produce the predicted, interacting trajectories. Hence, these approaches are also referred to as prediction by forward simulation [BDK20; Sch21].

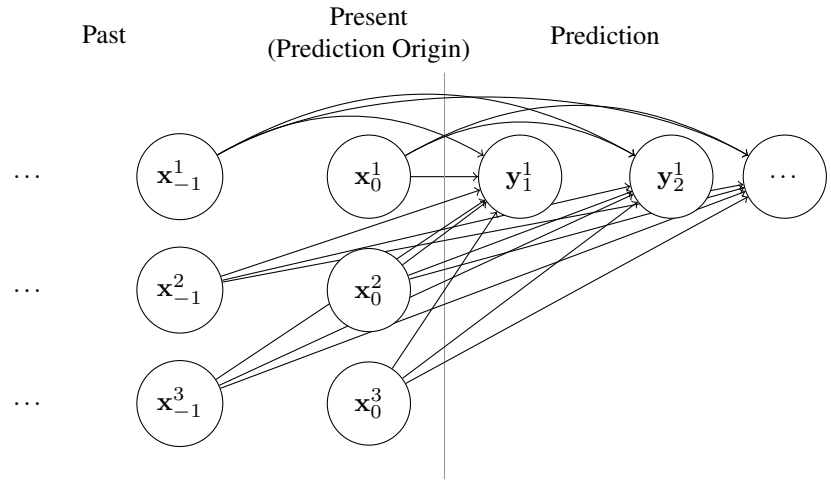


Figure 2.6: Structure of passive interaction-aware approaches: The entire trajectory of one vehicle is predicted based on the past states of itself and of all other vehicles. The number of past states leveraged for the prediction varies between approaches. The predicted trajectory neither influences predictions of other vehicles nor is it influenced by predictions of others. For the sake of clarity, only the predictions of vehicle 1 are shown, even though analogous dependencies exist for all other vehicles.

Finally, proactive approaches go even further by using a proactive policy for every agent. These approaches interpret the prediction problem as a Partially Observable Stochastic Game (POSG) [BDK20], or a joint optimization problem. The solution (prediction) is the set of trajectories that minimizes a cost function. These approaches can also be interpreted as prediction by joint planning, because they employ a planning algorithm for every agent to produce the trajectory prediction. Returning to the example, Blue might evaluate its options and come to the conclusion that strongly accelerating and squeezing into the roundabout before Orange possibly arrives might be its best option in this situation. Thereby, Blue expects that Orange brakes slightly and assumes that this is safe and tolerable for Orange.

As most recent publications on trajectory prediction fall into one of these three subcategories of interaction-aware behavior prediction, the following subsections introduce representative works that fall into these categories.

2.4.3.1 Passive Interaction-Awareness

The simplest form of modelling interaction when predicting the future situation $\mathbf{y}$ is again to factorize the problem into individual predictions of each agent by making the conditional independence assumption

$$p(y|x, \mathcal{M}) = \prod_{j=1}^{N} p(y^j|x, \mathcal{M}), \tag{2.9}$$

which is visualized in Figure 2.6. The central difference to truly independent predictions

according to (2.4) is the dependence of each individual prediction on the dynamic states of all vehicles x instead of the individual dynamic state x^j of the target vehicle.[2] The prediction $p(y^j|x, \mathcal{M})$ of the future trajectory of each target vehicle y^j is thus carried out independently. Due to this fact, these approaches lack the ability to make conditional predictions according to (2.3), and are thus only suitable for sequential prediction-and-planning architectures (Section 2.1.1). They cannot be used for holistic planning or behavior simulation.

As the direct formulation of the prediction model is difficult, all surveyed approaches following this scheme resort to deep learning methods. Despite the shortcomings of the conditional independence assumption, [Ett+21] and [Tol+21] subsume that many recent works follow this passive behavior prediction approach, as the metrics defined by the popular behavior prediction benchmarks by Lyft [Hou+20], nuScenes [Cae+20] and Argoverse [Wil+21] exclusively evaluate predictions of individual agents. Notable exceptions from this are the Interaction [Zha+19] and Waymo [Ett+21] datasets.

A prime example of a passive prediction approach is proposed by Djuric et al. [Dju+20], who use a gridmap-like birds-eye-view rendering of the surroundings of each target vehicle to encode the dynamic state $\mathbf{x}$ and the map $\mathcal{M}$. The information is then decoded by a CNN, and fed into a LSTM network, which directly predicts the average expected future trajectory, i.e., $\mathbf{E}\left\{\mathbf{y}^j|x, \mathcal{M}\right\}$. Optionally, the authors propose to predict the uncertainty as standard deviations around each future trajectory point.

Another well-known representative of this approach is VectorNet [Gao+20], which proposes a new graph-based input representation that is detailed in Section 2.6. Based on the map and past surrounding vehicle trajectories, the most likely future trajectory of one target vehicle is predicted, including its uncertainty.

Handling Multimodality Cui et al. [Cui+19] extend the work of [Dju+20] to predict multiple trajectory hypotheses per vehicle, to represent the multimodal nature of the problem, e.g., when it is unclear whether a vehicle will turn right or left at an intersection. Given the situation representation $\mathbf{x}$, the model is set up to directly output three future trajectories for each target vehicle. Training is performed in a Winner-Takes-All (WTA) [GBK12; Mak+19] fashion, i.e., only the distance between the ground truth and the closest predicted trajectory is minimized. This ensures that the resulting predictions are diverse, and cover multiple disparate options rather than collapsing into the single most likely trajectory prediction [**Lee+21**; **Hof+22**].

WTA training and other approaches that directly generate a set of representative samples suffer from the problem of *mode collapse*: The predictions cover some modes of the true

[2] This does not necessarily mean that the prediction of each vehicle uses the full dynamic state x of all vehicles in the situation. Often, the prediction only depends on the most recent state estimates of a limited number of surrounding vehicles. For simplicity and generality, this is not explicitly articulated in the following.

distribution, but ignore others. Elaborated training schemes to avoid this issue are proposed in [AB17; Sri+17; Cha+19; Mak+19]. Alternatively, the multimodality can be handled outside the model by introducing a new conditional random variable $\mathbf{m}^j$, that can assume any value from its sample space $\Omega_{\mathbf{m}^j}$, and splitting the prediction problem into

$$p(y^j|x,\mathcal{M}) = \sum_{m\in\Omega_{\mathbf{m}^j}} p(y^j|x,\mathcal{M},\mathbf{m}^j=m)p(\mathbf{m}^j=m|x,\mathcal{M}). \tag{2.10}$$

The resemblance to the maneuver-based formulation (2.6) to (2.8) is no coincidence, as $\mathbf{m}^j$ can encode a maneuver. One example of a combination of a passive interaction-aware and maneuver-based prediction is proposed by Deo et al. [DT18] in a highway setting, where a LSTM network encodes the states of all vehicles, and a pooling layer aggregates information of the relevant vehicles around each target vehicle. Based on this encoding, one neural network estimates the maneuver probabilities $p(m^j|x,\mathcal{M})$ for six maneuver classes, e.g., "brake and change to the right lane". A second network predicts a maneuver-conditional future trajectory $p(y^j|x,\mathcal{M},m^j)$. Marginalization according to (2.10) yields the multimodal trajectory prediction for each target vehicle.

Zhao et al. [Zha+21] apply the idea of Rehder et al. [RK15] of goal-directed prediction to vehicle trajectory prediction. The authors propose to generate multiple potential goal states that are spread over a large area of the map for each target vehicle in the first step. Then, trajectories are predicted for each goal state, and finally ranked by their likelihood. Further examples of the separation of goal or anchor prediction and goal-conditional trajectory prediction are proposed in [Cha+19; Zha+20; Gil+21].

The problem of mode collapse can also be addressed with a generic, uninterpretable latent variable $\mathbf{m}^j$ using a Conditional Variational Auto-Encoder (CVAE) [KW14], as proposed by Hong et al. [HSP19]. In their work, the authors train a CVAE, which generates trajectories via the concatenation of an encoder and a decoder neural network. A trajectory prediction is generated by encoding the scene context into a latent space, sampling one realization of a low-dimensional random variable $\mathbf{m}^j$ and then executing the decoder with the encoded scene context and the sample of $\mathbf{m}^j$ as inputs. A diverse set of predictions is generated by repeating the process for different realizations of $\mathbf{m}^j$.

Modeling Interaction By imposing the conditional independence assumption (2.10), all approaches in the previous paragraphs predict the trajectory of each target vehicle independently. Most presented approaches use a single-shot prediction architecture [Tol+21], i.e., the model directly outputs the full trajectory of an agent, or the parameters of a distribution that characterizes it. The limitations of passive behavior prediction have been discussed in the introduction of Section 2.4.3. The approaches presented in the following adapt the passive single-shot architecture to model interaction between predictions.

Kim et al. [KKC20] propose to perform the prediction repeatedly. The first iteration predicts the trajectory of each target vehicle independently. Successive iterations use the prediction of previous iterations as additional input, thereby enabling interaction between the predicted trajectories. A similar idea is proposed by Lee et al. [Lee+17], where multiple individual trajectory predictions per target vehicle are generated using a CVAE first. In a second step, each prediction is ranked according to its likelihood given the prediction of all other vehicles. Moreover, the model also iteratively refines each individual trajectory prediction, given the context of the other predictions.

Another line of work is presented by Tolstaya et al. [Tol+21]. The central motivation is to make predictions of surrounding vehicles that are conditioned on the planned trajectory of the automated vehicle, as formally defined in (2.3). The prediction is realized by extending the single-shot model from a previous work [Cha+19] with the planned future trajectory of the automated vehicle as an extra input. Thus, all predictions are able to interact with the automated vehicle, but do not interact with each other. The resulting prediction model can be used in a holistic planning approach.

2.4.3.2 Reactive Interaction-Awareness

The idea of iteratively refining the predictions, as proposed by [Lee+17; KKC20], is a first step towards modeling the interaction between vehicles at prediction time. However, many interaction-aware approaches structure the problem differently. The central idea is that the trajectory of each agent emerges as a result of its behavior. Therefore, these approaches establish a single step behavior model that predicts the next action of an agent, given its observation of the current situation. The prediction of a situation is obtained by recurrently simulating the observations of each agent, the ensuing actions according to the behavior model, and the state transitions as a consequence of the actions. This scheme enables interaction between predicted trajectories, because agents are represented in each other's observations. Examples of this approach can be found in car following [THH00; KTH09], highway [KTH07; WRK16; MWK17; Len+17], and urban scenarios [Sch+18b; Sch+19].

Formally, the task of predicting the future situation is thereby factorized into separate time steps

$$p(y|x, \mathcal{M}) = \prod_{k=0}^{H} p(y_{k+1}|x, y_{1:k}, \mathcal{M}), \tag{2.11}$$

with the single-step prediction

$$p(y_{k+1}|x, y_{1:k}, \mathcal{M}) = \prod_{j=1}^{N} p(y_{k+1}^{j}|x, y_{1:k}, \mathcal{M}) \tag{2.12}$$

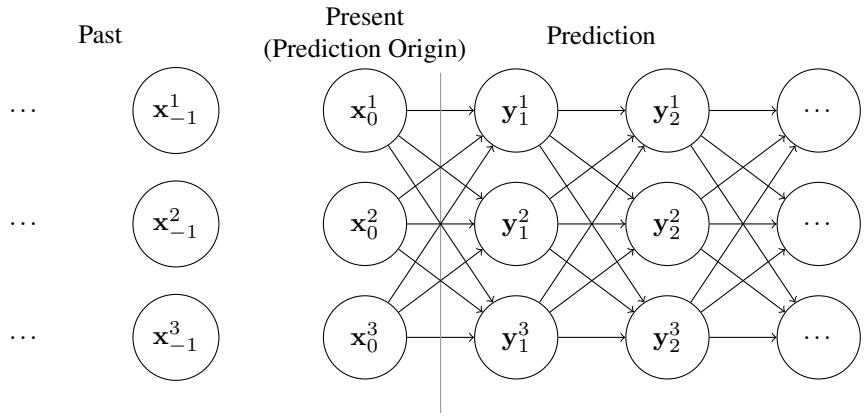


Figure 2.7: Structure of reactive interaction-aware approaches: The predicted next state of each vehicle depends on the previous state of itself and of all other vehicles. As the prediction unfolds step by step, predicted vehicles can react to the prediction of their surrounding vehicles, thereby enabling true interaction between them. Conditional inference can be performed by fixating the trajectory of one agent to the queried trajectory, and executing the prediction for all other vehicles. Through this mechanism, the behavior planner of an automated vehicle can assess how the situation would evolve if it decides for one trajectory. Optionally, the predictions can be based on more than one past state, which is omitted in the figure.

being carried out independently for each agent j. A further common theme is the assumption of a Markovian prediction model [Len+17]

$$p(y^j_{k+1}|x, y_{1:k}, \mathcal{M}) = p(y^j_{k+1}|y_k, \mathcal{M}), \tag{2.13}$$

i.e., each future state $\mathbf{y}^j_{k+1}$ depends exclusively on the directly preceding states $\mathbf{y}_k$ of all vehicles. To simplify the notation, it is assumed that for the initial prediction step $p(y_1|y_0, \mathcal{M})$, all relevant information from x is encoded in y_0. The emerging structure is illustrated in Figure 2.7.

Formulating a prediction model now amounts to describing the single-step transition density $p(y^j_{k+1}|y_k, \mathcal{M})$ for the jth target vehicle. The joint predictions of each agent give the next situation state (2.12), and recurrent application of the situational single-step prediction (2.12) yields the full prediction of the future situation (2.11). Often, $p(y^j_{k+1}|y_k, \mathcal{M})$ is further decomposed into observation model, action model, and state transition model, i.e.,

$$p(y^j_{k+1}|y_k, \mathcal{M}) = \int_{a\in A}\int_{o\in O} p(y^j_{k+1}|y^j_k, \mathbf{a}^j_k = a)p(\mathbf{a}^j_k = a|\mathbf{o}^j_k = o)p(\mathbf{o}^j_k = o|y_k, \mathcal{M})\, do\, da \tag{2.14}$$

with

- $\mathbf{a}^j_k$ — action of agent j at timestep k (random variable),
- $\mathbf{o}^j_k$ — observation of agent j at timestep k (rand. var.),
- $p(a^j_k|o^j_k)$ — behavior model,

- $p(o_k^j|y_k, \mathcal{M})$ observation model of agent j,
- $p(y_{k+1}^j|y_k^j, a_k^j)$ (kinematic) state transition model,
- A action space; sample space of $\mathbf{a}_k^j$,
- O observation space; sample space of $\mathbf{o}_k^j$.

For a coherent notation with Reinforcement Learning (RL) literature [SB18], the behavior model is henceforth denoted as the policy

$$\pi(\mathbf{a}_k^j = a_k^j|\mathbf{o}_k^j = o_k^j) \doteq p(\mathbf{a}_k^j = a_k^j|\mathbf{o}_k^j = o_k^j).$$

Due to the conditional dependency between the components of $\mathbf{y}$ induced by (2.11) to (2.14), no general closed form description of an individual trajectory $\mathbf{y}^j$ can be given. To generate predictions, practical implementations of this approach instead resort to a forward simulation scheme, described in Algorithm 1. Thereby, the prediction can be made in a deterministic manner by propagating only the mean state $\mathbf{E}\{\mathbf{y}_k\}$. Alternatively, it is possible to perform a Monte Carlo simulation by using a set of particles that represent the uncertainty in the initial situation state $\mathbf{y}_0$, and that further reflect process uncertainty by *sampling* from the observation, behavior, and kinematic state transition model.

Algorithm 1 Prediction by simulation, adapted from [BDK20]

1: **for** timestep $k = 0..H$ **do**
2: **for** agent $j = 1..N$ **do**
$o_k^j \leftarrow p(\mathbf{o}_k^j|\mathbf{y}_k = y_k, \mathcal{M})$ ▷ Sample observation o according to observation model
$a_k^j \leftarrow \pi(\mathbf{a}_k^j|\mathbf{o}_k^j = o_k^j)$ ▷ Sample action a from policy π
$y_{k+1}^j \leftarrow p(\mathbf{y}_{k+1}^j|\mathbf{y}_k^j = y_k, \mathbf{a}_k^j = a_k^j)$ ▷ Apply state transition model to determine next agent state y_{k+1}^j.
3: **end for**
4: **end for**

The core challenge of reactive interaction-aware approaches and the central topic of this thesis is the formulation of a behavior policy π. The following paragraphs discuss different approaches to obtaining the behavior model.

Manual Formulation Early works on behavior modeling were originally targeted at microscopic traffic simulations to study the formation of traffic jams, e.g., [THH00]. Thereby, the scheme from Algorithm 1 is applied with a manually formulated, deterministic behavior model in a car following scenario. The proposed Intelligent Driver Model (IDM) observes the current speed of the target vehicle and its preceding vehicle, as well as the bumper-to-bumper distance between both. Based on this information, it predicts the acceleration of the target vehicle, such that it strives to maintain a velocity-dependent safety distance, or accelerates until a desired speed is reached. The model and its extensions [THH00; KTH07; KTH09]

were later adapted for prediction purposes in the behavior planning of automated vehicles, for example, [Eve+16; LKK16; Bey+19]. Further popular examples of manually specified car-following behavior models for microscopic traffic simulation are the Gipps model [Gip80], and the Krauß model [KWG97].

Formulating models for urban traffic situations is clearly more demanding: Schulz et al. [Sch+18b] establish a set of rules to describe the influence of the road geometry, preceding vehicles, conflicting vehicles, speed limits and vehicle dynamics on the behavior. Moreover, similar to (2.7), multimodality is addressed by introducing a hidden route $\mathbf{r}$ and maneuver $\mathbf{m}$ variable for each agent, and formulating a behavior model $\pi(\mathbf{a}|\mathbf{o}, \mathbf{r}, \mathbf{m})$ that depends on these. The route variable defines the path that the agent takes through the road network, e.g., turning left at the next intersection, and the maneuver variable describes the interaction with other vehicles, e.g., taking or giving right of way at an intersection. These intentions influence the selected actions $\mathbf{a}$. By making predictions for multiple intentions, and tracking their accuracy over time using a Particle Filter (PF), the unknown intentions of surrounding drivers can be inferred. A later work [Sch+18c] proposes replacing the PF with a multiple model UKF to improve the accuracy or runtime of the intention inference.

Learning Driver Behavior Models: Behavioral Cloning Learning a behavior model from observed driving behavior instead of manually formulating it has many advantages: The manual effort is reduced, the model can be easily updated as new data becomes available, and it can be extended to handle various situations, given appropriate training data. Thus, as a followup to their manually formulated urban driver model, Schulz et al. [Sch+19] train a neural network policy as a replacement and conclude that "even simple feed-forward models are able to outperform the hand-tuned rule-based model".

The technique used by [Sch+19] and many others [WRK16; MWK17; Len+17] for training the policy is known as Behavioral Cloning [Arg+09; Fin+16] and dates back to the early days of automated driving [Pom89]. The core idea is to formulate the policy learning problem as a supervised learning problem. Given a dataset of trajectories, the policy neural network learns the mapping from (simulated) observations to actions. A detailed problem formulation and description of related work is given in Chapter 4.

BC inherits the limitations of supervised learning: The policy only yields reasonable actions for observations that are similar to the training dataset observations. Ross et al. [RB10] argue that this leads to a fundamental problem: As the policy is executed recurrently, and thereby influences its future observations through its actions, the distribution of observations during a simulative rollout according to Algorithm 1 differs from the distribution of observations on the fixed training dataset. This can lead to accumulating errors. For example, the policy may wrongly steer a vehicle towards the roadside. Having never seen how to recover from this in the training data, the policy has not learned to correct its fault, possibly steering the vehicle off

the road. This problem has widely been acknowledged [Fin+16; dHJL19] and is the central motivation of this thesis to explore different approaches to learning behavior policies. It is denoted as *covariate shift* [Spe+21], because the distribution of the input variable, commonly known as the covariate, changes between training and execution.

Reinforcement Learning Instead of learning a policy by imitation, the goal of RL is to find a policy that maximizes a manually defined reward function through interaction with a simulated environment [SB18]. The learning procedure consists of the agent repeatedly executing an initially random policy in the environment, similar to Algorithm 1, thereby collecting experience on which actions lead to high cumulative rewards. The successful actions are then reinforced, whereas the actions with low rewards are avoided. A successful policy is the result of iterative improvement in this way. The theoretical details are deferred to Chapter 5.

Many works use RL methods in an automated driving context, because it enables learning a policy through pure interaction with the simulated environment without the need to explicitly formulate an environment model. Some examples address tasks from highway driving [HWL20], merging [Bou+20], navigating occluded intersections [Ise+18; Kam+20] to camera-based end-to-end control of an automated vehicle [Ken+19]. The goal of these works is to learn a driving policy to control an automated vehicle. In contrast, the foreseen application of RL in this thesis is to model the behavior of surrounding vehicles for prediction or simulation purposes.

RL eliminates many limitations from BC: No training data is required, the policy can be trained in arbitrary simulated situations and is shaped by the reward function to comply with its requirements, e.g., to avoid collisions. On the other hand, this implies that the link to real world is cut and that it is hard to obtain policies that resemble the behavior of human drivers. To do so, one must assume that humans, too, act to maximize a subconscious reward function that encodes their driving preferences: Making progress towards their destination while having low longitudinal and lateral accelerations and driving safely. While easy to verbalize, it involves a lot of manual effort to formulate a mathematical reward function, as it is unclear how exactly these aspects should be rewarded and what their relative importance is [FLA16; Nau+20].

Inverse Reinforcement Learning is a principled approach to automatically reconstruct the reward function of human drivers based on a dataset of real-world trajectory demonstrations. The procedure works as follows: For an initial guess of a reward function, expressed as a weighted sum of reward terms, a policy is learned using a standard RL algorithm. Then, the difference between the trajectories from the policy and the demonstrations is evaluated. Based on this difference, the weights of the reward function are adapted, and a new policy is

trained. This repeats until the policy produces trajectories that are reasonably similar to the demonstrated trajectories. A more thorough description of the approach along with related works is given in Chapter 6.

For the purpose of behavior modeling, Inverse Reinforcement Learning (IRL) methods combine the benefits of BC and RL: The training goal is the imitation of driver behavior, similar to BC, and leads to policies that can accurately predict or model human behavior. The indirection of reconstructing a reward function and using RL to train a policy of this guessed reward function implies that the training is not restricted to the trajectories from the dataset. Instead, the simulation that is used during RL training can explicitly put weight on important situations that rarely occur in the real world, such as near-collisions. Being faced with these situations often means that the policy learns an appropriate reaction. Moreover, the reconstructed reward function can be used to train the policy in fictional situations, again augmenting the experiences that the policy makes during training and improving its ability to generalize to unseen situations. A third advantage of IRL is the ability to create auxiliary training goals beyond pure imitation, such as penalizing vehicles for leaving the track or colliding.

Other Approaches Some works develop the same intuition of (2.11) and (2.12), that the prediction needs to be decomposed into stepwise predictions of individual agents, from a deep learning perspective. Recurrent Neural Networks (RNNs) [Zha+22] are a natural choice for these approaches, as they allow for modelling such sequential processes. As the general RNN architecture is not well suited for modeling long term effects, recent works employ specialized RNN architectures, mainly LSTM cells and Gated Recurrent Units (GRUs) [Zha+22].

Since these deep learning based approaches have no notion of a meaningful observation model, another method to share information between agents is required to enable their interaction. In a pedestrian context, Alahi et al. [Ala+16] thus propose *social pooling*, whereby past trajectories of each pedestrian are encoded by a LSTM network to a hidden state. A social pooling layer aggregates hidden states from spatially close pedestrians. The prediction model leverages this aggregated information to predict the next position of each pedestrian. The trajectories are predicted by repeated execution of this scheme. The information from the social pooling layer enables the predictions to interact, e.g., by avoiding close encounters and making way for others. Gupta et al. [Gup+18] improve the pooling mechanism and combine it with Generative Adversarial Networks (GANs) [Goo+14] to enable the prediction of multimodal future trajectories.

The idea of social pooling is picked up by Lee et al. [Lee+17] and Deo et al. [DT18] for predicting vehicle trajectories. However, while the original social pooling paper [Ala+16] proposes a stepwise rollout of the model, enabling a social pooling operation during every step, [Lee+17] and [DT18] only pool once at the prediction origin and then proceed to

directly predict the future trajectories of all agents independently, effectively preventing true interaction between the predictions. Despite the use of the social pooling mechanism, these approaches thus fall into the category of passive interaction-awareness.

Tang et al. [TS19] introduce a different mechanism to share information between agents during a stepwise rollout of a prediction model. A joint world state is predicted step by step. To this end, one GRU network is instantiated for each agent. Each GRU receives the perspective of the corresponding agent on the rasterized map, including the encoded information on surrounding agents. Based on this information, the GRU predicts the parameters of a normal distribution that characterizes the next position of the agent. To enable the model to express multimodality due to path uncertainty, an additional discrete latent variable is introduced per agent, following the idea of (2.10).

Similarly, Rhinehart et al. [Rhi+19] roll out a RNN iteratively to predict the next position of each traffic participant, given the previous predictions of all surrounding vehicles. Other actors are directly represented via their relative positions in a top-view LiDAR gridmap, centered around each target vehicle. As the trajectory prediction is the result of the concatenation of multiple single-step networks, the training error can be directly backpropagated through multiple timesteps, circumventing the covariate shift problems discussed on page 28 of the BC approach. To represent uncertainty, the stepwise model is implemented as an invertible generative model that learns to transform samples from a normal distribution to the potentially more intricate target distribution of the prediction. One central requirement of the authors is the ability to condition the predictions on the planned trajectory of an automated vehicle, enabling the deployment in a holistic planning and prediction scheme as described in Section 2.1.1.

2.4.3.3 Proactive Interaction-Awareness

By design, behavior models that follow the scheme in Algorithm 1 are *purely reactive*, as agents do not reason about the future influence of their actions on others. The behavior model can be interpreted as a driving heuristic that implicitly anticipates the future traffic situation and acts accordingly. For example, agents in our reactive RL based method [**Sac+22a**] carefully approach the entry of a roundabout when it is unclear whether they can enter, but enter swiftly when a sufficiently large gap is encountered.

The limit of this implicit anticipation is reached when agents are faced with situations that are very different from the training situations. For example, Hoel et al. [HWL20] train a policy in regular highway scenarios using RL, but challenge it with wrong-way drivers at test time. While standard RL algorithms are unable to handle such domain shifts, the authors propose an approach to identify out-of-training situations and use safe actions (brake hard) in these cases. However, while braking hard is safe for the specific scenario, it is not a universally safe action, e.g., when a rear vehicle is closely following.

To make plausible predictions even in untrained situations, the idea of proactive interaction-awareness is to use behavior planning algorithms to plan for others, and use this plan as a prediction of their future trajectory. Compared to reactive behavior policies, true planning algorithms can flexibly adapt to unseen situations, because they explicitly anticipate the evolution of the traffic situation. This leads to a joint optimization problem among the ego vehicle and all surrounding agents, requiring a game-theoretic solution paradigm [BDK20]. The solution is a joint plan that is optimal for every agent. The joint plan can be interpreted as a realization of $\mathbf{y}$, the future traffic situation.

Most of the works discussed in the following are not explicit approaches to prediction, but rather address the prediction problem as a by-product of a behavior planning algorithm. The applications range from highway driving [LKK16; Sad+18; Li+18; Fis+19; Bur+22], merging [GS19; Bou+20], planning overtaking maneuvers [Sch+17b], to unsignalized intersections [Sad+18; Pru+19] and roundabouts [Pru+20].

All approaches that follow the holistic planning and prediction scheme described in Section 2.1.1, i.e. that plan not only with respect to the cost function of the vehicle in question, but also with the cost imposed on others, can be interpreted as approximate solution methods for a Partially Observable Stochastic Game (POSG) [BDK20]. Briefly, they are partially observable, because of limited perception of surrounding vehicles and their intentions, and they are stochastic if an agent cannot manipulate the world state deterministically. A formal definition is given in [BDK20]. Solution approaches to POSGs also go by the name of cooperative behavior planning.

With many agents and continuous observation and action spaces, finding the optimal solution of the POSG quickly becomes computationally intractable [Fis+19]. Common approaches to simplifying the problem are to consider only a limited number of interacting vehicles [YL12; LKK16; Sad+18; Pru+19; Fis+19; Bur+22], using only few discrete actions [Pru+19; Pru+20] or a high-level problem formulation that enforces a limited number of observations and actions [Oyl+16; LKK16; Li+18; GS19; GS20].

An elegant combination of high- and low-level planning is proposed by Fisac et al. [Fis+19] and Schulz et al. [Sch+17b], where a long-term strategic goal, e.g., merging in front of or behind an approaching vehicle, is the result of a high-level plan, and the concrete trajectory is the result of an ensuing low-level plan. The central advantage of this hierarchical problem formulation is the ability to solve the joint POSG problem in a reduced high-level action space, and to determine tangible trajectories individually in a low-level space.

Another common simplification is the formulation of the problem as a Stackelberg game [YL12; YL13; Sad+18; Bur+22], where agents take turns sequentially, after observing the actions of the previous agents. For example, Sadigh et al. [Sad+18] assume that the automated vehicle chooses its plan first, and a human controlled vehicle acts according to a *best response*

to that plan. This effectively tames the combinatorial nature of the problem, as the solution can now be found by sequentially choosing optimal strategies for each agent. Interestingly, this inverts the sequential planning logic described in Section 2.1.1, where the plan of an automated vehicle is the best response to the predictions of all surrounding vehicles [Fis+19]. Burger et al. [Bur+22] raise the concern that with the Stackelberg problem formulation, the automated vehicle may behave obstructively towards others. When their trajectory is planned as the best response to the trajectory of the automated vehicle, the others are effectively predicted to always put up with the behavior of the automated vehicle. To find a cooperative and courteous plan in a Stackelberg game, a cooperative cost function is proposed as well as a method to consider safety constraints in the trajectory optimization.

Finally, exact solution methods, such as [Pru+19; Pru+20], search for a *Nash Equilibrium*, where no participant can improve its obtained reward by changing its strategy [Nas51; Kre89]. A solution becomes possible by restricting each agent to select between few acceleration profiles along fixed paths.

As the line between RL based methods and planning based methods is blurred, some RL approaches are based on a game-theoretic fundament. For example, [Li+16; Li+18; GS19; Bou+20; GS20; AY22] use *level-k* reasoning [SW95]: after manually formulating a "level-0 policy", an agent is iteratively trained to learn a level-k policy in an environment populated with agents controlled by the previous level-$(k-1)$ policy. Köprülü et al. [KY21] extend this idea by allowing the agent to dynamically select a policy of an adequate level at runtime. The previously discussed single-shot prediction approaches [Lee+17; KKC20], which involve iteratively predicting future trajectories for all agents and using these predictions as input in subsequent iterations, share similarities with the level-k idea. However, these approaches create levels of concrete trajectory predictions instead of levels of behavior policies.

2.4.4 Other Prediction Methods

Gridmap-Based Prediction Ondrúška et al. [OP16; Ond+16] propose an integrated detection, tracking and prediction framework for an occupancy grid map representation of the environment of a robot using a RNN. Hoermann et al. [HBD18] pursue a similar idea to predict the occupancy grid around an automated vehicle using a CNN for up to $3\,\mathrm{s}$. The key advantage of directly operating on occupancy grid pixels is that no model assumptions are required and thus arbitrary traffic participants (e.g., cars, pedestrians) can be handled. Moreover, a probabilistic occupancy gridmap is the most flexible representation of uncertain future positions. However, interaction is not considered in these approaches and long-term predictions occupy large areas of the map with occupancy probabilities, making the predictions only suitable for short-term collision avoidance. Fast and Furious [LYU18], IntentNet [CLU18] also operate on raw Light Detection and Ranging (LiDAR) point clouds or occupancy grids, but try to detect individual objects first before predicting their trajectories, all

within one single neural network trained to solve 3D detection, tracking and short time motion forecasting. MultiXNet [Dju+21] adds a second stage to IntentNet to produce multimodal trajectory predictions with uncertainty characterization.

Set-Based Prediction Similar to gridmap-based predictions, the main application of set-based predictions is short-term collision avoidance. Consider the complete set of states that other traffic participants can physically reach within a limited time horizon. An automated vehicle can plan a guaranteed safe trajectory if it does not intersect with this set. Althoff [Alt10] develops multiple methods to construct the reachable sets of an automated vehicle and surrounding traffic participants. Thereby, *conventional reachability analysis* can formally prove the safety of an automated vehicle, whereas *stochastic reachability analysis* estimates the probability of reaching unsafe states, allowing to compare different planned trajectories with respect to their safety.

As the set of *physically* reachable states of others grows rapidly, it prohibits any close interaction with other vehicles. In subsequent work, Koschi et al. [KA17] instead propose the concept of *legal safety*, which assumes that other traffic participants abide by the traffic rules, restricting their reachable set to the *legally reachable set*. The reachable set is extended when drivers violate traffic rules. Based on these ideas, Pek et al. [Pek+20] propose a method that ensures the existence of safe fallback trajectories, thereby guaranteeing that the automated vehicle never causes a collision.

Log Replay Strictly not a prediction approach, the simplest form of simulating other drivers is a direct playback of recorded trajectories (*log replay*) [Ber+21]. This is only applicable for retroactively simulating traffic scenarios, but not for on-line prediction of currently observed situations. Typically, one vehicle under test is controlled by an algorithm, whereas all other vehicles are played back from data, e.g., [MWK17]. To reliably succeed in a log-replay simulation environment, the planned trajectory needs to be close to the original trajectory of the vehicle under test. If the planned trajectory is different, this might lead to collisions that are not caused by a fault of the trajectory planner, but due to the non-reactiveness of the surrounding played back vehicles. Bergamini et al. [Ber+21] describe this phenomenon as *simulation drift*.

2.4.5 Discussion

Except for the gridmap-based approaches, most works share the assumption that environment perception is an upstream problem, and directly operate on the state estimates of detected objects. Moreover, all surveyed approaches that make use of map data assume that the map is

given. An overview of how the contributions from this work connect with previous work in the field is provided in Figure 2.8.

Deep learning methods have made it possible to incorporate the entire environment, including nearby vehicles, as the input of the prediction algorithms. Thus, most recent works belong to the category of interaction-aware models. However, the subdivision between passive, reactive and proactive interaction-awareness highlights significant differences in the architecture of these models. The flow of information inside the different approaches is depicted in Figures 2.5 to 2.7.

Passive interaction-aware models use all information available at the prediction origin, including current and possibly past states of surrounding vehicles. However, they effectively predict the trajectory of each agent independently based on this information. These approaches remain popular, because they are conceptually easy to implement, and because the metrics of many prediction benchmarks do not penalize the logical inconsistencies that arise as a result of the independent prediction approach [Ett+21; Tol+21]. One appealing advantage of these approaches is the ability to use a single-shot architecture, i.e., the prediction network directly outputs the full predicted trajectory of an agent, which is computationally cheap.

In contrast, reactive approaches predict future trajectories step by step by repeatedly executing a single-step behavior model for each agent in a traffic situation. This enables true interaction between the agents at prediction time, because information can be exchanged between agents at every prediction step, as shown in Figure 2.7. Behavior models have a broader range of applications than passive prediction models: They straightforwardly enable hypothetical inference by fixating the trajectory of one or more agents before performing the simulation. Thus, these approaches can be used in both, a sequential or holistic cognition block (see Section 2.1.1). Trajectory predictions can be made for arbitrary lengths and respect the dynamic constraints of the kinematic state transition model. Moreover, depending on the modeling, the predictions are actual trajectories that allow for evaluating certain criteria along them, such as the longitudinal or lateral acceleration. This can be useful in cooperative behavior planning when reasoning about the influence of the plan of the ego on others, e.g., [Bey+19]. The behavior models can also be used for the simulation of the behavior of surrounding vehicles during the development of the cognition block. However, learning a behavior model that is both stable and accurate for long time horizons is significantly more difficult than the training of a passive prediction model. Therefore, this thesis investigates three approaches to construct such a behavior model: Behavioral Cloning in Chapter 4, Reinforcement Learning in Chapter 5, and Inverse Reinforcement Learning in Chapter 6.

Finally, proactive approaches raise the even harder problem of finding plans for all agents in a traffic situation that are jointly optimal. The resulting plan of surrounding vehicles can be interpreted as their prediction. While conceptually interesting, current practical implementations of this idea require severe simplifications to find a solution and are computationally

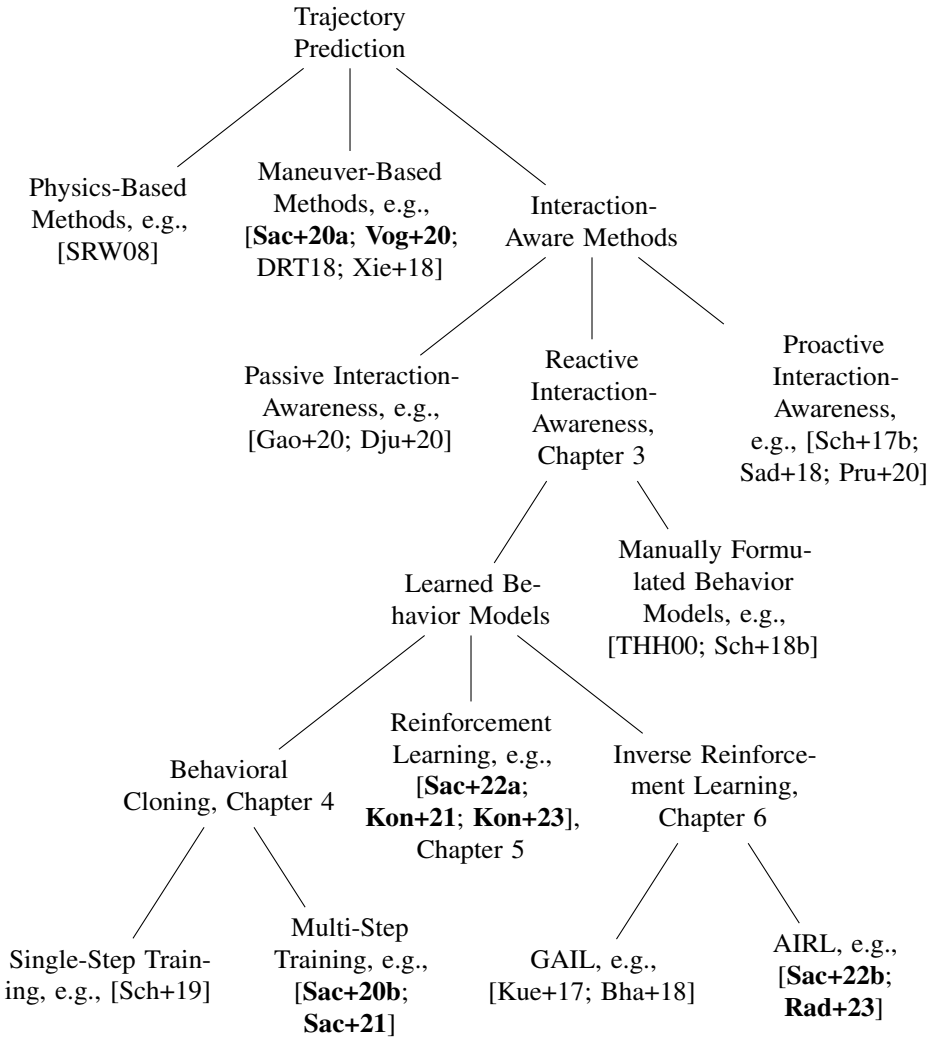


Figure 2.8: Overview of the most relevant approaches to trajectory prediction presented in Section 2.4, with some representative publications. The methods investigated in this thesis fall into the category of Reactive Interaction-awareness, where a microscopic traffic situation is executed with a behavior model for each agent to obtain the interacting predictions of the trajectories of all vehicles. Three different approaches to learning the behavior model are investigated: Behavioral Cloning, Reinforcement Learning and Inverse Reinforcement Learning. Publications associated with this thesis are emphasized with bold letters.

significantly more demanding than the execution of a reactive policy. Thus, these approaches are not further considered in this thesis.

2.5 Handling of Uncertainty

As discussed in Section 2.2, there are multiple sources of uncertainty: First, multiple routes in the road network induce multimodality in the prediction. The second source of multimodality is the different order in which maneuvers can be executed. Moreover, uncertainty within a single mode is caused by different driver characteristics, vehicle properties, and remaining aleatoric uncertainty. This section gives an overview on how different prediction approaches handle the uncertainty.

2.5.1 Conditioning as an Enabler

Maneuver-based approaches share the notion that predicting maneuver conditional trajectories $p(y^j|x^j, m^j)$, and separately estimating the maneuver $p(m^j|x^j)$, is more informative than directly predicting $p(y^j|x^j)$. This is backed by information theory [CT06, Theorem 8.6.1], as the (differential) entropy h never increases, given additional information $\mathbf{m}$, i.e., $h(\mathbf{y}|\mathbf{x}, \mathbf{m}) \leq h(\mathbf{y}|\mathbf{x})$. Moreover, if the prediction is executed repeatedly, as new observations x arrive, it often can be assumed that certain maneuver variables remain unchanged, e.g., the route that the agent is going to follow. Thus, the maneuver variable can be tracked over time using Bayesian filters, as for example proposed in [TF13; DRT18; Xie+18].

Some passive interaction-aware approaches [Cha+19; Zha+20; Gil+21; Zha+21] also introduce similar latent variables that encode different goal positions or paths on a map. Here, the motivation for introducing the new latent variable is often the ability to explicitly address multimodality of the road network and to circumvent the mode collapse problem (see p. 22) that occurs when directly predicting multiple hypotheses.

Reactive interaction-aware approaches, too, are often conditioned on latent variables that describe their goal. Goal conditioning is essential to these approaches, because the behavior models are always goal-directed. For example, a behavior model that controls a vehicle driving through a roundabout needs to know which exit to take.

All these approaches reduce the uncertainty in the predicted future traffic situation $\mathbf{y}$ by introducing uncertainty about latent variables $\mathbf{m}$. This thesis follows these ideas by introducing a latent variable that describes the path that a vehicle will take through the road network, i.e., whether a vehicle will take the first, second or third exit of a roundabout. Thereby, the major reason for multimodality is eliminated from the prediction model. Besides, this is the inspiration for Section 5.5 of this thesis, where additional latent variables are introduced that describe driver preferences, e.g., the desired time gap to a preceding vehicle, or the weight

of a lateral acceleration penalty. Thereby, even more uncertainty can be shifted from the conditional trajectory prediction $p(y|x, \mathbf{m} = m, \mathcal{M})$ to latent variables $\mathbf{m}$. Effectively, the conditional prediction model translates uncertainty from the low-dimensional preference space to the high-dimensional trajectory space.

2.5.2 Representation of Uncertainty

After factoring out the uncertainty induced by mode variables, the question of a representation of the future trajectories $p(y^j|x^j, m^j, \mathcal{M})$ remains. Some approaches [Cha+19; HSP19; Dju+20] directly produce a closed form density by describing the future positions $\mathbf{y}^j_{1:H} = (\mathbf{y}^j_1, \mathbf{y}^j_2, \dots)$ with each future position $\mathbf{y}^j_k$ as a 2D normal distribution characterized by its mean vector and covariance matrix. Such a closed-form specification of the density is desirable: It allows to analytically reason about future occupancies, but can also be used to draw samples of predicted positions in Monte-carlo based planning algorithms. Moreover, it is easy to train, interpret and visualize. However, modelling each future position individually as a Gaussian also implies the assumption that successive positions are conditionally independent, thereby neglecting the kinematic constraints that real trajectories are subject to. Thus, these approaches predict future occupancies, but not trajectories.

Addressing this issue, other approaches [Wie+12; Rei22] represent trajectories using basis functions such as Bézier curves, and predict the distribution of the coefficients of the basis functions. Each sample from the predicted coefficient distribution can be translated into a kinematically feasible trajectory.

However, both, a direct representation of the trajectory or a parametric representation via basis functions are incapable of expressing the correlation between predictions of different vehicles and are thus only suitable for passive behavior prediction. The same limitation applies to gridmap based representations of future occupancies such as [HBD18; Rid+20; Gil+21].

No surveyed related work yields a closed form description of the joint future trajectory distribution $\mathbf{y}_{1:H}$ of all agents that respects kinematic constraints along the trajectories as well as the interactions between all agents. Instead, many approaches predict one or multiple future trajectories [Lee+17; CLU18; Cas+20b; Suo+21], and optionally their probabilities, e.g., [Cui+19; Gil+21; Zha+21]. Using representative trajectories instead of distributions is also reflected by evaluation metrics in popular benchmarks such as nuScenes [Cae+20] and Argoverse [Wil+21], which require prediction approaches to generate multiple trajectory predictions per agent. The evaluation is based on the distance between the closest prediction and the ground truth.[3]

[3]The evaluation metrics of [Cae+20; Wil+21] are described thoroughly on `https://nuscenes.org/prediction` and `https://eval.ai/challenge/1719/evaluation` (Both: Accessed on May 30, 2022).

The fundamental idea behind these approaches is that high-dimensional random variables, such as the joint future trajectories, are too complex to be directly expressed in closed form due to the correlation between its components. Instead, samples from the density can be significantly easier to generate. This idea also lays behind the field of generative models, such as CVAEs [KW14] and GANs [Goo+14], which are prominently employed to generate realistically looking images, i.e., samples from a space with more than 10^6 dimensions [Kar+20]. In the context of trajectory prediction, a direct application of CVAEs has been proposed by [CLU18; Ma+19]. However, as discussed in Section 2.4.3.2, to properly model the sequential nature of trajectory formation and interaction between agents, this work focuses on the generation of trajectories using a behavior model. There are many similarities between GAN and IRL, which will be discussed in Chapter 6.

2.6 Environment Representation

The representation of the environment that forms the input of the prediction algorithm is a key differentiator of the different approaches to trajectory prediction. Standard neural networks, which underlie most recent interaction-aware prediction approaches, have a fixed number of inputs. This requires some consideration of how to represent a variable number of vehicles in a complex road network to the prediction algorithm.

Three major types of representations are used in related works: feature vectors, images, and graphs. This section discusses the advantages and disadvantages of each. To simplify the discussion, the focus is placed on the representation of the surroundings of a single target vehicle. To predict a situation with multiple interacting agents, this representation is computed from the perspective of each agent individually before the prediction is performed.

Feature Vector Based Representation A natural approach to representing the situation that the target vehicle is faced with is a feature vector. Each component of the feature vector describes a different property of the situation. A simple car-following situation can fully be described by three parameters: the speed of the target vehicle, as well as the distance and speed difference to its preceding vehicle. These features are for example used in the IDM [THH00] and the Krauß model [KWG97]. Models that address more complex situations require additional features. For example, Lenz et al. [Len+17] describe multi lane highway situations via the speed of the target vehicle, and the sizes, relative distances and velocities of up to seven neighboring vehicles. Values of vehicles or lanes that are not present are indicated by a boolean, and the corresponding feature values are set to 0. Similar representations for highway situations are used by [HWL18; HWL20] and the rule based MOBIL model [TK09]. The advent of learning-based methods enabled the use of additional features, such as those from multiple past timesteps [Lef+14].

In urban situations, even more features are required for a proper representation of the situation. Schulz et al. [Sch+19] proposes a model that uses more than 40 features to describe the target vehicle state, road course, traffic rules, and interactions with other vehicles. Importantly, the proposed model is conditioned on the route that the agent takes through the road network. This is implemented via the variables that describe the future road course of the vehicle: For example, if the route intention is to turn left at the next intersection, the road course variables are different than if the vehicle were to turn right. The road course is described by the road curvature at different distances and relative angles to the road centerline at different distances. This thesis uses a similar representation that is described in detail in Section 3.2.

Image-Based Representation Many works, especially passive interaction-aware approaches, use an image-based representation of the traffic situation. The advantage of this representation is that it can flexibly represent complex traffic situations without the effort of manually defining features. One example of this approach is proposed by Henaff et al. [HCL19], where the channels of an RGB image are used to represent lane markings, the target vehicle and surrounding vehicles from a birds-eye-view perspective centered around the target vehicle. This image is processed by an ordinary CNN. Like many other image-based approaches, the position and speed of the target vehicle is explicitly fed into the model by concatenating it with the output of the CNN. To add historic context, this operation is performed for the 20 most recent past timesteps, and the result is again concatenated. This jointly encoded information is then processed by a neural network that performs the prediction. Many other works, such as [Cui+19; Dju+20; Ber+21], use similar birds-eye-view rasterized RGB images. To decode the information from the image for the ensuing trajectory prediction, advanced CNN architectures such as MobileNetV2 [San+19] or ResNet-50 [He+16] are employed. These were originally targeted at computer vision applications. The use of these complex architectures illustrates that a significant computational effort is introduced when image-based representations are employed. Also, image-based representations typically use more memory than feature vector based representations. For example, [Dju+20] uses a 300x300 image with 3 channels, which is multiple orders of magnitude more data than any feature vector based representation.

One notable other stream of work [LYU18; CLU18; Dju+21] directly operates on raw sensor data projected into a birds-eye-view image, such as voxelized LiDAR measurements. To add context, additionally rasterized maps are leveraged. These approaches thereby address the perception and prediction problem simultaneously.

Graph-Based Representations strive to combine the expressiveness and compactness of feature vectors with the flexibility of image-based representations. A seminal work in this domain is VectorNet [Gao+20]. The authors propose to represent all relevant map elements as

well as surrounding vehicle trajectories as polylines. A polyline consists of multiple connected line segments. A polyline graph is constructed by defining each line segment to be a node of the graph, with the start and end position as its features. Each polyline graph is then processed by a Graph Neural Network (GNN). Similar to CNNs, features are aggregated using pooling functions, such that each polyline is described by a fixed-length vector.

In a second step, these polyline-level features of map elements and surrounding agent trajectories form a "global interaction graph" that is again processed by a GNN. Based on this encoding, the final prediction network predicts the future trajectory of the target vehicle. Compared to a baseline approach that uses rasterized images as input, the required Floating Point Operations (FLOPs) are reduced by an order of magnitude and the amount of network parameters is reduced by 70% [Gao+20].

As graph-based representations are a very recent development, no consensus on the representation has emerged. Significantly different graph-based architectures are for example proposed in [Die+19; Lia+20; Cas+20a; JDZ22], but are omitted here for brevity.

Discussion The recent advances in graph-based representations indicate that they will eventually replace image-based representations, as they offer the same flexibility with significantly reduced computational overhead. Nevertheless, this work builds on a feature vector based representation for three reasons: First, it enables to precisely control the information that is processed by the behavior model, as different feature sets can be easily defined. Secondly, each input of the resulting behavior model has a clear meaning, which allows to reason about the function of the behavior model by examining the changes in the output when manipulating individual inputs. Thirdly, in contrast to graphs, feature vectors have a fixed size, which significantly facilitates the implementation by enabling the use of efficient data structures and vectorization operations as provided by the scientific computing packages NumPy [Har+20] and PyTorch [Pas+19]. This also allows for the use of comparatively simple, small and hence fast neural network architectures. However, the methods proposed in this work could also be realized with image- or graph-based representations. This is demonstrated in a followup work [**Kon+23**], where the Reinforcement Learning approach presented in this thesis is augmented with a graph-based environment representation.

2.7 Conclusion

One important application of behavior prediction is behavior planning. The combination of prediction and planning can be made sequentially, i.e., first predict, then plan. However, this leads to overcautious plans and ultimately to the frozen robot problem, which is why recent works in behavior planning explore a holistic planning approach that estimates the impact of the planned behavior on others. This holistic planning requires the prediction to be

capable of hypothetical inference, i.e., the predictions of surrounding vehicles need to react to hypothetical plans of the ego vehicle.

The widely acknowledged survey on motion prediction by Lefèvre et al. [LVL14] differentiates between physical, maneuver-based and interaction-aware prediction models. Of these, only the latter are capable of predicting the interaction between vehicles. More specifically, this thesis further subdivides interaction-aware models into passive, reactive and proactive models, discussed in detail in Section 2.4.5, and explains why only reactive and proactive interaction-aware models are truly capable of modeling the interaction between predicted trajectories. Due to the severe simplifications that most proactive approaches entail, this thesis focuses on reactive models.

Reactive models make predictions by rolling out a single-step behavior model for each participant of the traffic situation. Through this mechanism, interaction can properly be modeled and hypothetical inference can be performed. The accuracy of the predictions however strongly depends on the quality of the behavior model. Manually formulated behavior models are typically collision-free, but inaccurate for long-term predictions. Learning behavior models is an appealing alternative, but challenging because prediction errors accumulate and are fed back into the model during the execution. For this reason, this thesis explores three methods to obtain more robust behavior models with Behavioral Cloning and (Inverse) Reinforcement Learning. Another important benefit of reactive prediction models is that the obtained behavior models can also be directly employed for the simulation of driver behavior, for example to test the interaction between an automated driving function and surrounding vehicles in a simulator.

Concerning uncertainty, conditional maneuver variables are very advantageous for addressing the multimodal nature of the prediction and are therefore used in this work to differentiate between potential future routes. As the density of the future traffic situation $\mathbf{y}$ in closed form is highly entangled, this thesis instead predicts a sample from the density, similar to most related works.

Lastly, different representations of the traffic situation as an input to the prediction neural network are compared. This thesis employs a feature-based representation, because it enables controlling the information used by the prediction model and facilitates insight into its functioning.

3 Simulation Setup

Parts of this chapter have been published in [**Sac+21**; **Sac+22a**; **Sac+22b**].

This chapter describes the simulation that forms the basis for the evaluation and training of the methods described in the following chapters. In Figure 1.2, this chapter closes the first link: It shows how a set of interacting trajectories is obtained by executing a policy for each vehicle in a simulated traffic situation.

The simulation setup is depicted in Figure 3.1: Following the convention of RL [SB18], this work distinguishes between environment and agent. An agent controls a single vehicle by making observations from the environment and selecting appropriate actions via its behavior model, also called policy. This chapter assumes that the policy is given; the following chapters describe how it can be obtained. The environment simulation calculates the effect of the action using a kinematic bicycle model by updating the state of the agent accordingly. Then, the relations between the vehicles are determined, i.e., which vehicles are preceding and conflicting. Finally, the observation model is executed to determine the next local observation of each vehicle and the cycle begins anew.

The simulation of the traffic situation is one realization y of the random variable $\mathbf{y}$ that characterizes the future traffic situation. Equivalent to the general problem formulation in Section 2.3, the initial simulation state $y_0 = y_0^{1:N} = (y_0^1, y_0^2, \dots, y_0^N)$ comprises the states of all agents $1 \dots N$ at the initial timestep 0. To predict the evolution of the traffic situation, the simulation loop in Figure 3.1 is executed simultaneously for each agent in the traffic situation. Hypothetical inference can be performed by fixating the trajectory of one or more agents, whereas the remaining agents are controlled through the simulation loop. To represent uncertainty, the kinematic model, the observation model and the policy are formulated as probabilistic models and allow for additional noise or conditional variables. Multiple uncertain predictions of the same traffic situation can be generated by repeating the simulation from y^0 with different conditional or noise variables.

3.1 Kinematic Model

For a realistic simulation, a kinematic model that respects the non-holonomic constraints of real world vehicles is required. For this purpose, similar to [Sch+19], the kinematic bicycle model [WQ01] is employed. It is a simple kinematic model that allows for modelling the inability of cars to change their orientation in standstill, but ignores tire slip.

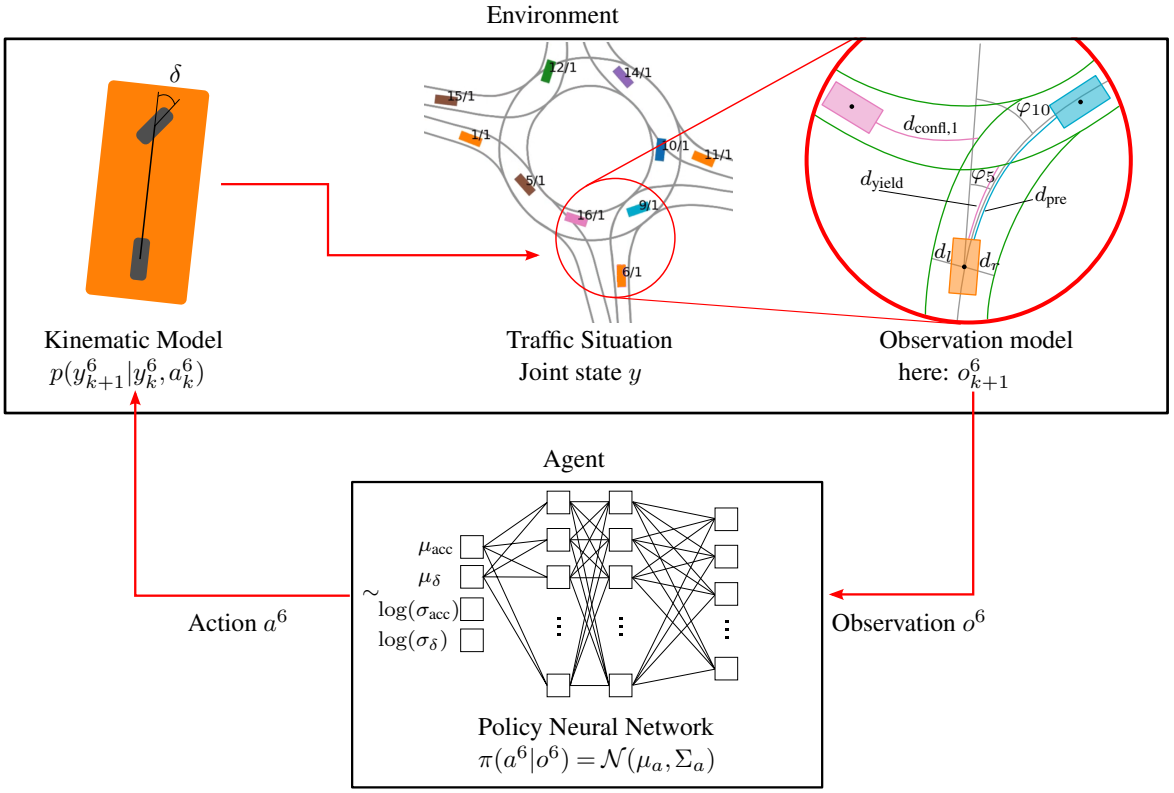
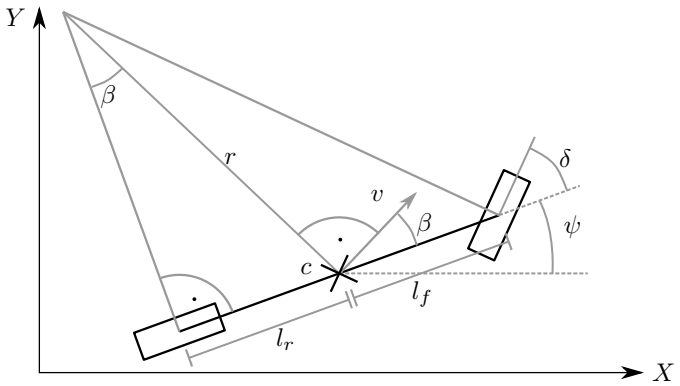


Figure 3.1: Simulation from the perspective of one agent 6. The action is selected deterministically by the policy network as $a^6 = (\mu_{\text{acc}}, \mu_\delta)$. Alternatively, it can be drawn from the distribution $\mathcal{N}(\mu_a, \Sigma_a)$, parameterized by $\mu_a = (\mu_{\text{acc}}, \mu_\delta)$ and the diagonal covariance matrix $\Sigma_a = \text{diag}(\sigma^2_{\text{acc}}, \sigma^2_\delta)$, indicated by the tilde. Image adapted from [**Sac+22a**].



δ: Steering angle
β: Slip angle
ψ: Heading angle
c: Centre of gravity (COG)
r: Turning radius
l_r, l_f: Distance axles to COG
v: Speed

Figure 3.2: Kinematic bicycle model, adapted from [Kon+15]

This work adapts two simplifying assumptions from [Kon+15] compared to [WQ01]: The rear wheel cannot be steered and only the effective speed at the center of gravity is considered instead of the individual wheel speeds. In doing so, the kinematic equations

$$\dot{p_x} = v\cos(\psi + \beta) \tag{3.1}$$

$$\dot{p_y} = v\sin(\psi + \beta) \tag{3.2}$$

$$\dot{\psi} = \frac{v}{r} = v\frac{\sin\beta}{l_r} \tag{3.3}$$

$$\beta = \arctan\left(\frac{l_r}{l_f + l_r}\tan(\delta)\right) \tag{3.4}$$

$$\dot{v} = a_{\text{lon}} \tag{3.5}$$

can be derived from the trigonometric relations in Figure 3.2. Hereby, (p_x, p_y) is the position of the center of gravity. The remaining variables are described in the figure. The speed v is a scalar value along the current direction of movement. The model is controlled via the acceleration a_{lon} and the steering angle δ, which are the output of the behavior model. Apart from ensuring kinematic feasibility of the predictions, using the kinematic bicycle model inside the simulation enables keeping track of the full kinematic state $(p_x^j, p_y^j, \psi^j, v^j)$ of each agent. In RL, this is used to evaluate the discomfort of trajectories, e.g., by assessing the longitudinal and lateral acceleration. For this purpose, the lateral acceleration

$$a_{\text{lat}} = v^2/r = v^2\frac{\sin\beta}{l_r} \tag{3.6}$$

is approximated via the current turning radius r. To ensure physical plausibility, the kinematic parameters of an Audi A6, listed in Appendix C.1, are used for the simulation. The longitudinal acceleration is restricted to be in $(-7, 3)\,\mathrm{m/s^2}$ and the steering angle is restricted to be within $(-\pi/7, \pi/7)\,\mathrm{rad}$. Driving in reverse is ruled out by the simulation.

Apart from the kinematic state, this work follows the modelling of [Sch+19], such that the full simulation state $y^j = (p_x^j, p_y^j, \psi^j, v^j, m^j)$ of an agent additionally contains a conditional maneuver variable m^j, which describes the route that an agent takes through the roundabout, e.g., whether it takes the first or second exit. The route is encoded as the list of road segments that the agent will drive through.

3.2 Observation Model

The observations that the agents base their decisions on are of major importance. They characterize the local environment of an individual agent. The observation is expressed as a vector of features of the local environment of an agent.

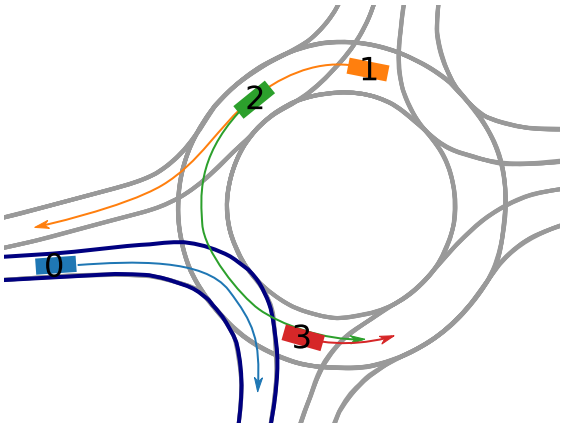


Figure 3.3: Illustration of relations between vehicles, which also depend on the route intentions:
Preceding: 2 is the preceding vehicle of 1; 3 is the preceding vehicle of 2. Because it leaves at the first exit, 0 has no preceding vehicle. Otherwise, 3 would be the preceding vehicle of 0.
Conflicting: As the route intentions of 1 and 2 are unknown to 0, both are conflicting vehicles of 0 that have priority. As soon as it leaves the roundabout, 1 loses its status as a conflicting vehicle to 0. When 2 enters the highlighted corridor of 0, it becomes the preceding vehicle of 0 and is no longer its conflicting vehicle.
Non-Priority: In turn, 0 is currently a non-priority vehicle to 2. However, 0 is not a non-priority vehicle to 1, because 1 plans to leave the roundabout before the entry of 0.

The first part of the observation vector describes the current state of the vehicle in relation to the map, i.e., the speed, the distances to the road boundaries d_l, d_r and the heading angle with respect to the future road course $\varphi_{0...20}$ as well as the road curvature $c_{0...20}$. Importantly, these features depend on the route intention of the vehicle, e.g., the angles to the road center differ depending on the route that is encoded in the maneuver variable m^j.

The remaining components of the observation vector describe the relation to relevant surrounding vehicles, i.e., the preceding vehicle, the closest two conflicting vehicles that have right-of-way at the next merge point, and the closest non-priority vehicle that has to yield at the next merge point. These relations also depend on the route intention of each agent, as illustrated in Figure 3.3. The relation to the preceding vehicle is described by the bumper-to-bumper distance d_{pre} between the vehicles and their velocities. In right-of-way situations, the relation is characterized via the speed of the agent and its conflicting vehicle v and $v_{\text{confl,1}}$ as well as their distances to the merge zone d_{yield} and $d_{\text{confl,1}}$. A representation of the second-closest conflicting vehicle is required for the behavior model to decide whether it can enter directly behind the closest conflicting vehicle.

Throughout the publications that this thesis is based on, the observation model has been continually extended. The fundamental feature set was introduced in [**Sac+21**]. Later, [**Sac+22a**] introduces features that describe non-priority vehicles that are approaching the roundabout $(d_{\text{merge}}, v_{\text{nonpr}}, d_{\text{nonpr}})$ to improve the model performance in situations where a

Table 3.1: Components of the observation vector o

Feature	Symbol	Unit	Default	max	μ	σ
Speed	v	$\mathrm{m/s}$	–	–	$6.7\,\mathrm{m/s}$	$3.69\,\mathrm{m/s}$
Distance to left and right boundary	d_l, d_r	m	–	–	$2.27\,\mathrm{m}$, $1.97\,\mathrm{m}$	$0.69\,\mathrm{m}$, $0.76\,\mathrm{m}$
Heading relative to lane in $\{0,5,10,20\}\mathrm{m}$	$\varphi_{0...20}$	rad	–	–	≈ 0	$0.08, 0.1, 0.12, 0.25$
Road curvature in $\{0,5,10,20\}\mathrm{m}$	$c_{0...20}$	m^{-1}	–	–	≈ 0	$\approx 0.04\,\mathrm{m}^{-1}$
Speed of preceding vehicle	v_{pre}	$\mathrm{m/s}$	v	–	$7.2\,\mathrm{m/s}$	$3.6\,\mathrm{m/s}$
Distance to preceding vehicle	d_{pre}	m	$30\,\mathrm{m}$	$30\,\mathrm{m}$	$21.4\,\mathrm{m}$	$8.7\,\mathrm{m}$
Distance to next yield line	d_{yield}	m	$40\,\mathrm{m}$	$40\,\mathrm{m}$	$32.1\,\mathrm{m}$	$12.4\,\mathrm{m}$
Speed of conflicting vehicle	$v_{\text{confl,1}}$	$\mathrm{m/s}$	$5\,\mathrm{m/s}$	–	$5.7\,\mathrm{m/s}$	$1.1\,\mathrm{m/s}$
Distance of conflicting vehicle to merge zone	$d_{\text{confl,1}}$	m	$40\,\mathrm{m}$	$40\,\mathrm{m}$	$31.3\,\mathrm{m}$	$13.3\,\mathrm{m}$
Angle of conflicting vehicle to merge point	ψ_{confl}	rad	$\pi/2$	–	1.26	0.48
Speed of 2nd conflicting vehicle	$v_{\text{confl,2}}$	$\mathrm{m/s}$	$5\,\mathrm{m/s}$	–	$5.24\,\mathrm{m/s}$	$0.75\,\mathrm{m/s}$
Distance of 2nd conflicting vehicle to merge zone	$d_{\text{confl,2}}$	m	$40\,\mathrm{m}$	$40\,\mathrm{m}$	$38.1\,\mathrm{m}$	$5.61\,\mathrm{m}$
Distance to next priority merge zone	d_{merge}	m	$40\,\mathrm{m}$	$40\,\mathrm{m}$	$36.7\,\mathrm{m}$	$8.76\,\mathrm{m}$
Speed of non-priority vehicle	v_{nonpr}	$\mathrm{m/s}$	$0\,\mathrm{m/s}$	–	$1.67\,\mathrm{m/s}$	$3.2\,\mathrm{m/s}$
Distance of non-priority vehicle to merge zone	d_{nonpr}	m	$40\,\mathrm{m}$	$40\,\mathrm{m}$	$29.2\,\mathrm{m}$	$16.3\,\mathrm{m}$

non-priority vehicle violates right-of-way rules, thereby requiring an inner-roundabout agent to brake slightly. [**Sac+22b**] additionally introduces ψ_{confl}, the heading angle of the conflicting vehicle towards the merge point, thereby enabling an agent that wants to enter the roundabout to detect earlier whether a conflicting vehicle will leave the roundabout. For comparability, all experiments described in this thesis have been repeated with the full set of observations.

The observation vector is partially illustrated in Figure 3.1, with all components listed in Table 3.1. It forms the input of the policy neural network. To ensure stability of the neural network, the inputs need to be bounded and are therefore limited to the maximum values indicated in the table, if the features are unbounded. All features have a natural minimum bound, for example the distance to the next yield line. After the yield line has been crossed and no next yield line exists, its value is set to the default. The same procedure is used for the distance to the merge zone. These natural bounds are not explicitly stated in the table. In the case of distances to other vehicles, the upper limit can be interpreted as the limited visual range of an agent.

If one feature is missing, for example because there is no preceding vehicle, it is replaced by a surrogate value. These values are selected to logically describe the situation. For example, if an agent has no preceding vehicle, the feature “distance to the preceding vehicle” is set to a large distance and the feature “speed of the preceding vehicle” is set to the own speed of the agent. Thereby, an uncritical car-following situation is described where the influence of the preceding vehicle is negligible. These default values are listed in Table 3.1, if applicable.

Moreover, to improve the convergence speed of the network and thereby reduce the training time, the features should be of approximately equal magnitude, and centered around 0 [Zha+22, Ch. 8.5.1]. Therefore, a standardization is performed before processing the features with a neural network by subtracting the mean and dividing by the standard deviation. These

values are empirically estimated on the training dataset described in the next section and also listed in Table 3.1.

3.3 Dataset

Given an initial situation, the simulation loop can be executed using a behavior model. To assess the quality of the model, the evolution of the simulated traffic situation needs to be compared to the ground truth situation. For this purpose, the simulation needs to be capable of playing back real world data. The second and equally important use of real world data is the generation of training data, which is used and explained in the chapters on BC (Chapter 4) and IRL (Chapter 6).

Trajectory Data The dataset was recorded at two different roundabouts near Ingolstadt (Germany) shown in Figure 3.4. The first roundabout is close to a village and has high traffic densities, especially at rush hours. The second roundabout is located further away in a rural area. Traffic is typically lower, but vehicles are driving faster. To obtain real-world data, the trajectory dataset was captured using a DJI Mavic 2 Zoom drone. While the data is captured using a drone, this does not imply that the models trained in this thesis require drone data as input. The observation vector described in Section 3.2 is designed such that it can also be determined from the on-board perspective of an automated vehicle. To this end, the perception system of the automated vehicle needs to determine the position, heading and speed of surrounding vehicles and must project them onto a high-definition map of the environment. Then, all observation features can be computed, regardless of the origin of the data.

Capturing data from an aerial perspective has many advantages over recording data from a vehicle: It allows to simultaneously observe the interaction between many vehicles and thereby realistically populate the simulation environment. There are no occlusions. Coherent vehicle trajectories can be observed for a long time, typically their whole way through the roundabout. A large quantity of trajectories and diverse driving styles can be captured in relatively short time. Finally, recording the data with a drone is less conspicuous than using a measurement vehicle and thus less likely to influence the observed behavior.

To capture the largest possible area of approximately $90 \times 160\,\mathrm{m}$, the drone was hovering approximately $100\,\mathrm{m}$ above the roundabout center, the legal maximum at time of recording. Each roundabout was filmed 8 times for approximately 20 minutes at different times of the day to capture low and high traffic densities. In total, approximately 8000 vehicle trajectories have been captured, with 270 to 900 vehicles in each recording.

Figure 3.4: Snapshot of the drone footage at both recording locations. Non-overlapping recordings from the left roundabout are used for training, validating and testing the learned models. Trajectories from the right roundabout are exclusively used for testing the ability of the model to generalize to untrained situations.

To extract the vehicle trajectories, the recorded videos were processed by the commercial service provider DataFromSky[1]. The processing steps are described in detail in [Ape+15]: Distortion correction of the video, geo-alignment of frames to maintain a stable perspective throughout the 20-minute recording, vehicle detection, transformation from pixel coordinates to a world-fixed 2D Cartesian coordinate system, and tracking of the detected vehicles using a particle filter. As the shape and kinematics of trucks and busses are often wrongly estimated, situations which include these vehicle classes are excluded from the dataset.

The result is a dataset of trajectories, where the position, speed, heading, and tangential and lateral acceleration of each vehicle is known at each time step. This data can be imported to the simulation framework. After determining the route that each vehicle takes, and its related vehicles, the local observations can be computed. Effectively, this converts the original trajectories, i.e., sequences of kinematic states, to sequences of local observations. Based on these, statistics for each component of the observation vector can be computed, and the mean and standard deviation for the standardization in Table 3.1 are determined. The histograms of all features are depicted in Appendix A.1.

A reference measurement was performed using a highly precise localization system to evaluate the quality of the trajectory dataset. The procedure and detailed results are described in Appendix A. The average displacement error of the trajectories from the drone footage is $0.25\,\mathrm{m}$, and in 95% of all cases below $0.6\,\mathrm{m}$. The speed estimate of the DataFromSky (DFS) pipeline is on average $0.027\,\mathrm{m/s}$ higher than the ground truth speed, with a standard deviation of $0.144\,\mathrm{m/s}$. Barmpounakis et al. [Bar+19] also evaluate the error of the speed estimate of the DFS pipeline on a different dataset. If the calibration is performed successfully, it is stated to be below $1.2\,\mathrm{km/h}$, which is in line with the measurements in Appendix A. For the evaluation of long term predictions, the position and speed errors can be considered negligible compared to the prediction error.

[1] `https://datafromsky.com/`, accessed on July 1, 2022

The acceleration estimate is on average $0.04\,\mathrm{m/s^2}$ higher than the ground truth. As the DFS pipeline needs to reconstruct the acceleration effectively as a second order derivative of the detected vehicle position, it is relatively noisy with a standard deviation of $0.26\,\mathrm{m/s^2}$.

Training, Validation and Testing Data One fundamental concept to reliably assess the performance of machine learning methods is the split between training, validation and testing data [Bis06, Ch. 1.3]. While the training data is directly used for improving the model, the validation data is used for indirectly improving the model, e.g., by tuning hyperparameters or determining when to stop the training. Finally, the test data is used exclusively for evaluating the model.

To avoid leaking information from the test or validation dataset to the training dataset, the data is split at a recording-level, as opposed to using some trajectories from one recording for training and others for validation. The training dataset comprises trajectories from 3 separate recordings captured at different day times and traffic densities at the left roundabout from Figure 3.4. It contains trajectories from 1007 different vehicles with a concatenated duration of approximately 180 minutes. The validation dataset contains trajectories from 289 vehicles from a different recording with a concatenated total duration of approximately 40 minutes. The testing dataset comprises trajectories from 3 different recordings at the first roundabout, containing 1169 different vehicles. Furthermore, 8 recordings from the second roundabout are used for testing with additional 1797 vehicles. This split between training and testing data allows to evaluate the ability of a learning method to make predictions on the same roundabout, as well as the ability to make predictions on a previously unseen roundabout.

Maps As described in Section 3.2, the models proposed in the following chapters rely on information on the road, such as the distance to the road boundaries, or the current curvature. To provide these values, a map is required. The map is created by manually tracing the boundaries of each road segment in Figure 3.4, and transforming their coordinates from pixel-values to a world-fixed 2D Cartesian coordinate system. Furthermore, the relation between segments is stored, i.e., preceding segments, succeeding segments, and the priority in right-of-way merges. From this information, a map with all possible routes through the roundabout as well as their priority in right-of-way situations is constructed automatically. Each route defines a Frenet coordinate system, i.e., a curvilinear coordinate system along the track center. A position in Cartesian coordinates can be transformed to the Frenet system defined by the route and vice versa. The route of a vehicle from the trajectory dataset is determined by transforming multiple points from the trajectory into the Frenet system of each possible route, and selecting the route with the lowest total lateral deviation. After determining the route of each vehicle, the relations between vehicles can be established, as previously illustrated in Figure 3.3.

3.4 Implementation

The simulation framework described in this chapter is a central component to the experiments in this thesis. It is used both for training and evaluating all policies presented in this work. This section gives an overview on the aspects that have to be considered during its implementation.

Differentiable Simulation To realize the multi-step training algorithm described in the next chapter, the simulation framework is implemented in a differentiable manner. This means that the framework can compute the gradient of the trajectory prediction error with respect to the actions selected by the policy during each step. This requires differentiating the outputs of all three components of the simulation framework with respect to their inputs—the kinematic model, each function of the observation vector in Table 3.1, and the policy. A justification for this claim is provided in Section 4.2.1. For the implementation, this means that no off-the-shelf simulation framework can be used, but that the simulation needs to be implemented from scratch in a software framework that enables automatic differentiation. For this purpose, PyTorch [Pas+19] is used. PyTorch and other automatic differentiation frameworks enable the regular implementation of a function, and automatically compute the analytic gradient of the output of the function with respect to one or multiple inputs. As the computation of gradients through the simulation incurs additional runtime, it is only enabled during multi-step training, whereas it is not required during evaluation or reinforcement learning.

Among prominent automatic differentiation frameworks, PyTorch was selected for the implementation, because it allows for direct and systematic debugging of all functions, equal to standard Python code. Moreover, the library provides methods for the implementation of deep neural networks, such as neural network architectures and optimizers that make use of the computed gradients. Hence, not only the simulation framework, but all policy neural networks implemented in this work and the optimizers used during training use components from the PyTorch library.

Speed of the Simulation Another important requirement to consider is the simulation speed. Throughout the course of this work, thousands of training runs have been performed. For quickly deciding which ideas are worth pursuing, training results should be available as fast as possible. In most methods presented in this work, many traffic situations need to be simulated during each training epoch. Hence, the simulation needs to be as fast as possible. In the following, multiple measures that are taken to reach a high simulation speed are described.

Vectorization Every component of the simulation is implemented in a vectorized manner, meaning that it can effectively process multiple data in parallel. With this, all vehicles are processed in parallel instead of sequentially during one simulation step. Moreover, multiple traffic situations are processed in parallel. At hardware level, PyTorch translates the vectorized function calls to Single Instruction, Multiple Data (SIMD) CPU instructions such as Intel's AVX [Gep17], where one instruction is performed on 8 float data points in parallel. Also, the vectorization implicitly leads to advantageous memory access patterns, because it operates on contiguous blocks of memory, meaning that the CPU cache is used effectively.

Overall, the parallelized simulation loop is executed as follows: First, each element of the observation vector is computed in parallel for all vehicles in all simulated traffic situations. Next, these observations are processed in parallel by the policy neural network to determine the actions. Finally, the kinematic model processes the actions of all vehicles in parallel to determine their next states. Then, the next simulation step starts with the computation of the next observations. For optimized memory access, all data is stored in contiguous memory, i.e., all vehicle states from all traffic situations are stored in one array, all observations are stored in one array, and so on.

Storing Intermediate Results Computing the 22 observation features from Table 3.1 is the most compute-intensive part of the simulation. To reduce the required computations, many intermediate results are stored. For example, splines through all possible routes on the map are only computed once per simulation. The transformation from Cartesian coordinates to the Frenet system is only performed once per vehicle per simulation step. Also, the relations between vehicles are stored between simulation steps and only re-evaluated when a vehicle crosses a segment boundary.

Optional Visualization A clear separation between the core simulation and the visualization is made. The simulation is executed without any visualization and stores the simulated trajectories in a data structure. This data structure can optionally be visualized in a second step.

Runtime Thanks to these points, 100 situations with a total of 1250 vehicles are simulated for 50 steps in approximately $4.5\,\mathrm{s}$ on a single core of an i7-9700 CPU. This means that approximately $14\,000$ state transitions are simulated in $1\,\mathrm{s}$ of CPU time. For the typical simulation step width of $0.2\,\mathrm{s}$, this means that $2\,800\,\mathrm{s}$ of driving can be simulated in $1\,\mathrm{s}$ of CPU time. Moreover, as the simulation only occupies a single CPU core, up to 8 independent training runs can be executed in parallel. This is used throughout this thesis, as the same experiment is often repeated multiple times, starting from different initial random neural

network weights. This is important to distinguish whether a change in performance can be attributed to a functional change, or simply to a lucky or unlucky network initialization.

For comparison, this is considerably faster than the widely used 3D traffic simulation Carla [Dos17], which approximately runs in real time, i.e., simulates $1\,\mathrm{s}$ of traffic in $1\,\mathrm{s}$ of CPU time, and requires a recent GPU for this. A recent work that also implements a relatively fast RL environment is introduced by Wang et al. [WKA21] and is stated to simulate approximately 500 transitions per second on server-grade hardware.

4 Direct Policy Learning: Behavioral Cloning

Parts of this chapter have been published in [**Sac+20b**; **Sac+21**].

The previous chapter illustrated how trajectory predictions are generated by executing a driver behavior model, denoted as policy. However, it is yet unclear where this policy comes from. It is the missing link to close the simulation loop in Figure 3.1. Its task is to select appropriate actions $a \in A$, i.e., steering and acceleration, given the current observation $o \in O$ of the local environment of an agent. Therefore, this chapter addresses the reverse process of the previous chapter: learning a policy from a set of observed trajectories. With this, the second connection in the behavior triangle in Figure 1.2 is established.

Single-step Behavioral Cloning (BC) is a straightforward approach to learning the policy: For this purpose, the trajectory dataset is played back in the simulator to obtain the observation vector of each vehicle at each timestep. Simultaneously, the action of each vehicle at each timestep is inferred by using an inverted kinematic model. With this, a dataset of observations and corresponding actions is constructed, and a standard supervised learning task is formulated to train a neural network that learns the mapping from observations to actions.

Due to its simplicity, single-step BC is commonly used in the literature for learning behavior models and therefore discussed as a baseline in Section 4.1. However, related works remark that a vehicle controlled by the learned policy tends to "slowly drift off the road" [Boj+16] and that the method is "insufficient for handling complex driving scenarios" [BKO19]. The reason for this is that the policy is typically trained on a dataset of regular driving, but makes small errors during the execution in the simulation loop. These errors accumulate until the policy is eventually confronted with observations that are so different from the training data that it is incapable of selecting a reasonable action. This phenomenon is investigated in this chapter and motivates the proposal of multi-step training in Section 4.2.

Multi-step training also is a supervised learning approach to obtaining a policy. Compared to single-step BC, the full trajectory is predicted during training, instead of only the next action. To predict the trajectory, the policy neural network is executed in the simulation loop. After the trajectory has been generated, the deviation to a corresponding ground truth trajectory is computed, and the gradient of this error signal is used to improve the policy.

While this core idea of multi-step training is simple, it entails the practical problem that the gradient of the error needs to be backpropagated through each component of the simulation

environment at each time step, which is shown in Section 4.2.1. This is likely the reason why no other works have proposed this approach earlier.

After introducing the single-step baseline and proposing multi-step training, behavior models are trained with both approaches and compared in Section 4.3.

4.1 Single-Step Behavioral Cloning

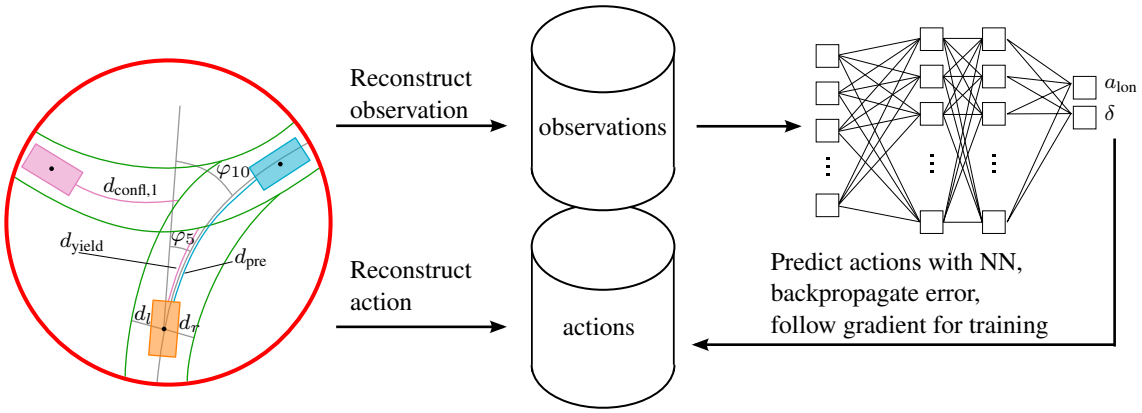
Behavioral Cloning (BC) is a straightforward approach to learning a behavior model. The fundamental idea is to train a regression model, typically an artificial neural network, to learn the relation between observations and corresponding actions from a recorded dataset. The first work in this direction by Pomerleau [Pom89] dates back to 1989, where the relation between a camera image and the travel direction is learned for an early automated vehicle. First, a dataset of camera images and corresponding travel directions is generated by a simulator. Next, a neural network is trained to predict the travel direction from the camera image. The learned policy is then deployed to demonstrate the learned ability of laterally controlling a test vehicle at $0.5\,\mathrm{m/s}$. Twenty-five years of advances in computing hardware, neural network architecture and sensors enable [Boj+16] to replicate Pomerleau's work, now demonstrating automated steering of a car on public roads and highways with few human interventions.

Besides these much acclaimed end-to-end approaches to controlling an automated vehicle, many works focus on the behavior modelling aspect, and exclude processing the raw sensor data from the model. To this end, it is assumed that an upstream perception module detects surrounding vehicles and estimates their state (position, speed, heading angle, shape). These high-level inputs, together with the ego state and a map, now form the input of the situation prediction module. As discussed in Section 2.6, the model input is a representation of the local environment of the target vehicle; it can be encoded as a feature vector, image, or graph. Many recent works follow this scheme to learn a driver behavior model: [MWK17] compares different neural network architectures to learn a longitudinal-only car following model; [WRK16; Len+17] train a combined longitudinal and lateral behavior model for highways, and [Sch+19; Ber+21] train models for urban situations.

4.1.1 Approach

The previously mentioned works follow the idea of BC to train the behavior model. The fundamental approach is illustrated in Figure 4.1. First the training dataset $\mathcal{D} = \{(o_1, a_1), (o_2, a_2), \dots\}$ of observations and corresponding actions is constructed with the simulator described in Chapter 3. Then, a neural network

$$\pi_\theta : O \to A,\ o \mapsto \hat{a}$$



(a) Construction of the observation and action dataset **(b)** Training of the policy neural network

Figure 4.1: Illustration of single-step behavior cloning: First (a), the observation dataset is constructed by playing back the recorded trajectories in the simulator and evaluating the observation vector for each vehicle at each time step. Simultaneously, the action at that time step is reconstructing by using the inverse kinematic model. Then (b), the policy neural network is trained on this dataset to predict the action that fits best to the corresponding observation.

that maps from the observation space O to the action space A is trained to predict the action $\hat{a}$. The goal of the training is to minimize the loss function

$$\ell(\theta) = \sum_{(o_k, a_k) \in \mathcal{D}} e_k^\top W e_k \tag{4.1}$$

with the per-sample error

$$e_k = \pi_\theta(o_k) - a_k \tag{4.2}$$

that evaluates the difference between the predicted and the actual action. If the action space has two dimensions (e.g., acceleration and steering), the weight matrix W compensates for the differences in scale of the components to ensure numeric stability, for example by setting it to the inverse standard deviation of the actions in the training data, $W = \Sigma_a^{-1}$.

The neural network training is performed by calculating the error gradient $\nabla \ell(\theta)$ using the backpropagation algorithm [RHW86] and adapting the neural network parameters

$$\theta_{n+1} \leftarrow \theta_n - \alpha \nabla \ell(\theta_n) \tag{4.3}$$

in the direction of the negative gradient iteratively until convergence with a small step width α. Commonly, more elaborated optimization schemes such as Adaptive Moment Estimation (Adam) [KB15] are employed to reduce the required number of iterations.

Instead of directly predicting an action, many related works predict a density over the action space, which enables them to quantify the aleatoric uncertainty [HW21] about the action, i.e., the uncertainty that remains about the behavior after the observation is known. The policy

$$\pi_\theta(\mathbf{a} = a|\mathbf{o} = o)$$

is now a conditional density. In practice, the density is characterized as a bivariate Gaussian [Sch+19], mixture of Gaussians [Len+17; MWK17], or piecewise uniform distribution [MWK17]. This is realized by implementing the policy neural network

$$f_{\text{NN},\theta} : o \mapsto (\mu_{a_{\text{lon}}}, \sigma_{a_{\text{lon}}}, \mu_\delta, \sigma_\delta) \tag{4.4}$$

such that it maps from an observation o to the parameters of the density, i.e., the mean and covariance matrix of a 2D Gaussian distribution of acceleration and steering. The training procedure remains unchanged, except for the loss function, which is now the negative log-likelihood

$$\ell_{\text{nll}}(\theta) = -\sum_{(o_k,a_k)\in\mathcal{D}} \log \pi_\theta(\mathbf{a} = a_k|\mathbf{o} = o_k). \tag{4.5}$$

Effectively, training with this loss function amounts to finding an observation-conditioned action density $\pi_\theta(\mathbf{a} = a|\mathbf{o} = o)$ where the likelihood of the ground truth actions is maximized by adapting the policy parameters θ. Given an observation, an action can be determined by drawing a random sample from the density, or by simply emitting the mean value.

4.1.2 Baseline Model

As a baseline model, this thesis implements such a BC model, π_{BC}. The model shall be employed in the simulation framework described in Chapter 3. To obtain the dataset of observations and corresponding actions, the trajectory dataset is played back in the simulator. While the standardized observations can directly be extracted, the longitudinal acceleration and steering action $a = (a_{\text{lon}}, \delta)^\top$ are reconstructed by inverting the kinematic bicycle model (3.3) to (3.5), as proposed by [Sch+19]:

$$a_{\text{lon},k} = \frac{v_{k+1} - v_{k-1}}{2\Delta t} \tag{4.6}$$

$$\delta_k = \arctan\left(\frac{(l_r + l_f)\Delta\psi_k}{\sqrt{v_k^2 - (l_r\Delta\psi_k)^2}}\right) \tag{4.7}$$

with the difference quotient of the heading angle

$$\Delta\psi_k = \frac{\psi_{k+1} - \psi_{k-1}}{2\Delta t} \tag{4.8}$$

The speed in the dataset is always at least $0\,\mathrm{m/s}$. The steering angle δ_k is undefined when $v_k \to 0$. It is set to 0 in this case. The term under the square root in (4.7) cannot become negative, as replacing the difference quotient $\Delta\psi_k$ with $\dot{\psi}$ from (3.3) shows. It becomes 0 at $|\sin\beta_k| \to 1$, but in practice, the slip angle $|\beta|$ is always much smaller than $90°$. The loss function (4.1) with $W = \mathrm{diag}(\sigma_{a_\mathrm{lon}}^2, \sigma_\delta^2)^{-1}$ is used. The detailed training setup is described in Section 4.3.1.

4.1.3 Discussion

The presented related works investigate different input representations, neural network architectures, and action spaces. However, fundamentally, all formulate the same problem of predicting the next action, given the current observation. This approach is susceptible to two pitfalls described below: causal misidentification and distributional shift.

Causal Misidentification Lenz et al. [Len+17] and Wheeler et al. [WRK16] make an interesting discovery: Both find that the most important feature for predicting the next action is the current action, which is a component of their observation vector. Including the last action as an input to the model seems intuitive to ensure that the sequential actions selected by the model are smooth.

However, succumbing to this intuition is hazardous! No regression algorithm is able to uncover causal relations on a fixed dataset of observations [Pea09], not even the most advanced neural network architecture. Instead, regression models only learn the correlation between input and output. In the case of [WRK16; Len+17], the models simply learn that the next action is the same as the last action, as successive actions are highly correlated. However, the cause of the behavior, which lies in the local environment of the vehicle, is ignored by the models. de Haan et al. [dHJL19] describe this phenomenon as "causal misidentification" and provide further examples.

Lenz et al. [Len+17] also investigate the effect of excluding the last action from the observation and remark that "Surprisingly, the feed forward model [...] without the last action performs best in the closed-loop simulation, although both NLL [the training loss] and intuition suggests otherwise". Clearly, without access to the last action, the model must have learned a behavior model that better reflects the causal relation between local traffic situation and action, and that therefore performs better in the closed-loop simulation.

Causal misidentification can principally not be ruled out in a supervised learning setting, but this basic pitfall can be avoided by excluding those variables from the input of the model that clearly have no causal effect on the output. This does not only include the past actions, but also variables that enable inferring the last action, e.g., the sequence of past velocities. This is the reason why the observation model (Table 3.1) in this thesis does not include past actions

or multiple past time steps, which would allow reconstructing past actions. While this protects the model from causal confusion, it implies that potentially relevant information cannot be used for the prediction. The consequences of leaving out this information are evaluated in the experiments section.

Distributional Shift Even models that do not seem to be affected by causal confusion suffer from the accumulation of errors during the execution of the policy. Schulz et al. [Sch+19] describe that the model cannot compensate for the errors that emerge when the model is deployed in a long-term simulation of more than $15\,\mathrm{s}$. Bojarski et al. [Boj+16] remark: "The network must learn how to recover from mistakes. Otherwise the car will slowly drift off the road." Similar issues are raised by [Pom89; BKO19], even with a vast amount of training data.

The underlying phenomenon is known as *distributional shift* [RB10; RGB11; dHJL19] or *covariate shift* [Bag15; Spe+21]: Standard supervised learning assumes that the distribution of the training data is the same as the distribution of the data at execution time. This assumption is violated when the model is executed in the simulation, because the selected actions influence the ensuing observations. For example, the model might select an action that steers the car towards the road boundary. As this new situation of driving close to the boundary was not represented in the training data, the BC model can select an adequate action only by chance and possibly steers the vehicle off the road.

The inability of the policy to extrapolate adequate actions beyond situations that are covered by the training data is a fundamental property of neural networks. Roughly speaking, neural networks are capable of interpolating for inputs that are close to its training data, but unable to extrapolate for inputs that are far beyond the region that is covered by training data. This is demonstrated for two simple functions in Figure 4.2. Without additional data or prior assumptions on the underlying function, the correct continuation of the functions cannot be learned, because any continuation seems equally plausible. The problem aggravates when the input space is high-dimensional, as exceeding the space covered by training data in any of the input dimensions can lead to erroneous extrapolation.

This illustrates that BC based on a dataset of successful driving alone is bound to fail. Possible solutions to the problem are: 1.) Synthetically generating training data that shows mistakes and corresponding corrections, as for example proposed by [Pom89; Boj+16; BKO19; Ber+21]. However, this requires knowing possible mistakes in advance and requires a lot of manual effort in generating realistic errors and corrections. 2.) Repeatedly executing the learned BC policy in a simulator, and querying an interactive expert for the correct actions to new observations that are experienced during this execution [RGB11]. These new pairs of observations and actions augment the training dataset and enable the training of an improved

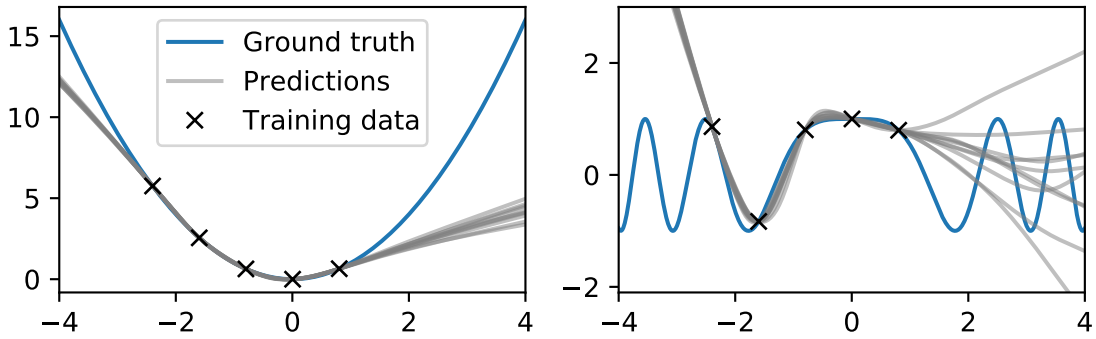


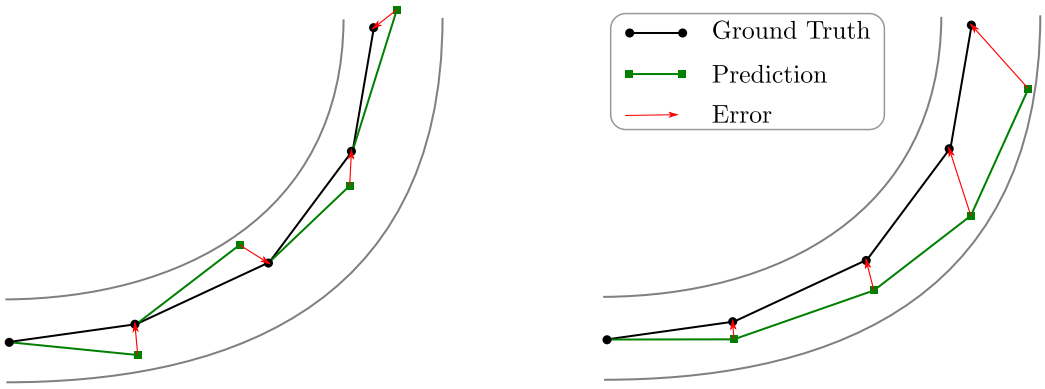
Figure 4.2: Demonstration of the ability of neural networks to interpolate between and extrapolate beyond their training data. The ground truth test functions are $y = x^2$ and $y = \cos(x^2)$. For each test function, 10 neural networks are trained based on the highlighted 5 data points. In both cases, the neural networks are able to approximate the test functions well close to the training data, but are unable to extrapolate beyond the training data. For this experiment, a multi layer perceptron with 1 input, 2 inner layers with 50 neurons and ELU nonlinearities, and 1 output neuron is used.

policy. This approach however requires the availability of an expert or an oracle that can provide correct actions for previously unseen observations.

Conclusion It is tempting to try to improve the single-step BC loss by experimenting with different input representations, neural network architectures, and training techniques. However, due to the phenomena of causal misidentification and distributional shift, the single-step BC error does not allow reliable conclusions to be drawn about the performance of a policy when executing it in a simulator. Thus, the focus of the following section and chapters lies on improving the policy performance with respect to causal misidentification and distributional shift.

4.2 Multi-Step Training

Instead of training a model to minimize the single-step error of predicting the next action, this thesis and the associated publications [**Sac+20b**; **Sac+21**] propose to minimize the long-term trajectory prediction error. This conceptual difference between single- and multi-step training is illustrated in Figure 4.3. By minimizing the long-term error, the policy is forced to learn to compensate for poor actions, i.e., handle distributional shift. Moreover, only a model that correctly identifies the causal relations between observation and action is able to achieve low long-term prediction errors.



(a) Single-step training **(b)** Multi-step training

Figure 4.3: Illustration of the difference between single- and multi-step training. During single-step training, the next action is predicted for each observation along the trajectory, and the error is computed. The goal of the training is to minimize this error. In contrast, multi-step training predicts the full trajectory and minimizes the deviation between the predicted and the ground truth positions at every time step. Hence, while single-step training minimizes the prediction error over one simulation step, multi-step training directly minimizes the long-term prediction error over multiple simulation steps.

4.2.1 The Need for a Differentiable Simulation

Despite the conceptual simplicity of the idea, there is one major challenge in its implementation: To train a model that minimizes the long-term prediction error, the policy must be executed in the simulation environment during training. For a gradient-descent style minimization of the trajectory prediction error, the gradient of this error with respect to the policy parameters needs to be computed. It follows that the gradient of every component of the simulation model must be computable, as the following calculation of the derivative reveals.

Suppose the simulation of one agent j is composed of three deterministic functions, as described in Chapter 3: First, the kinematic model κ,

$$y_{k+1}^j = \kappa(y_k^j, a_k^j), \tag{4.9}$$

that determines the next state y_{k+1}^j based on the current state and action a_k^j. Second, the observation model λ,

$$o_k^j = \lambda(y_k^j, W_k^j), \tag{4.10}$$

which determines the local observation vector o_k^j of the agent as listed in Table 3.1 based on the state of the agent y_k^j and the surrounding world W_k^j, which subsumes all information on surrounding vehicles and the map. Third, the policy neural network π_θ,

$$a_k^j = \pi(o_k^j, \theta), \tag{4.11}$$

which determines an action a_k based on the observation. For clarity, the neural network parameters θ are written here as an explicit input to the policy function π. Based on these three functions, the prediction y_k^j at time step k is built up sequentially, by applying (4.9) to (4.11) to first compute y_1^j, then y_2^j, and so on, starting from the initial state $y_0^j = x_0^j$. Thus, all variables a_k^j, o_k^j, y_k^j are dependent on θ, which is not explicitly stated in the following.

Let

$$\text{dist}(x_k^j, y_k^j) = ||\text{pos}(x_k^j) - \text{pos}(y_k^j)||_2 \tag{4.12}$$

evaluate the displacement between the predicted position $\text{pos}(y_k^j)$ and the ground truth position $\text{pos}(x_k^j)$ in meters. The displacement is penalized via the quadratic loss function

$$f_2(x) = x^2. \tag{4.13}$$

For training, the gradient of the loss with respect to the policy parameters θ needs to be determined:

$$\frac{\partial}{\partial\theta} f_2(\text{dist}(x_k^j, y_k^j)) = 2\ \text{dist}(x_k^j, y_k^j) \frac{\partial}{\partial\theta} \text{dist}(x_k^j, y_k^j) \tag{4.14}$$

with

$$\frac{\partial}{\partial\theta} \text{dist}(x_k^j, y_k^j) = \frac{\partial \text{dist}(x_k^j, y_k^j)}{\partial x_k^j} \frac{\partial x_k^j}{\partial\theta} + \frac{\partial \text{dist}(x_k^j, y_k^j)}{\partial y_k^j} \frac{\partial y_k^j}{\partial\theta} \tag{4.15}$$

Hereby, the chain rule of multi-variable calculus is applied. The resulting first term can be omitted, because the ground truth is not influenced by the policy, i.e., $\partial x_k^j / \partial\theta = 0$. The second term is further expanded, again, by applying the chain rule,

$$\frac{\partial y_k^j}{\partial\theta} = \frac{\partial\kappa(y_{k-1}^j, a_{k-1}^j)}{\partial y_{k-1}^j} \frac{\partial y_{k-1}^j}{\partial\theta} + \frac{\partial\kappa(y_{k-1}^j, a_{k-1}^j)}{\partial a_{k-1}^j} \frac{\partial a_{k-1}^j}{\partial\theta}. \tag{4.16}$$

Thus, the kinematic model κ must be differentiable with respect to both of its inputs y_{k-1}^j and a_{k-1}^j. The recurrence relation in the first summand makes clear that all partial derivatives $(\partial y_{k-1}^j/\partial\theta, \partial y_{k-2}^j/\partial\theta, \ldots, \partial y_1^j/\partial\theta)$ need to be calculated to evaluate $\partial y_k^j/\partial\theta$. Further, the newly occurring partial derivative

$$\frac{\partial a_{k-1}^j}{\partial\theta} = \frac{\partial\pi(o_{k-1}^j, \theta)}{\partial o_{k-1}^j} \frac{\partial o_{k-1}^j}{\partial\theta} + \frac{\partial\pi(o_{k-1}^j, \theta)}{\partial\theta} \tag{4.17}$$

with

$$\frac{\partial o_{k-1}^j}{\partial\theta} = \frac{\partial\lambda(y_{k-1}^j, W_{k-1}^j)}{\partial y_{k-1}^j} \frac{\partial y_{k-1}^j}{\partial\theta} + \frac{\partial\lambda(y_{k-1}^j, W_{k-1}^j)}{\partial W_{k-1}^j} \frac{\partial W_{k-1}^j}{\partial\theta} \tag{4.18}$$

shows that also the observation model λ must be differentiable with respect to the predicted vehicle state y_{k-1}^j. At this point, again, a recurrence relation of $\partial y_k^j/\partial\theta$ to all past derivatives emerges when (4.18) is inserted into (4.17) and then (4.16). The second term of (4.18) can

be omitted, because only the trajectory of one vehicle at a time is optimized, whereas the surrounding vehicles are played back from a recording, hence W_{k-1}^j is not influenced by the policy parameters and $\partial W_{k-1}^j / \partial \theta = 0$.

This shows that all parts of the simulator need to be differentiable with respect to their inputs. Thus, none of the popular off-the-shelf simulators such as Carla [Dos17] or Sumo [Lop+18] can be used for executing the simulation loop during training. To avoid the need to manually implement the gradient computations (4.14) to (4.18), the simulation is implemented in the automatic differentiation framework Pytorch [Pas+19]. After running the simulation, Pytorch can directly calculate the gradient of the loss function $\partial f_2(\text{dist}(x_k^j, y_k^j)) / \partial \theta$.

At this point, also the difference to single-step BC becomes clear: By assuming the past states and observations to be fixed, and not a consequence of the past policy execution, i.e., assuming $\partial y_{k-1}^j / \partial \theta = 0$ in (4.16) and $\partial o_{k-1}^j / \partial \theta = 0$ in (4.17), the gradient of the actions with respect to the policy parameters can be simplified to the gradient of the policy with respect to its parameters and no recurrence relation emerges. Thus, as the name suggests, single-step BC can be seen as a special case of multi-step training. In practice, one minor difference exists: multi-step training for one single step predicts the next position by predicting an action and applying the kinematic model. In contrast, single-step training as described in the previous section directly predicts the next action, with the ground truth generated by inverting the kinematic model. Functionally, both leads to the same result.

Relation to RNNs The gradient computation (4.15) to (4.18) is related to the backpropagation through time algorithm [Zha+22, Ch. 9.7], which computes the gradients of RNNs. Similar to the simulation model (4.9) to (4.11), RNNs are also recurrently executed to make predictions of sequential data. The central difference to RNNs is that multi-step training imposes more structure on the prediction problem, as it integrates a kinematic model and limits the information available to the predictor by the pre-defined observation vector. These interfaces with physical meaning allow the interpretation of any intermediate state as the kinematic state of the vehicle at that point in time. In contrast, there is no straightforward way to meaningfully interpret the intermediate states of a RNN. Moreover, the current state of a vehicle regulates which observation is made next, thereby introducing a feedback loop that is not part of regular RNNs. The difference between the multi-step structure and a regular RNN is illustrated in Figures 4.4 and 4.5. Zyner et al. [ZWN20] propose a RNN-based trajectory prediction, but ignore the influence of surrounding vehicles on the predicted vehicle.

4.2.2 Training

Once the simulation is implemented in a differentiable way, training a policy that minimizes the deviations between actual and predicted position is simple. The policy is learned from

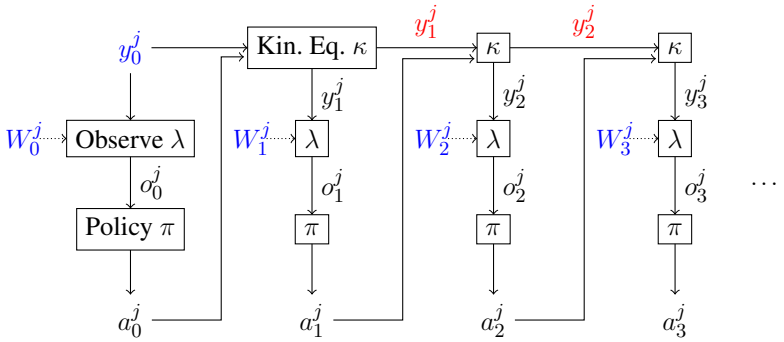


Figure 4.4: Unfolded simulation loop during multi-step training. Inputs are blue and outputs are red. Based on the initial vehicle state y_0^j and the world state W_0^j, the observation model λ generates the observation o_0^j. With this, the policy π determines the action a_0^j. The current state and action are processed by the kinematic model κ, which yields the predicted state y_1^j. Then, these steps are repeated until the desired number of prediction steps are reached. Thanks to this structure, all intermediate variables are interpretable as observation, action, and kinematic state. Figure adapted from [**Sac+21**], © 2021 IEEE.

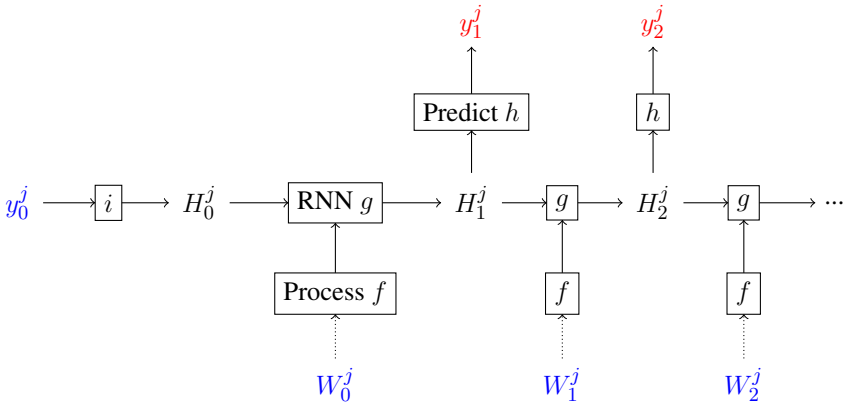


Figure 4.5: Concept of a functionally similar RNN, composed of four different neural networks f, g, h, i: The initial hidden state H_0^j is generated by the neural network i from the initial vehicle state y_0^j. In each step, the recurrent network g emits the next hidden state H_{k+1}^j based on the previous hidden state and the processed world state $f(W_k^j)$. The prediction of the vehicle state y_k^j is performed by the network h based on the current hidden state H_k^j. While it is conceivable that this structure learns something equivalent to a kinematic model for the prediction, there is no regulation on which part of the world state is relevant at which time step. In contrast, the observation model λ in Figure 4.4 always emits an observation that precisely describes the relation of the vehicle to the surrounding world through the features listed in Table 3.1. A standard RNN lacks the necessary information about the relevance of different parts of the world state W_k^j, which can lead to causal misidentification and poor predictive ability.

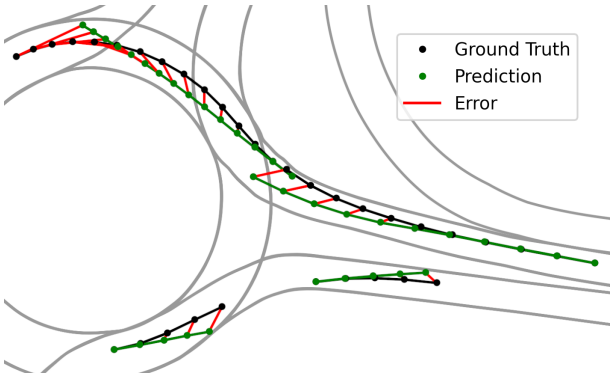


Figure 4.6: Multi-Step Training: During early training, the predicted and ground truth trajectories differ significantly. The training minimizes the pointwise deviations between the trajectories, depicted in red, by adapting the policy parameters. To this end, the loss (4.19) is defined as the sum of all of these squared errors. Predictions are terminated after H steps, or when they leave the road. As the error gradient attracts them towards the ground truth, they should remain longer on the track after the next training iteration. Figure from [**Sac+21**], © 2021 IEEE.

a dataset $\mathcal{D} = \{\tau_1, \tau_2, \dots\}$ of H-step ground truth trajectories of individual agents and their surrounding world, $\tau_j = ((x_0^j, W_0^j), \dots, (x_H^j, W_H^j))$. For each trajectory, the simulation (4.9) to (4.11) is executed from the initial state $y_0^j = x_0^j$ with the current policy π_{θ_n} to obtain the corresponding predicted trajectory $\hat{\tau}_j = (y_1^j, y_2^j, \dots, y_H^j)$. During training, only the focal agent j is predicted using the policy, whereas the trajectories of the surrounding vehicles are played back from the ground truth data; they are part of W_k^j and therefore not dependent on the policy parameters θ.

After making the predictions, the weighted loss

$$\ell_{\text{traj}}(\theta_n) = \frac{1}{|\mathcal{D}|} \sum_{j=1}^{|\mathcal{D}|} \omega_j \sum_{k=1}^{H} f_2(\text{dist}(x_k^j, y_k^j)) \tag{4.19}$$

is evaluated and its gradient with respect to the current policy parameters θ_n is determined by automatic differentiation. The loss is visualized in Figure 4.6. All samples have a weight of $\omega_j = 1$, unless otherwise stated.

The policy parameters are adapted according to the gradient of the loss iteratively using

$$\theta_{n+1} \leftarrow \theta_n - \alpha \nabla \ell_{\text{traj}}(\theta_n) \tag{4.20}$$

or any other improved gradient descent scheme; in this work, Adam [KB15] is employed.

Even for good behavior policies, large prediction errors arise occasionally due to the complex situation dynamics. Consider a situation in which a vehicle is close to the roundabout entry in an ambiguous situation. While the ground truth vehicle enters the roundabout swiftly,

the policy predicts the vehicle to stop at the entry to let another vehicle pass. The predicted vehicle then needs to let multiple other vehicles pass before it can enter the roundabout. As a consequence, a large prediction error arises. This error is not the result of a random Gaussian deviation of the prediction, but rather of the inherent multimodality of the roundabout entry situation. Minimizing the quadratic loss $f_2(x) = x^2$ in (4.19) however assumes that the underlying error follows a Gaussian distribution. A more robust loss function

$$f_H(x) = \begin{cases} \frac{1}{2}x^2, & \text{if } |x| < \nu \\ \nu|x| - \frac{1}{2}\nu^2, & \text{if } |x| \geq \nu \end{cases} \tag{4.21}$$

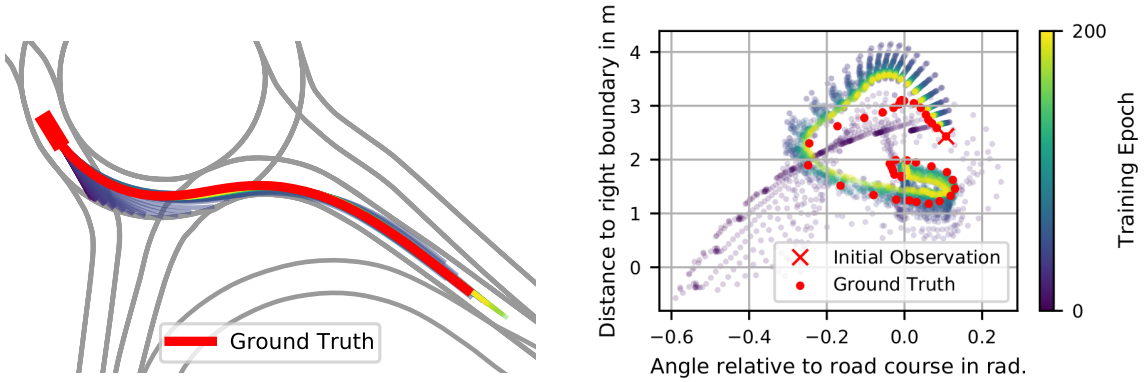
is proposed by Huber [Hub64]. Compared to a purely quadratic loss function, the influence of large errors on the loss is reduced, because it increases linearly after passing a threshold ν. As a result, the gradient of large errors

$$\frac{\mathrm{d}f_H(x)}{\mathrm{d}x} = \begin{cases} x & \text{if } |x| < \nu \\ \operatorname{sign}(x)\nu & \text{if } |x| \geq \nu \end{cases} \tag{4.22}$$

is limited to a maximum value of $|\nu|$, i.e., the influence of individual predictions is limited during gradient descent in (4.14). For this reason, the Huber-loss f_H replaces the quadratic loss f_2 in (4.19) for the experiments with a value of $\nu = 10[\mathrm{m}]$.

Especially during early training, many predictions leave the track. Then, the assignment to a road segment can become ambiguous. To prevent ill-defined observations from hampering the training, the simulation of an agent is terminated early in this case. Correspondingly, the loss of that agent cannot be evaluated until the final time step H, but only to an earlier time step $G < H$. For this agent, the sum (4.19) needs to be truncated at G. This leads to a counterintuitive phenomenon: Predictions that are terminated early often have a lower loss than those that are predicted for the full horizon, because the last trajectory steps, which typically have the highest prediction error, are excluded from the loss sum. Thus, a policy that swiftly leaves the track from any starting point might have a lower loss than a policy that remains on the track. However, as a gradient-based optimization (4.20) is used, the predicted trajectories are effectively attracted towards the ground truth trajectories, as illustrated in Figure 4.6. The absolute loss value is therefore not a meaningful instrument to compare different training runs, but rather a tool to enable the gradient computation. The loss is also truncated for predictions that are involved in a collision. To stimulate the policy to avoid collisions and leaving the road, these truncated samples have an increased weight of $\omega_j = 3$ in the sum (4.19).

Multi-step training can also be interpreted differently: During training, the set of observations that the model is confronted with is augmented, which alleviates the issue of distributional shift visualized in Figure 4.2. While a dataset of regular driving only rarely contains situations



(a) Training progress in trajectory space

(b) Observations made during multi-step training

Figure 4.7: Training progress of one single vehicle during multi-step training. **(a)** During early training (dark blue), the vehicle quickly leaves the track. Later (green-yellow), it learns to stay on the track and travels approximately the same distance as the ground truth vehicle. **(b)** During training, a single-step model is only faced with observations from the ground truth data (red). The multi-step model experiences all colored points in the observation space during its training. For example, while the single-step learner never observes distances to the road boundary smaller than $1\,\mathrm{m}$, these observations are regularly made by the multi-step learner. This allows the model to learn how to react in these situations to avoid leaving the road. For illustration purposes, only two dimensions of the high-dimensional feature vector in Table 3.1 are visualized.

in which a vehicle is rapidly approaching the road boundary, this situation is regularly experienced during the early phase of multi-step training, as visualized in Figure 4.7. This increases the ability of the learned models to correct for mistakes, compared to single-step training.

4.2.3 Related Works

Concurrently to the associated publications [**Sac+20b**; **Sac+21**], published in 2020 and 2021, three other works [Suo+21; Ści+21; Sch+22] independently proposed to train a driving policy by minimizing the multi-step trajectory prediction error by employing a differentiable simulation in 2021 and 2022. All find that the trained policies are significantly more robust than the single-step BC approach. However, the works differ significantly in their goals and implementation:

Behavior Modelling With the goal of multi-agent trajectory prediction, Ścibior et al. [Ści+21] propose to execute a simulation loop where each vehicle is controlled by the same policy, similar to the procedure described in Chapter 3. In contrast to this thesis, the authors use an image-based representation as an input to the behavior policy. The image is generated by rasterizing a birds-eye-view of the map and surrounding vehicles of each agent at every

simulation step. The policy is realized as a CNN that uses this image as input to determine the next action. Finally, a kinematic bicycle model is used for determining the next state. As illustrated in the previous section, the observation model needs to be differentiable for multi-step training. Thus, the rasterization procedure is implemented using a differentiable rendering library, leading to large memory consumption and a reported training time of 6 weeks on a data center Graphics Processing Unit (GPU).

Suo et al. [Suo+21] have the goal of building a realistic multi-agent driving simulation. They propose to use a joint driving policy, which receives as input the current and past states of all agents and predicts their future trajectories jointly. As one central policy controls all agents, actions can be selected coordinately. This contrasts the approach proposed in this thesis, which predicts the actions of each agent individually, based on their observation. This modeling is preferred, because it more accurately reflects the decision process of human drivers, who also individually decide on their actions, instead of selecting them jointly. One reason why a joint policy is employed in [Suo+21] is that the model predicts full trajectories at each simulation step instead of the immediate next action, which requires more coordination between the agents. When employing the policy in the simulation, predicting full trajectories enables a trade-off between execution time and precision: Either only the first step of each trajectory is executed before making a new prediction in the next simulation step, or multiple steps are executed at once.

Scheel et al. [Sch+22] use multi-step training to learn a driving policy with the goal of directly controlling an automated vehicle in real-world traffic. To do so, similar to this thesis, a policy is learned by executing it in a differentiable simulation with the goal of matching recorded ground-truth trajectories. Their central result is that a policy trained using a differentiable simulator not only imitates driver behavior better than naïve BC and enhanced variants, but also requires the least interventions when applying the policy in real-world traffic.

Minimizing the multi-step prediction loss (4.19) is not restricted to the training of neural network policies. For example, in [**Sac+20b**], it is also employed for the calibration of a parametric car-following model, the IDM [THH00]. This idea is not new and has also been explored by [PS05; TK13a]. The fundamental difference to these approaches is that neural network behavior policies typically have multiple orders more parameters than parametric car-following models such as the IDM, which has 5 parameters. For this reason, car-following models can be calibrated using gradient-free optimization algorithms, which does not require a differentiable simulation. Efficient neural network training, on the other hand, is always implemented as a gradient descent optimization scheme [GBC16, Ch. 8], and therefore multi-step training requires a differentiable simulation to calculate the gradient of the loss function with respect to the network parameters.

Differentiable Simulation in Other Domains The idea of using a differentiable simulation, typically implemented in an automatic differentiation framework such as PyTorch, has been explored in many other contexts. This is a broad field, such that a comprehensive overview is beyond the scope of this thesis. Instead, only some interesting applications are listed in the following.

One early example originates from the control theory problem associated with driving a truck with a trailer in reverse to a target location [NW89]. The input of the neural network controller are six state variables that describe the kinematic state of the truck-trailer combination. During training, the trajectory of the truck is simulated by executing the combination of the neural network controller and the kinematic state transition for multiple steps, starting from different initial positions. Finally, the distance from the trailer to the target location is used as an error signal. This error is backpropagated to adapt the weights of the neural network controller during training. The authors demonstrate that the controller learns to steer the truck even for adverse initial conditions.

Bächer et al. [BKS21] focus on the field of soft robotics. This is an especially challenging field, because the dynamics of soft systems are highly nonlinear and hard to characterize. The authors list three applications of differentiable simulations: First, a differentiable simulation can be used to adapt simulation parameters to better characterize a real world system. For example, it can be learned how a material deforms when a force it applied, and this function can then be used for a simulation of that material. Secondly, differentiable simulations can be used for actuation tasks, to learn a controller, similar to the work on controlling a reverse driving truck. The third application that is mentioned is state estimation: This can be formulated as an optimization problem with the goal of finding the state variables that minimize the deviation between actual and expected measurements. By using a differentiable simulation environment, a gradient-following optimization method can be employed to find the state variables that most accurately explain the measurements.

In the field of physical simulations, Um et al. [Um+20] investigate the potential of integrating a machine-learned correction model into a simulation loop to compensate for numerical errors in the evaluation of a discretized Partial Differential Equation (PDE) solver. The idea is demonstrated to improve the simulation accuracy for different applications such as the simulation of turbulences and the simulation of combined advection and diffusion.

Another interesting application of differentiable simulations in biology is the protein folding problem, where the goal is to predict the three-dimensional structure of a protein, based on its amino acid sequence. For this task, Ingraham et al. [Ing+19] propose a differentiable simulation with a learnable energy field function. The predicted protein structure is shaped by this energy field, as the structure unfolds to minimize the free energy. This folding can be stimulated step by step, and the difference between the predicted structure and the actual structure is backpropagated through the simulation to learn the energy field function.

4.3 Experiments

This section demonstrates how a policy is learned with single- and multi-step training. To this end, both approaches are trained under equal conditions—equal training trajectories, network architecture, observation vector and evaluation situations. Single-step training is executed with and without access to the last observation to demonstrate the phenomenon of causal misidentification. Multi-step training is executed with trajectories of different lengths to investigate their influence on the quality of the learned model.

The central questions that will be investigated in this experiments section are:

- Can a policy neural network learn to accurately predict the next action, given the current observation, with single-step training?
- Is a policy learned with single-step training capable of controlling all agents successfully in a simulated traffic situation?
- Can it be demonstrated that single-step training suffers from distributional shift and causal misidentification?
- Does multi-step training reduce the trajectory prediction error, compared to policies learned with single-step training?
- Is multi-step training robust to distributional shift and causal misidentification?
- In multi-step training, is training for more steps advantageous compared to training with shorter trajectories?

4.3.1 Single-Step Training

As a baseline, a single-step model is trained. The dataset used for training and validation is described in Section 3.3. The trajectories in both datasets are resampled to a rate of $5\,\mathrm{Hz}$ to reduce correlation between consecutive trajectory points, thereby eliminating redundant elements in the training set. The data is played back in the simulator to reconstruct the observations and actions along the trajectories. The training dataset contains a total of 60,169 observations and corresponding actions. The validation dataset contains 11,694 items. The training and validation data are extracted from different recordings, such that it is ruled out that training and validation samples come from the same trajectories.

A standard fully connected neural network is employed to learn the policy. Given the observation vector, it predicts the corresponding acceleration and steering angle. The observation vector (Table 3.1) with 22 features determines the number of inputs. Empirically, two fully connected layers of 50 neurons each are sufficient to learn a good policy. The last layer has two outputs, the acceleration and steering action. After each layer, a $\tanh$ nonlinearity is used. A diagram of the architecture is shown in Figure C.1. To reduce overfitting, dropout [Sri+14] with a probability of 0.2 is used after the two fully connected layers during training.

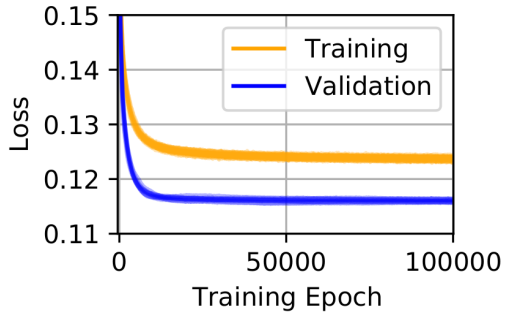


Figure 4.8: Training and validation loss during seven repetitions of single-step behavioral cloning training. The progress is very similar, such that all curves overlap. The validation loss is lower than the training loss, because dropout is only applied during training.

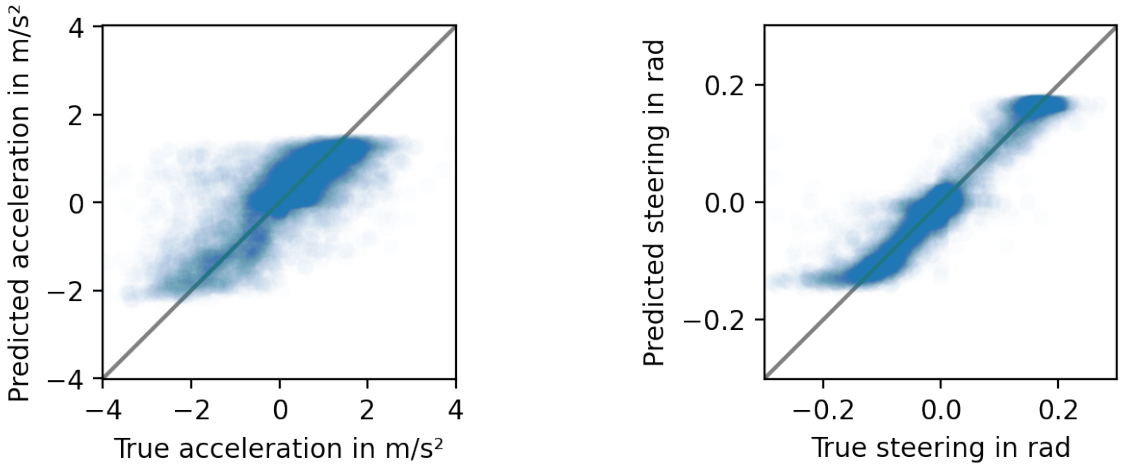


Figure 4.9: Ground truth action and prediction by the best learned model. Data points are plotted transparently to emphasize the clusters in the data.

Briefly, this means that the output of each neuron is set to 0 with that probability. In effect, the neurons in the following layer learn to not rely too much on a specific neuron, and are forced find redundant representations of the input-output relation. Empirically, this technique leads to networks that are less prone to overfitting and improves predictions on unseen test data significantly [Sri+14]. During evaluation and testing, dropout is disabled. The model is trained with the Adam optimizer [KB15] with the default parameters suggested in the paper, listed in Appendix C.2.

Training the model for 100,000 epochs takes approximately 5 minutes on a NVIDIA RTX 2060. The training is repeated seven times to ensure reproducibility. The loss curves are shown in Figure 4.8. All repetitions converge to similar values for the validation loss after approximately 20,000 training epochs. The model with the lowest validation loss is used for all further evaluations.

To visually evaluate the quality of the learned model, Figure 4.9 contrasts the predicted actions with the actual actions for the validation dataset. The ideal result would be that all predictions perfectly match the ground truth and therefore fall on the gray diagonal. While the plots show that most predictions are close to the ground truth, it also reveals that the prediction problem is ambiguous: For most observations, many different accelerations and steering angles are feasible. This leads to the points where the prediction and the ground truth disagree, most notably for the accelerations. One further reason for the prediction errors is the relatively large standard deviation of the error of the ground truth acceleration ($\sigma_a \approx 0.26\,\mathrm{m/s^2}$, c.f. Appendix A), which is inherent to the acceleration as a quantity typically measured as a first or second order derivative of an original sensor measurement.

To demonstrate the learned behavior, the policy is executed in a test situation. The underlying simulation is executed with a step width of $\Delta t = 0.2\,\mathrm{s}$; the temporal sequence is shown in Figure 4.10. For a better overview, only every seventh step is plotted. For this and all following experiments, it is assumed that the route that a vehicle takes through the roundabout is known a priori, i.e., whether a vehicle takes the first, second or third exit. This is dictated by the policy, which acts goal-directed and expects observations that describe the future road course, see Table 3.1. In a real-world prediction system, this information is not known prior to the prediction. Instead, one prediction would need to be issued for every possible junction along the road, and the probability of each prediction would need to be estimated online. A practical approach to this by Schulz et al. [Sch+18b; Sch+18c] models the problem as a dynamic Bayesian network and uses a particle filter or multiple-model unscented Kalman filter to estimate the node probabilities. As the focus of this work lies on learning improved behavior models, this is not implemented here and the following evaluations can be considered an evaluation of the best-matching hypothesis. A quantitative evaluation of the policy performance in the simulation is shown jointly with the multi-step models in Section 4.3.3.

Action Feedback To illustrate the phenomenon of causal misidentification, another single-step policy is trained with the last acceleration and steering angle as additional observations. Everything else remains unchanged. Compared to the model that has no access to these observations, this policy is significantly more successful in predicting the next action, as Figure 4.11 illustrates in comparison to Figure 4.9. However, the additional information interferes with the model learning the causal relation between observations and actions. Instead, the model relies strongly on the last actions to predict the next actions. As a consequence, many vehicles leave the track or collide when executing the policy in a closed loop simulation, depicted in Figure 4.12. On the other hand, the model is not exclusively replicating the last action and seems to have learned at least some degree of causal relation, as some vehicles manage to successfully follow the road and enter the roundabout. Still,

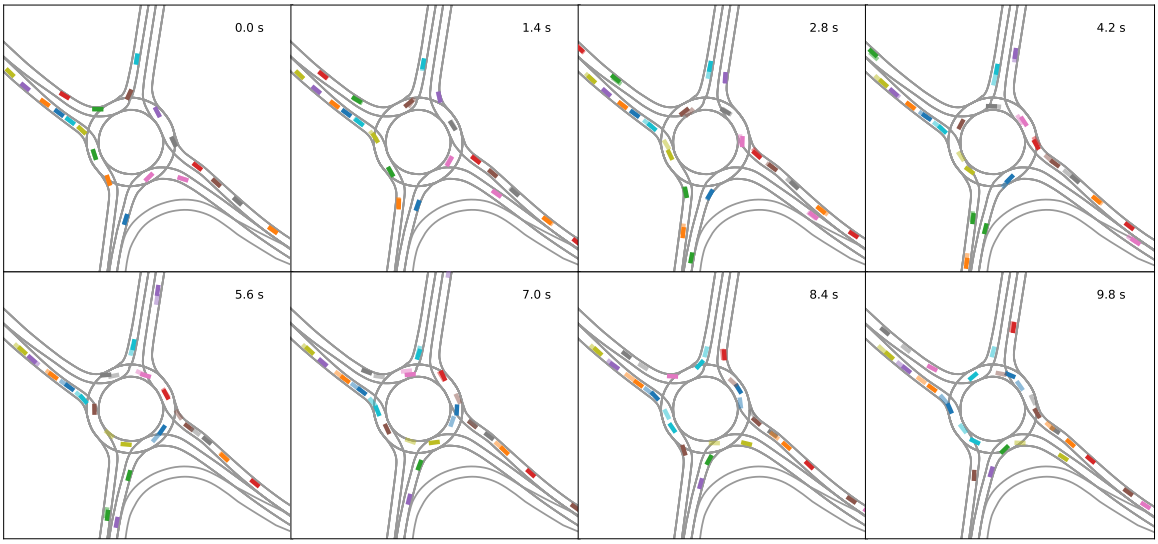


Figure 4.10: Execution of the single-step model in closed loop, i.e., all vehicles are controlled by the learned policy and see and interact with the other predicted vehicles. Vehicles from the predicted situation are shown transparently, whereas the corresponding vehicles from the ground truth situation are colored solidly. The situation is from the test dataset and has not been used for training. Only vehicles present in the initial situation (top left) are predicted. Clearly, the policy has learned to stay on the track in most cases and can handle stop-and-go, right-of-way and unhindered driving situations. For many vehicles, the prediction is accurate, especially when they are constrained by surrounding vehicles. After $5.6\,\mathrm{s}$, the predicted situation starts to diverge from the ground truth situation: While the brown vehicle coming from east is predicted to enter the roundabout, the corresponding ground truth vehicle waits at the entry to let the blue vehicle pass. This has a knock-on effect on all following vehicles from east, whose predictions can now enter the roundabout significantly earlier than their ground truth counterparts. Apart from that, the prediction of the brown vehicle coming from north leaves the track after $2.8\,\mathrm{s}$ and the pink vehicle leaving the roundabout towards west drives off the track after $7\,\mathrm{s}$.

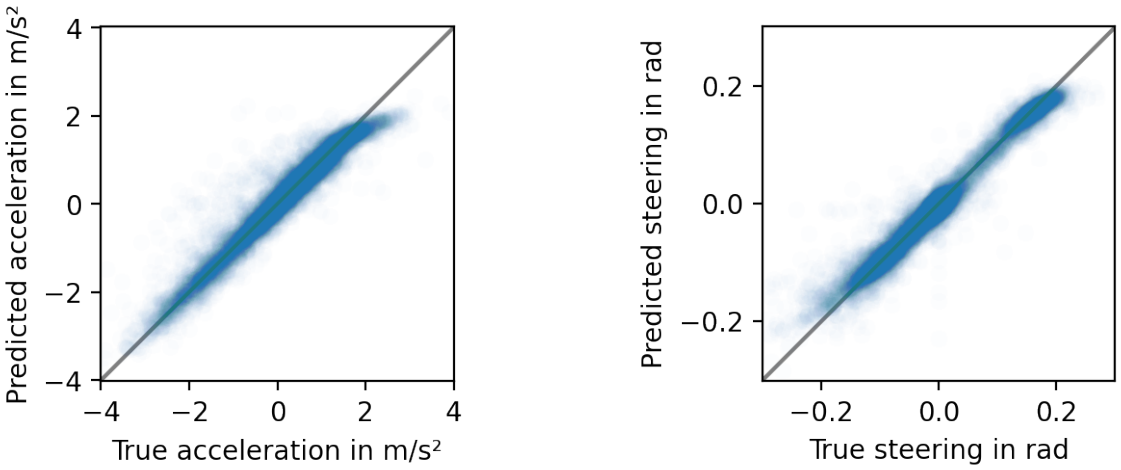


Figure 4.11: Ground truth action and prediction by the policy that can observe the last acceleration and steering angle. Compared to the model without access to this information in Figure 4.9, the prediction of the next action is significantly better. In spite of this, executing the policy in Figure 4.12 reveals that it is incapable of controlling vehicles in closed loop, steering the majority of them off the track in a $10\,\mathrm{s}$ simulation.

this experiment demonstrates that including the last action in the observation vector leads to undesired consequences when training a single-step model.

4.3.2 Multi-Step Training

Next, different models are trained using multi-step training as described in Section 4.2.2. The central parameter that is investigated in this section is the length of the trajectories that are used for training. Up to which point can a decreased prediction error justify the increased training time when using longer trajectories?

To investigate this question, trajectory segments of different lengths are extracted from the training and validation datasets. Training models with long multi-step trajectories can lead to the phenomenon of vanishing or exploding gradients, which is well known from the training of deep neural networks and recurrent neural networks [Zha+22, Ch. 5.4 and 9.7]. To avoid this and to reduce the training time, the trajectories should consist of as few steps as possible. For this reason, all trajectories are resampled to $1/\Delta t = 5\,\mathrm{Hz}$. This reduces the number of simulation steps required to represent a trajectory by a factor of 6 compared to the original sampling rate of approximately $30\,\mathrm{Hz}$. The resulting time step length of $\Delta t = 0.2\,\mathrm{s}$ can be interpreted as the effective reaction time of the policy. As it is lower than the human reaction time, which is estimated to be between $0.7\,\mathrm{s}$ and $1.5\,\mathrm{s}$ in braking situations [Gre00], it should suffice to successfully imitate human behavior.

The trajectories used for training are $1\,\mathrm{s}$, $2\,\mathrm{s}$, $4\,\mathrm{s}$, $8\,\mathrm{s}$ and $16\,\mathrm{s}$ long, which corresponds to 5 to 80 simulation steps. To obtain trajectories of the corresponding lengths, the original

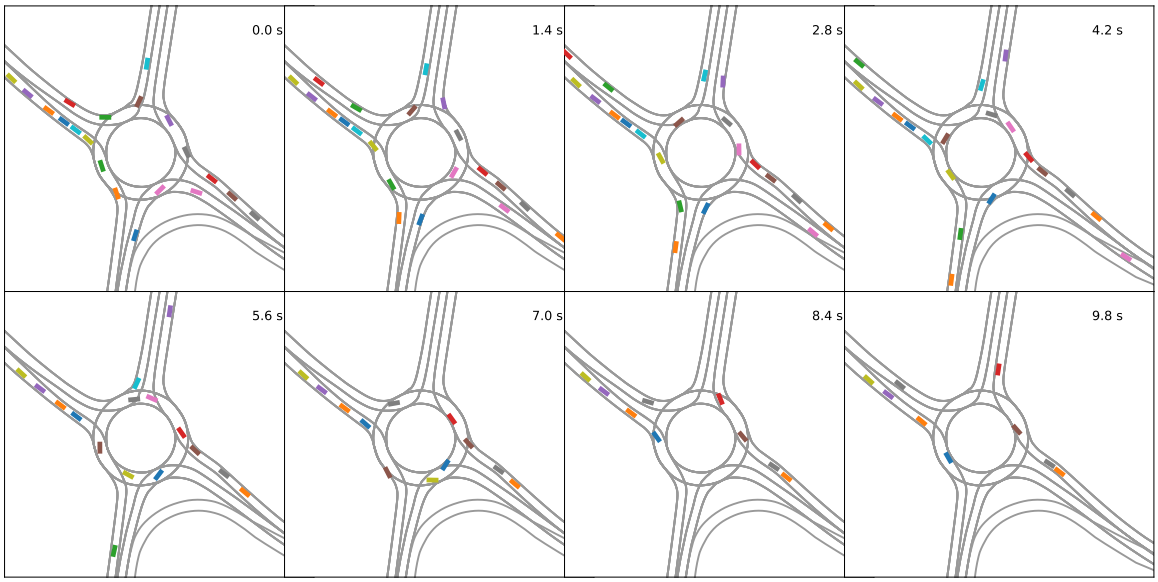


Figure 4.12: Execution of the single-step model with access to the last action in closed loop: The model mainly bases its prediction on the last action and predicts vehicles to keep an approximately constant acceleration and steering angle. Thus, many vehicles leave the track or collide. The ground truth evolution of the situation is omitted in this figure, as the model is clearly incapable of making plausible predictions.

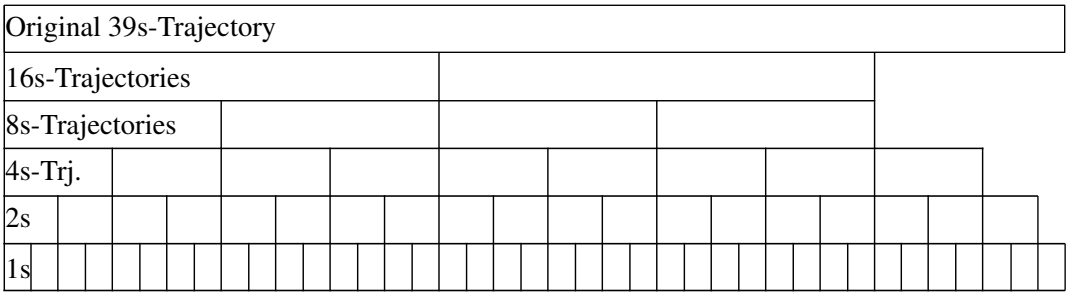


Figure 4.13: Slicing of trajectories: The original trajectory is sliced into as many non-overlapping shorter segments as possible.

trajectories are sliced into non-overlapping segments. This slicing leads to a decreased dataset size for longer prediction horizons. With every doubling of the trajectory length, the number of trajectories is cut in half, as shown in Figure 4.13. For example, The original 60,000 trajectory steps are sliced into approximately 11,300 $1\,\mathrm{s}$-trajectories composed of 5 steps, 6,000 $2\,\mathrm{s}$-trajectories composed of 10 steps, and so on. In each training situation, one vehicle is controlled by the policy, while the trajectories of all surrounding vehicles are played back from the data.

To maintain comparability, the same neural network architecture as for single-step training is used. Dropout is not applied, because it introduces noise to the actions, which amplifies after multiple prediction steps and therefore destabilizes the training. For the optimization, Adam [KB15] is used with default parameters.

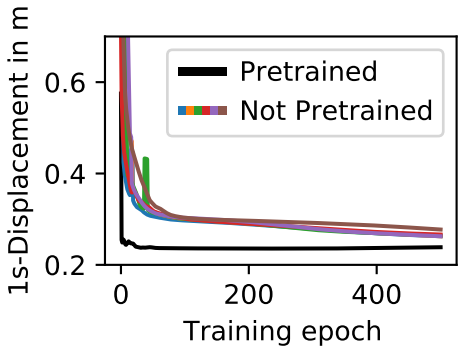


Figure 4.14: Displacement between 1 s-predictions and ground truth during training. The pretrained model reaches a low displacement after few epochs, whereas the other models do not achieve similar performance within 500 training epochs.

Despite using the Huber-loss, directly training policies for prediction horizons above 4 s from scratch is instable due to the large positional errors and gradients that occur. Often, the resulting policies fail to learn to stay on the track. For this reason, the training is executed consecutively: First, the training of the 1 s-model is performed. Instead of using random initial weights for the model, the parameters from the single-step model are used as a basis. These provide a reasonable initialization and thereby reduce the training time of the model, as depicted in Figure 4.14. Then, the 1 s-model with the lowest validation error is used as the basis for training the 2 s-model. This continues until the training of the 16 s-model based on the best 8 s-model. With this, stable results for all models can be achieved. Each model is trained for 500 epochs. The effect of doubling the number of simulation steps while halving the number of training trajectories cancels each other out, such that the training time of all multi-step models is approximately 1 h on a single core of an i7-9700 CPU @3 GHz.

The performance of the learned models increases with the training length. Especially the number of vehicles that leave the track or collide decreases and is the lowest for the 8 s- and 16 s-models. An example of the best 16 s-model along with the ground truth evolution of the traffic situation is shown in Figure 4.15. The quantitative performance of the models is evaluated together with the single-step model in the next section.

For the 16 s model, the performance between epochs fluctuates strongly and eventually collapses to a policy that often leaves the track. As this does not occur for shorter prediction horizons, this is likely due to the increasingly large errors that occur for 16 s-predictions. These dominate the error gradient computation, but are often due to developments in the situation that could plausibly have evolved differently. Example 4.1 shows an example of how this applies to multi-step training.

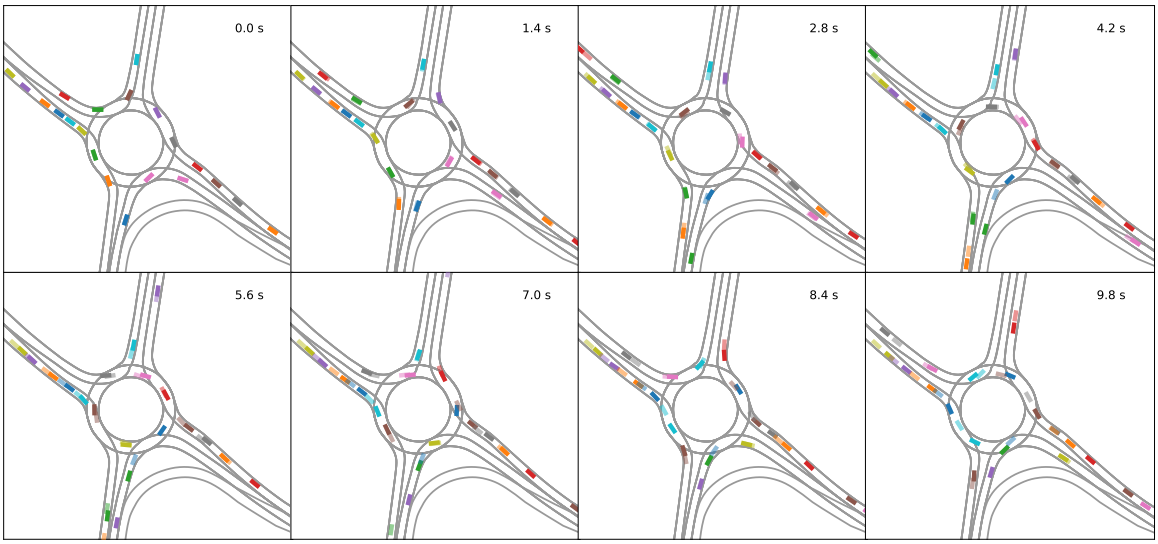


Figure 4.15: Ground truth (solid) and closed loop prediction (transparent) of the 16 s-model in the same situation as Figure 4.10. No vehicle leaves the track or collides. Most predictions remain close to the ground truth throughout the whole prediction horizon, especially when constrained by surrounding vehicles. One notable exception is the blue vehicle coming from the south. While the ground truth vehicle enters the roundabout at 4.2 s, the prediction stops at the entrance to let another vehicle pass. As a knock-on effect, the predictions of the brown and gray vehicle coming from east can enter the roundabout, whereas the corresponding ground truth vehicles need to wait until the ground truth blue vehicle has passed.

Example 4.1: Divergence between prediction and ground truth

Consider the situation in Figure 4.15. At $4.2\,\mathrm{s}$, the blue ground truth vehicle at the southern entry drives into the roundabout, whereas the predicted vehicle stops to let another vehicle pass. While both behaviors are plausible, the loss strongly penalizes this divergence. During training, the error gradient attracts the prediction towards the ground truth, thereby incentivizing the prediction to follow it into the roundabout, regardless of the conflicting olive vehicle. As this divergence between the predicted situation and ground truth is inherent to the nature of the problem and aggravates with longer prediction horizons, it is pointless to train with prediction horizons beyond 8 to $16\,\mathrm{s}$. As soon as the predicted situation diverges too strongly from the ground truth, this phenomenon destabilizes the training and prevents the model from achieving zero failures.

Another perspective on this is given by Metz et al. [Met+22]: The authors argue that while a gradient can be analytically computed by backpropagating through an arbitrary number of steps of a nonlinear simulation, the exact value gives only an illusion of precision, as the outcome is dominated by random or chaotic effects. Two methods employed in this chapter, starting the training with pre-trained models of shorter horizons, and using the Huber loss, thereby effectively clipping the error gradients beyond a certain threshold, can be interpreted as methods to address this problem of volatile gradients. However, predictions for horizons much larger than $10\,\mathrm{s}$ are arguably subject to effects that no predictor could have anticipated at the prediction origin, hence it does not seem sensible to extend multi-step training to longer horizons.

Action Feedback Similar to the previous section, multi-step training is used to train a policy that has additional access to the last acceleration and steering angle to investigate whether it also suffers from causal misidentification. Clearly, knowing the current acceleration and steering is helpful for short-term predictions up to 1 to $2\,\mathrm{s}$, as it allows to predict the continuation of the current kinematics in ambiguous situations, e.g., when a vehicle could both, enter the roundabout or stop at the yield line. After multiple prediction steps, the last action feature becomes uninformative, as it is an artifact of the prediction itself and carries no information on the behavior of the ground truth vehicle. The policy has no means to discern whether the information is reliable, because it is close to the prediction origin, or whether it is unreliable, because it is executed at a later prediction step. The policy neural network can only learn to either rely on the last action, improving the short-term and deteriorating the long-term performance, or it learns to ignore the last action, resulting in no change compared to the model that has no access to the last action. To resolve this flaw, the time since the prediction origin is introduced as a third additional observation. Now, the policy can learn to

rely strongly on the last action at the beginning of the prediction, and put less weight on it at later timesteps.

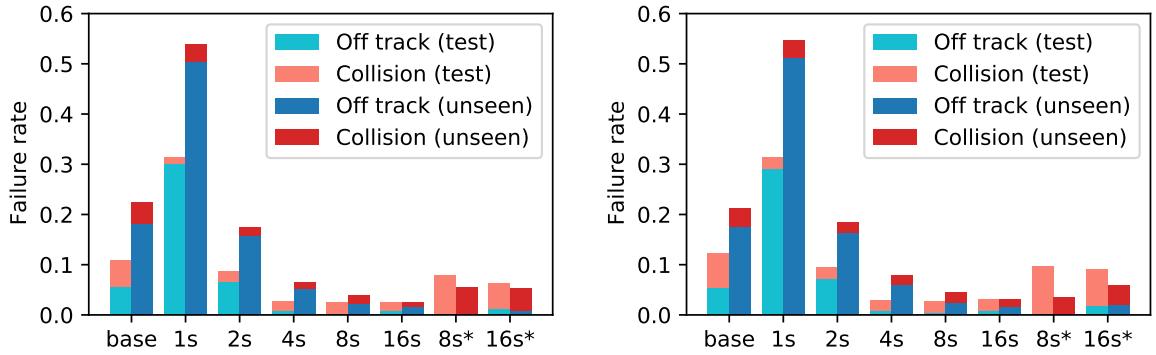
To evaluate the impact of action feedback, a policy with access to the last action and one policy with additional access to the prediction time is trained. Apart from the additional observations, the $1\,\mathrm{s}$-models are trained from scratch for 1000 epochs, as no good baseline model exists. All other training modalities remain unchanged. Again, the $8\,\mathrm{s}$- and the $16\,\mathrm{s}$-model have the lowest collision and off-track rate. With the inclusion of the prediction time, the failure rate decreases. However, it is still significantly higher than for the corresponding multi-step models that have no access to the last action. When executing the learned policies in closed loop simulation, the policies are reluctant to change the acceleration strongly, even if the situation requires it. For example, many collisions occur when an inner-roundabout vehicle should brake, because another vehicle has entered the roundabout before it. Instead, the inner-roundabout vehicle maintains an approximately unchanged acceleration. The detailed results and a comparison to the other models follow in the next section.

4.3.3 Model Comparison

To evaluate the learned models, two different evaluation settings are used. In the *closed loop* regime, all vehicles in the traffic situation are controlled by the learned policy and interact with the other predicted vehicles. This is how the model will be used in a real-world prediction task. In the *open loop* regime, the policy controls only one single vehicle whereas all other vehicles in the situation are played back from the recording. This is how the policy is executed during multi-step training. Clearly, it is unrealistic to use this setting for the evaluation of the prediction performance, because a lot of information about the future is provided by the surrounding vehicles. However, compared to closed loop evaluation, it can be ruled out that erroneous predictions of one vehicle influence the prediction of other vehicles due to knock-on effects as shown in Figures 4.10 and 4.15.

The evaluation is performed on two datasets. First, it is performed on the test dataset at the training roundabout. This dataset has not been used for training and can therefore be used as a reference on how well the learned policy handles situations similar to the training situation. It contains 118 $10\,\mathrm{s}$-situations with a total of 1432 trajectories. Secondly, the evaluation is performed on a dataset that was recorded on another roundabout, containing 402 $10\,\mathrm{s}$-situations with 1918 trajectories. This roundabout has never been seen during training. Evaluations on this dataset therefore show how well the learned policies generalize to situations different from the training situation.

Failure Rates First, the failure rate of the single-step model (denoted *base*) and the five multi-step models (trained on $1\,\mathrm{s}$, $2\,\mathrm{s}$, …, $16\,\mathrm{s}$ trajectories) is compared in the four combi-



(a) Closed Loop

(b) Open Loop

Figure 4.16: Performance of the different models in closed and open loop during a 10 s simulation. Number of trajectories used for evaluation: $N_{\text{test}} = 1432$, $N_{\text{unseen}} = 1918$. The asterisk indicates that the last action and prediction time is included in the observation.

nations of open and closed loop execution on the test and unseen dataset. Additionally, the results for the best two models with additional action and prediction time input are shown, indicated with an asterisk. The single-step model that suffers from causal misidentification is excluded from any further evaluation, as it always performs significantly worse than the other single-step model. A predicted trajectory is designated as a failure when it leaves the track or when it collides with another vehicle. While these failure modes are rarely evaluated in related works on trajectory prediction, they give an important insight into the plausibility of the predictions.

The failure rates are depicted in Figure 4.16. The performance difference of all models between closed and open loop execution is small. Generally, the number of vehicles leaving the track is higher than the number of vehicles colliding. The failure rate in the test situation in the known roundabout is always lower than the failure rate in the unseen roundabout. This indicates that the training would benefit from training data in additional situations to learn a more general policy.

The single-step base model has a failure rate between 10 and 20%. This is a major improvement over previous work [**Sac+21**], where the failure rate of a single-step model lies between 60 and 80%. The improvement can likely be attributed to the use of a significantly larger training set, the use of dropout and additional observations compared to the original work. Surprisingly, the 1 s-model with failure rates between 30 and 50% performs worse than the single-step model. Two reasons for this are the lack of dropout regularization during multi-step training as well as the reduced amount of training data: While single-step training is performed on approximately 60,000 training points, the 1 s-model is trained on approximately 11,300 non-overlapping 1 s-trajectories. For the models that are trained for longer horizons, however, the benefits of multi-step training outweigh these disadvantages: The 2 s-model has a slightly lower failure rate than the base model and the 16 s-model has the lowest overall

failure rate around 3% in both open and closed loop. Despite the inclusion of the prediction time in the $8\,\mathrm{s}$ and $16\,\mathrm{s}$-models, the policies with action feedback rely strongly on the last action feature and produce significantly more collisions than their counterparts without access to this information.

For the $8\,\mathrm{s}$- and the $16\,\mathrm{s}$-model, collisions are the dominant cause of failure. This hints to one fundamental problem of all BC approaches: Pure BC offers no straightforward way to explicitly train the model to brake before collisions or to remain on the track, as discussed in example 4.1. Moreover, collisions and near-collisions occur only rarely in the training data. Hence, learning a policy that handles these critical situations through imitation is hard. It requires either a significantly larger dataset that contains these situations or a different way of confronting the model with these situations. These issues will be addressed in the next chapters.

Along-Track Prediction Error Next, the prediction quality of the models is compared. To this end, the along-track prediction error is measured. Compared to simply measuring the distance between a prediction and the ground truth, measuring the signed along-track distance allows evaluating whether the predictions are behind or in front of the ground truth vehicle. Failed trajectories are included in the evaluation, but only until the point of failure. For example, if a prediction leaves the track, it is interrupted and hence cannot be further evaluated.

For the $16\,\mathrm{s}$-model in the closed loop test situation, the resulting error histogram of the predicted position after $10\,\mathrm{s}$ is shown in Figure 4.17. The average offset is $0.62\,\mathrm{m}$, i.e., the predictions are on average slightly in front of the ground truth. The distribution of the prediction error does not follow a normal distribution, as a Shapiro-Wilk test [SW65] confirms with $p < 0.05$. The distribution of the error has longer tails than a normal distribution. This finding supports the use of the Huber-loss (4.21) for multi-step training, which is more robust to outliers than the commonly used squared loss. Many of the large prediction errors can be explained by different decisions about entering the roundabout between prediction and ground truth vehicle, which also occurs in both Figures 4.10 and 4.15.

Similar error distributions arise for the other models. Instead of showing the full error histogram for one prediction horizon, the course of the empirical mean and standard deviation of the prediction error during $10\,\mathrm{s}$ closed loop prediction is depicted in Figure 4.18. As the error distribution is not normal, the standard deviation can only be interpreted as a measure of dispersion of the prediction error to compare the different models.[1]

[1] Another common measure for model quality is the prediction Root Mean Square Error (RMSE). Given the empirical mean μ_e and standard deviation σ_e that are shown in Figure 4.18 and Figure 4.20, the RMSE can be computed according to $\sqrt{\mu_e^2 + \sigma_e^2}$. For details, see Appendix B.1.

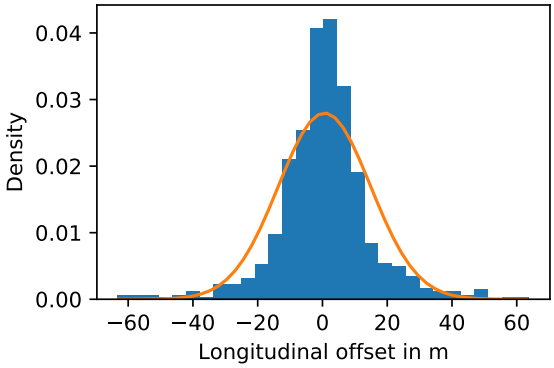


Figure 4.17: Histogram and fitted normal distribution of the longitudinal displacement after a $10\,\mathrm{s}$-prediction of the $16\,\mathrm{s}$-model in the closed loop test situation. $N = 818$, estimated normal distribution parameters $\mu = 0.62\,\mathrm{m}, \sigma = 14.26\,\mathrm{m}$. A Shapiro-Wilk test confirms the visual impression that the error is not normally distributed, $p = 5 \cdot 10^{-18}$.

In the test situation, all evaluated models exhibit similar errors with a mean offset close to $0\,\mathrm{m}$ and a similar course of the standard deviation. The models with action feedback feature a slightly lower mean error and standard deviation for short prediction horizons. On the other hand, the $8\,\mathrm{s}$-model with action feedback has the largest bias for $10\,\mathrm{s}$ predictions in the test situation. In the unseen situation, all predictions systematically lie behind the ground truth positions. This effect is the strongest for the multi-step models that have been trained for long horizons. One explanation for this is that the traffic density at the unseen roundabout is significantly lower, such that vehicles are typically driving at higher speeds. None of the models has learned to drive as fast as the situation allows. Rather, the models stop accelerating after reaching the velocities typical for the training situation. Surprisingly, the single-step and $1\,\mathrm{s}$-model exhibit the lowest prediction errors in the unseen situation, but have the highest failure rates as discussed previously. Figure 4.19 shows a closed loop prediction of a traffic situation to illustrate the bias in the prediction at the untrained roundabout.

The policies are trained in open loop, but executed in closed loop. This can be considered another form of distributional shift, as it implies that the behavior of surrounding vehicles differs between training and execution time from the perspective of one vehicle controlled by the policy. To investigate the severity of this regime shift, the same evaluations are made in open loop. The results are shown in the right part of Figure 4.16 and in Figure 4.20. As to be expected, the prediction error is slightly lower in open loop simulation, because erroneous predictions of one vehicle have no knock-on effect. Again, a large difference in performance between the test situations and the unseen situations exists. However, the overall difference in performance between open and closed loop execution is small, such that it can be concluded that the shift from open loop training to closed loop execution is not problematic.

The comparison in this section shows that multi-step training for horizons above $2\,\mathrm{s}$ significantly reduces the failure rates compared to the commonly used single-step BC. In terms of

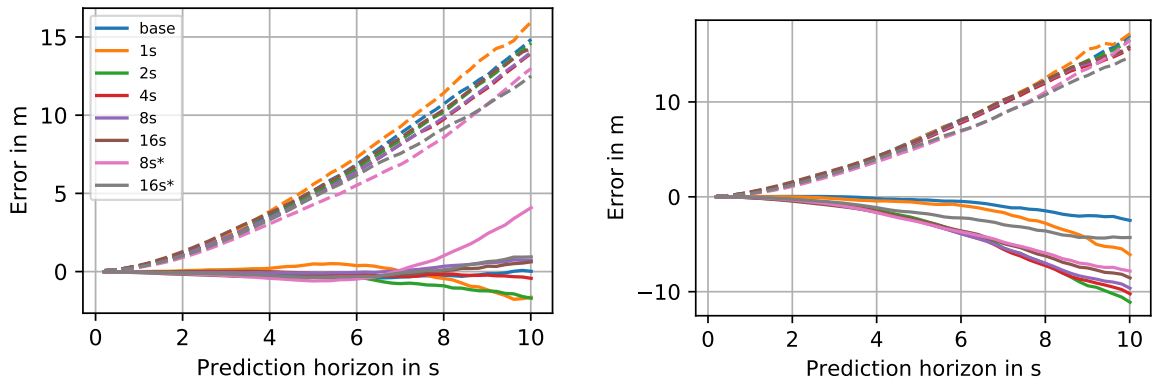


Figure 4.18: Empirical mean closed loop prediction offset (solid) and standard deviation (dashed) during a 10 s prediction on the test roundabout (left) and the unseen roundabout (right). Number of trajectories used for evaluation: $N_{\text{test}} = 1432$, $N_{\text{unseen}} = 1918$. The asterisk indicates that the last action and prediction time is included in the observation. Note that the failure rate of the single-step and the 1 s model is larger than 20% in the unseen roundabout. As predictions are aborted after a failure, these trajectories are not included in the offset calculation. Hence, despite their seemingly good performance, these models cannot directly be compared to the other models.

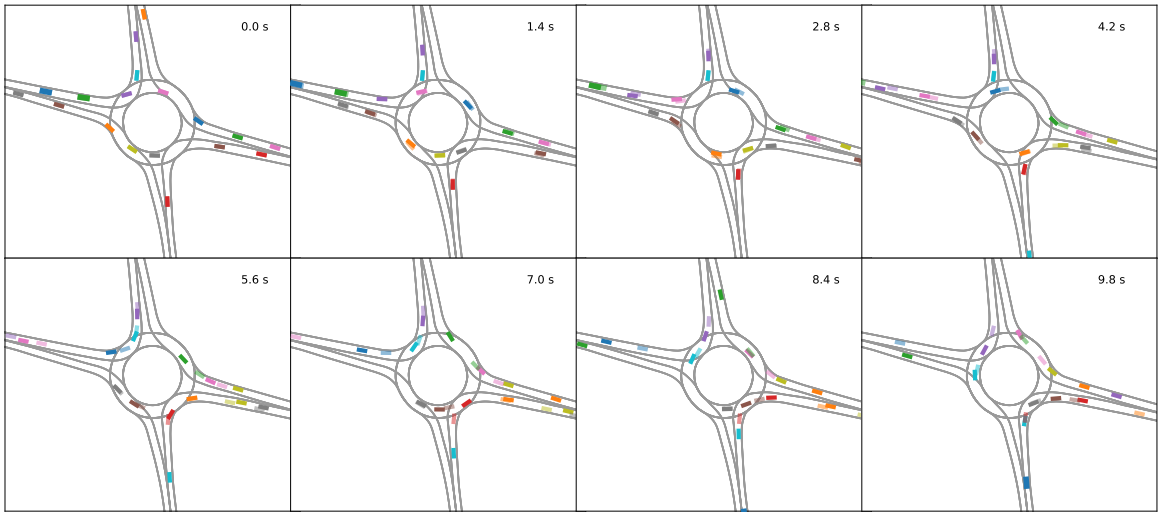


Figure 4.19: Ground truth (solid) and closed loop prediction (transparent) of a situation at the untrained roundabout with the 16 s-model. While no vehicle collides or leaves the track, the predicted vehicles enter the roundabout slower than their ground truth counterparts and often lag several meters behind. Similarly biased predictions are issued by the other multi-step models.

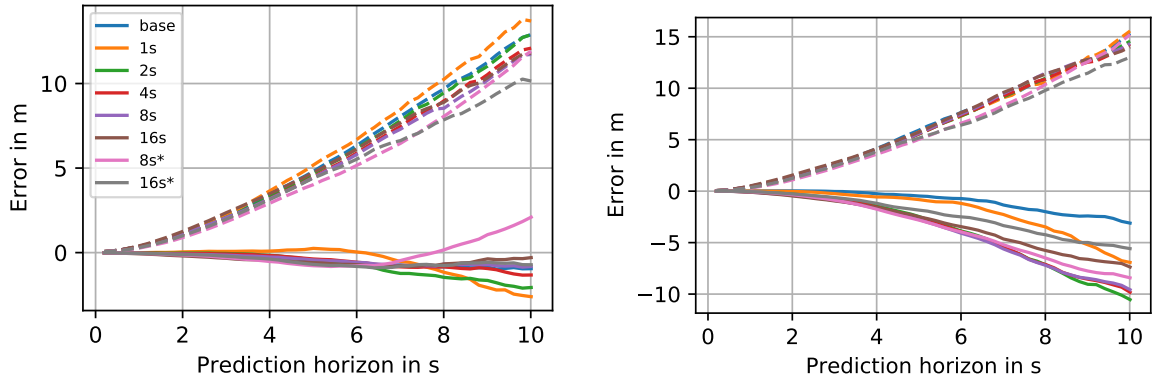


Figure 4.20: Empirical mean open loop prediction offset (solid) and standard deviation (dashed) during a 10 s prediction on the test roundabout (left) and the unseen roundabout (right). Number of trajectories used for evaluation: $N_{\text{test}} = 1432$, $N_{\text{unseen}} = 1918$. The asterisk indicates that the last action and prediction time is included in the observation.

prediction accuracy, all models perform similarly. Feeding back the last action and prediction time into the model slightly improves the prediction accuracy, but increases the failure rates significantly and is therefore not further pursued. While most models are capable of low-bias predictions in situations similar to the training situation, they show a systematic offset in the situation at the unseen roundabout. Thus, to further improve the models, a larger dataset with a more diverse set of training situations is required.

4.4 Conclusion

This chapter investigates two methods to train a policy neural network from a dataset of trajectories. Single-step BC training has been described in various related works. It minimizes the prediction error of the next action. To address the shortcomings of single-step training, multi-step training is proposed. It minimizes the trajectory prediction error when executing the model recurrently for multiple steps. In both cases, the training goal is to imitate the behavior from a dataset, hence the name Behavioral Cloning.

The key findings that answer the research questions raised in the beginning of the experiments section on page 71 are the following:

Single-Step Training Single-step training learns a policy that predicts the next action on average correctly, given the current observation. However, the prediction error of the acceleration has a relatively large spread, as shown in Figure 4.9. This can be explained with the ambiguity of the problem: drivers select different accelerations when faced with the same situation. Also, as the ground truth acceleration signal is effectively estimated as a second order derivative from detected vehicle positions in a drone video, it is relatively noisy.

The learned single-step policy is capable of controlling all vehicles in a simulated traffic situation, as demonstrated in Figure 4.10. The predicted positions remain close to the ground truth for most vehicle for multiple seconds.

Distributional Shift Yet, in some cases, some vehicles controlled by the single-step policy slowly drift off the track. In the quantitative evaluation in Figure 4.16, approximately 10% of all vehicles collide or cross the lateral road boundary. This can be explained with the phenomenon of distributional shift, where the policy is confronted with observations during execution that do not resemble the observations from its training dataset.

Causal Confusion Also, by artificially introducing the last action as an additional policy input, the phenomenon of causal confusion can be demonstrated for the single step model: This lowers the prediction error of the next action, shown in Figure 4.11, but it dramatically increases the errors when executing the policy in the simulation, visualized in Figure 4.12. The reason is that the model relies heavily on the last action as a predictor of the next action, and puts less emphasis on the inputs that describe the environment. While this correlative model of the action being approximately constant between two successive timesteps allows for good single-step predictions, it is incapable of controlling the vehicles in the simulation because it does not learn the functional relation between the observation of the environment and the selected action.

Multi-Step Training The central contribution of this chapter is multi-step training, which addresses the demonstrated shortcomings of the commonly used single-step training approach. The idea is to predict not only the next action, but rather the full trajectory during training. Then, the trajectory prediction error and the gradient of this error with respect to the policy neural network weights is computed. This signal is used in a gradient-descent style for training the policy to minimize the prediction error. One important prerequisite for the automatic gradient computation is that each component of the simulation environment needs to be differentiable, meaning that a specialized implementation of the traffic simulation in an automatic differentiation framework such as PyTorch is required.

The models trained with multi-step training exhibit significantly lower failure rates than the single-step models. Hence, it can be concluded that multi-step training is robust against distributional shift. This can be explained by the model experiencing potential future observations when executed in the simulation during training, thereby learning to address them and to compensate for errors, such as drifting towards the road boundary. This procedure forces the policy to learn a functional model of the relation between observations and actions, because only a functional model minimizes the deviation between predicted and ground truth

trajectories. The benefits of this can clearly be observed in Figure 4.16, where the policies trained with $8\,\mathrm{s}$ or $16\,\mathrm{s}$ trajectories exhibit the lowest failure rates.

Multi-step training also reduces the effect of causal confusion, which is shown by training a policy that additionally has access to its last action. Yet, the multi-step models with this information exhibit higher failure rates than the models without it, indicating that there is still some degree of causal confusion. As a consequence, the last action should not be included in the observation vector.

It is also shown that multi-step models that are trained with longer trajectories ($8\,\mathrm{s}$ or $16\,\mathrm{s}$) exhibit lower failure rates than those that are trained for shorter time horizons. However, the benefit of training for longer simulation horizons wears off when chaotic failure modes start to dominate the prediction error, as discussed in example 4.1. Hence, the optimization procedure becomes increasingly instable for longer horizons. Training for $8\,\mathrm{s}$ has shown to be a good compromise between the quality of the final model and the stability of the training.

On the other hand, even the most successful multi-step policy occasionally leaves the track or collides with other vehicles. The reason for this is that the model is rewarded exclusively for imitating the ground truth behavior during training, which sometimes contrasts with learning a collision-free model, as illustrated in example 4.1. The next chapter introduces a method that addresses this issue by explicitly penalizing vehicles that collide or leave the track.

5 Learning from Rewards: Reinforcement Learning

This chapter builds on the ideas presented in [**Kon+21**; **Sac+22a**].

The previous chapter demonstrates how the space of observations is effectively augmented by multi-step training, compared to single-step training. However, this augmentation is limited: The policy can only be trained in situations for which training data is available, and it does not experience situations or map topologies for which no training data exists. Moreover, no explicit way to achieve goals such as avoiding collisions and staying on the track exists. Therefore, this chapter explores Reinforcement Learning (RL), an alternative method to learning policies.

While BC tries to find a policy that directly imitates the behavior in the training dataset, RL seeks a policy that maximizes a manually specified reward function. This closes the next link in the behavior triangle in Figure 1.2: deriving a policy from a reward function.

Fundamentally, RL methods gather experiences by interacting with the environment. The experiences are then used to deduce which actions lead to high rewards and should be reinforced, and which actions are to be avoided as they result in low rewards. To gather these experiences, the learner must initially act randomly and later focus on selecting the successful actions to fine-tune its behavior model. For this reason, RL is also characterized as learning by trial and error [SB18].

Similar to multi-step training, RL requires a simulation environment as described in Chapter 3, but removes the need for any real-world training data. Moreover, in contrast to multi-step training, the simulation environment is not required to be differentiable. Instead, the gradient is approximated stochastically, which will be shown in Section 5.1. The training can be performed in arbitrary simulated situations and maps, thereby leading to more robust and versatile policies than BC.

Many innovative applications of RL come from robotics, where the goal is often to learn a control policy for tasks with high-dimensional nonlinear system dynamics, e.g., grasping or walking [Iba+21]. Typically, one single robot interacts with an environment that remains stationary during training in this setting. Most RL algorithms focus on this problem. Section 5.1 describes this *single-agent* RL problem formally and summarizes the theory behind the solution approaches employed in this thesis.

In a traffic situation, multiple independent agents interact with each other. For a plausible prediction of the situation, this thesis not only needs to find a control policy for individual agents, but the combined policies should also produce plausible interactions between different agents, e.g., for the negotiation of right-of-way situations. Extensions of the single-agent methods to address the *multi-agent* problem are described in Section 5.2.

The key contribution of this chapter lies in the application of the single- and multi-agent RL algorithms to the problem of behavior modelling. To this end, the algorithms described in the theory chapters are applied to the problem of one single vehicle learning to drive in Section 5.3 and to the problem of learning a driving policy for the full simulation environment with multiple interacting vehicles in Section 5.4. Hereby, the central challenges lie in the implementation: To support the multi-agent setting and further extensions that are presented later in Chapter 6, the RL algorithms need to be implemented from scratch. This also enables a deep insight into their internal functioning. Moreover, a new reward function is presented with focus on plausible behavior in curves and realistic interaction between vehicles, e.g., observance of right-of-way rules and cooperative merging behavior.

Commonly, the goal of RL is to find one single policy that maximizes the reward function. In contrast, this work proposes a method to represent the *heterogeneity* of driving behavior. To this end, Section 5.5 proposes an approach to represent a variety of different behaviors that maximize different reward functions with one single flexible policy.

5.1 Fundamentals of Reinforcement Learning

This section introduces the fundamental terms of single-agent RL. Moreover, the policy gradient RL algorithms applied in this thesis are introduced. The description and notation is based on the book by Sutton and Barto [SB18]. A reader familiar with the topic can skip to Section 5.2.

Partially Observable Markov Decision Processes A Partially Observable Markov Decision Process (POMDP) is an abstract problem formulation of a sequential decision process, depicted in Figure 5.1. An agent interacts with their environment by making observations and executing actions. Each of these transitions is assigned a reward. The goal of the agent is to act in a way that maximizes the aggregated rewards during an episode, i.e., the sequence of transitions until the simulation ends. The decision process is Markovian, because the action of an agent changes the state of the environment in a partially stochastic way that depends exclusively on the current environment state. It is partially observable, because the agent receives a limited observation of the full environment state. The constituting components of a POMDP are listed in Table 5.1.

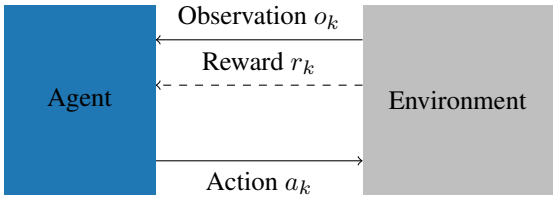


Figure 5.1: A Partially Observable Markov Decision Process: An agent makes observations of the environment, and selects actions to maximize the received rewards.

Table 5.1: Components of a POMDP

Symbol	Description
$y_k \sim \mathbf{y}_k, \quad y_k \in Y$	Environment state: realization, rand. var., and space
$a_k \sim \mathbf{a}_k, \quad a_k \in A$	Action: realization, rand. var., and space
$o_k \sim \mathbf{o}_k, \quad o_k \in O$	Observation: realization, rand. var., and space
$p(y_{k+1}\|y_k, a_k)$	State transition density
$p(o_k\|y_k)$	Observation density
$\mathcal{R} : Y \times A \to \mathbb{R}, \quad \mathcal{R}(y,a) \mapsto r$	Reward function

In the notation, uppercase letters denote sets, bold lowercase letters denote random variables, and regular lowercase letters denote concrete variables, e.g., samples from a random variable. All variables are vectors, except for the scalar reward r. For example, the observation space is O, with a concrete observation $o_k \in O$ being a sample from the observation random variable $\mathbf{o}_k$ that is characterized by its density $p(o_k|y_k)$. As usual in RL literature, sampling from a random variable is indicated by a tilde, e.g., $o_k \sim \mathbf{o}_k$.

The simulation described in Chapter 3 can be interpreted as an instance of a POMDP, if an additional reward is attributed to each transition. In this case, as shown in Figure 3.1, the combination of kinematic model, simulation state, and observation model forms the environment. The policy is executed by the agent and provides an adequate action for each observation. From the perspective of one agent in a traffic situation, all other traffic actors are part of the environment. In the first part of this chapter, the discussion and derivatives focus on the single-agent case, i.e., one single vehicle driving through a lone world.

Solving a POMDP means finding a policy $\pi : O \to A$ for the agent that maximizes the obtained aggregated rewards

$$R(\tau) = \sum_{k=0}^{H} \gamma^k r_k \quad \text{with } r_k = \mathcal{R}(y_k, a_k). \tag{5.1}$$

Hereby, future rewards are attenuated with the discount factor $\gamma \in (0, 1]$, typically close to 1, to ensure convergence of the sum for infinite horizons H. Moreover, this establishes a preference for collecting rewards quickly. Due to the exponential complexity of H-step

decision sequences[1], practical solution approaches search for an approximately optimal solution. Many solution methods exist that can roughly be categorized in model-based and model-free approaches. Model-based approaches implement a model of the environment to predict the influence of the actions of the agent on the environment state and the obtained rewards. This model can be used for planning, for example by simulating the effects of different action-sequences and executing the action that maximizes the predicted accumulated rewards using Monte Carlo Tree Search [KWW22, Ch. 9.6]. One major hurdle of these approaches is obtaining a model that accurately reflects the true environment dynamics for planning. Also, the required computational effort makes them infeasible for quickly predicting the evolution of an observed traffic situation.

Model-free RL, on the other hand, solves these two problems. This set of methods makes it possible to learn a purely reactive behavior policy that directly selects an action based on the observation without any planning. Moreover, the policy is learned through interaction with the environment. This alleviates the need for an explicit formulation of an environment model within the agent. On the other hand, model-free RL typically requires many interactions with the environment until a good policy has been found. While there are some approaches of learning by directly interacting with the real world, e.g. [Haa+19], most model-free approaches learn in a simulated environment, which generates experiences risk-free and typically much faster.

Model-free RL approaches can further be divided into tabular methods that operate on countable, fully observable state spaces, and approximate methods, that operate on continuous observation spaces [SB18, Ch. 9]. Approximate solution methods can be further differentiated into value-based methods and policy-based methods, where only policy-based methods support continuous action spaces [ZY20]. As the focus of this work is on partially observable problems, the description of tabular and value-based methods is omitted for brevity, despite many conceptual similarities. The reader is referred to [SB18] for an unbiased introduction to RL.

One further differentiation needs to be made between On- and Off-Policy algorithms. On-policy RL algorithms, such as Proximal Policy Optimization (PPO) [Sch+17a], improve the policy based on the experiences collected while executing that policy in the simulation environment. On the other hand, off-policy methods, such as Soft Actor Critic (SAC) [Haa+18] and Deep Q-Network (DQN) [Mni+15], are designed to improve the policy not only based on the experience from the most recent policy execution, but rather on a dataset of previously collected experiences [Lev+20]. These methods have the potential to greatly reduce the required amount of simulation during the training. Unfortunately, these methods

[1] Consider a decision problem with countable sets of $|A|$ possible discrete actions and $|O|$ possible observations at each step. An exhaustive search for the best action sequence requires comparing all $(|A||O|)^H$ possible episodes, which becomes infeasible for large planning horizons H and large or continuous action or observation spaces.

cannot be applied in multi-agent settings, because the environment dynamics is not stationary; it depends on the changing policies of all agents in the environment. Thus, old experiences become useless when the policy changes. For this reason, this thesis focuses on on-policy algorithms, described in the next sections.

Similar to BC, the policy in approximate RL solution methods is realized using a function approximator, typically a neural network. Like all function approximators, RL policies cannot handle distributional shift, cf. Figure 4.2. A RL policy is thus incapable of selecting appropriate actions for situations that do not resemble the training situations. However, compared to BC, this can be addressed by ensuring that the model experiences a diverse range of situations during training by initializing the simulation accordingly during training.

5.1.1 Policy Gradient Reinforcement Learning

Policy gradient RL is the domain of RL that enables learning policies for continuous observation and action spaces. The central idea is to represent the policy as a neural network, mapping from observations to actions. The training is performed in a gradient ascent procedure.

For this, the gradient of the policy parameters with respect to the expected cumulated rewards of trajectories, $\nabla_\theta \mathbf{E}_{\tau\sim\pi_\theta}\{R(\tau)\}$, must be estimated. In the following, $\mathbf{E}_{\tau\sim\pi_\theta}\{R(\tau)\}$ is abbreviated as $J(\theta)$. Hereby, $\tau \sim \pi_\theta$ denotes the process of generating a trajectory sample τ by executing the stochastic policy π_θ in the environment. Because this is equivalent to drawing samples from a distribution, e.g., $o \sim \mathbf{o}$, the same notation is used.

Once the gradient has been computed, it can be used to update the policy parameters using gradient ascent,

$$\theta_{n+1} \leftarrow \theta_n + \alpha \nabla_\theta J(\theta_n), \tag{5.2}$$

equivalent to (4.20). Thus, in an iterative procedure, a set of policy parameters θ that maximizes the expected accumulated rewards $J(\theta)$ is found.

While the differentiable simulation described in Section 4.2 can be directly employed for calculating $\nabla_\theta J(\theta)$, RL generally assumes a non-differentiable environment, and approximates the gradient stochastically. This has the benefit of being applicable to arbitrary non-differentiable environments, but entails increased computational complexity. To enable the estimation of the policy gradient, the policy

$$\pi_\theta(\mathbf{a} = a | \mathbf{o} = o)$$

is implemented as an observation-conditional action density. Similar to Section 4.1.1, this is realized by mapping the observations to the parameters of a random distribution, e.g., a bivariate normal distribution. Then, the policy gradient can be estimated as follows:

The following derivation of the policy gradient (5.4) to (5.13) is taken from [Ach18], based on the ideas presented in [Sch16, Ch. 2.6].
The probability of a trajectory

$$\tau = ((y_0, o_0, a_0, r_0), (y_1, o_1, a_1, r_1), \dots) \tag{5.3}$$

being generated by a stochastic policy π_θ in a POMDP environment with known state transition density $p(y_{k+1}|y_k, a_k)$ and observation density $p(o_k|y_k)$ can be expressed as

$$p(\tau|\theta) = p(y_0) \prod_{k=0}^{H} p(y_{k+1}|y_k, a_k)\pi_\theta(a_k|o_k)p(o_k|y_k). \tag{5.4}$$

For brevity, $p(\mathbf{x} = x)$ is always written as $p(x)$. Taking the logarithm transforms the product of the factors into the sum

$$\ln p(\tau|\theta) = \ln p(y_0) + \sum_{k=0}^{H} \left(\ln p(y_{k+1}|y_k, a_k) + \ln \pi_\theta(a_k|o_k) + \ln p(o_k|y_k)\right). \tag{5.5}$$

For the following derivation, the "log-derivative trick" [Ach18]

$$\frac{\mathrm{d}}{\mathrm{d}x} f(x) = f(x) \frac{\mathrm{d}}{\mathrm{d}x} \ln f(x) \tag{5.6}$$

is used in (5.9).
With this, the policy gradient can be computed:

$$\begin{aligned}
\nabla_\theta J(\theta) &= \nabla_\theta \mathbf{E}_{\tau\sim\pi_\theta}\{R(\tau)\} && (5.7)\\
&= \nabla_\theta \int_\tau p(\tau|\theta) R(\tau)\, d\tau && (5.8)\\
&= \int_\tau p(\tau|\theta)(\nabla_\theta \ln p(\tau|\theta)) R(\tau)\, d\tau && (5.9)\\
&= \mathbf{E}_{\tau\sim\pi_\theta}\{\nabla_\theta \ln p(\tau|\theta) R(\tau)\}, && (5.10)
\end{aligned}$$

and with

$$\begin{aligned}
&\nabla_\theta \ln p(\tau|\theta)\\
&= \cancel{\nabla_\theta \ln p(y_0)} + \sum_{k=0}^{H} \cancel{\nabla_\theta \ln p(y_{k+1}|y_k, a_k)} + \nabla_\theta \ln \pi_\theta(a_k|o_k) + \cancel{\nabla_\theta \ln p(o_k|y_k)}\\
&= \sum_{k=0}^{H} \nabla_\theta \ln \pi_\theta(a_k|o_k), \qquad (5.11)
\end{aligned}$$

the policy gradient can be written as

$$\nabla_\theta J(\theta) = \mathbf{E}_{\tau\sim\pi_\theta}\left\{\sum_{k=0}^{H}\nabla_\theta \ln \pi_\theta(a_k|o_k)R(\tau)\right\}. \tag{5.12}$$

In practice, this gradient can be estimated

$$\widehat{\nabla_\theta J(\theta)} = \frac{1}{|\mathcal{D}_{\text{RL}}|}\sum_{\tau\in\mathcal{D}_{\text{RL}}}\sum_{k=0}^{H}\nabla_\theta \ln \pi_\theta(a_k|o_k)R(\tau) \tag{5.13}$$

by repeatedly executing π_θ in the simulation, storing the trajectories in a dataset $\mathcal{D}_{\text{RL}}$, and replacing the expectation $\mathbf{E}_{\tau\sim\pi_\theta}\{\cdot\}$ in (5.12) with the empirical mean $\frac{1}{|\mathcal{D}_{\text{RL}}|}\sum_{\tau\in\mathcal{D}_{\text{RL}}}(\cdot)$.

To compute the policy gradient in any automatic differentiation framework, a pseudo loss function

$$\ell_{\text{RL}}(\theta) = \frac{1}{|\mathcal{D}_{\text{RL}}|}\sum_{\tau\in\mathcal{D}_{\text{RL}}}\sum_{k=0}^{H}\ln \pi_\theta(a_k|o_k)R(\tau) \tag{5.14}$$

is implemented whose gradient ∇_θ is equal to (5.13) and can be determined automatically. This derivation yields the simplest conceivable policy gradient scheme in Algorithm 2, called "REward Increment = Nonnegative Factor x Offset Reinforcement x Characteristic Eligibility (REINFORCE)" [Wil92]. In the algorithm, one iteration of collecting a new trajectory dataset and improving the policy via the estimated gradient is called an epoch.

Algorithm 2 REINFORCE [SB18, Ch. 13], originally proposed by [Wil92]

1: Initialize policy π_{θ_0} randomly.
2: **for** epoch $i = 0..N$ **do**
3: Collect trajectory dataset $\mathcal{D}_{\text{RL}}$ by executing the current policy π_{θ_i} in the environment.
4: $\theta_{i+1} \leftarrow \theta_i + \alpha\nabla_\theta \ell_{\text{RL}}(\theta_i)$ ▷ Improve policy using gradient ascent
5: **end for**

The policy gradient (5.13) has an intuitive interpretation: $\nabla_\theta \ln \pi_\theta(a_k|o_k)$ shows how to raise the probability of selecting action a_k after observing o_k by changing the policy parameters θ. Weighting the sum of these gradients with $R(\tau)$ yields the direction where the probability of good actions (positive reward) increases and the probability of bad actions (negative reward) decreases.

While Algorithm 2 illuminates the fundamental idea of policy gradient methods, the remainder of this section presents important improvements required for the application to real-world problems.

Example 5.1: Practical policy gradient

A car drives towards an obstacle. For simplicity, it always starts from the same initial situation and selects a deceleration only once. It then continues to drive with constant deceleration until it stops or collides. Its goal is to avoid a collision with minimal deceleration. For this simplified situation, the policy $\pi(a|o) = \pi(a)$ does not depend on an observation. The policy is modelled as a Gaussian distribution $\pi(\mathbf{a}) = \mathcal{N}(\mu, \sigma^2)$; here, the policy parameters $\theta = (\mu, \sigma)$ are simply the parameters of the Gaussian. In this simplified example, the trajectory τ can be described via one single action a and the trajectory return $\mathcal{R}(\tau)$ is equal to the reward r.

The simulation is repeated 8 times. In each simulation, a random acceleration is drawn from the current policy, $a \sim \pi(\mathbf{a}) = \mathcal{N}(\mu, \sigma)$. In some cases, the agent selects an acceleration that avoids the collision (positive reward r, green), and in other cases it collides (negative reward r, red). To ensure that the collision is avoided with minimum deceleration, braking is also penalized, such that larger decelerations also lead to negative rewards.

Each combination of action a and reward r is an experience, from which the experience dataset $\mathcal{D}_{\text{RL}}$ is formed. Based on this, the policy is improved in the gradient ascent step. The training progress is illustrated in Figure 5.2, with the eight dots in each image representing the randomly drawn action during each simulation.

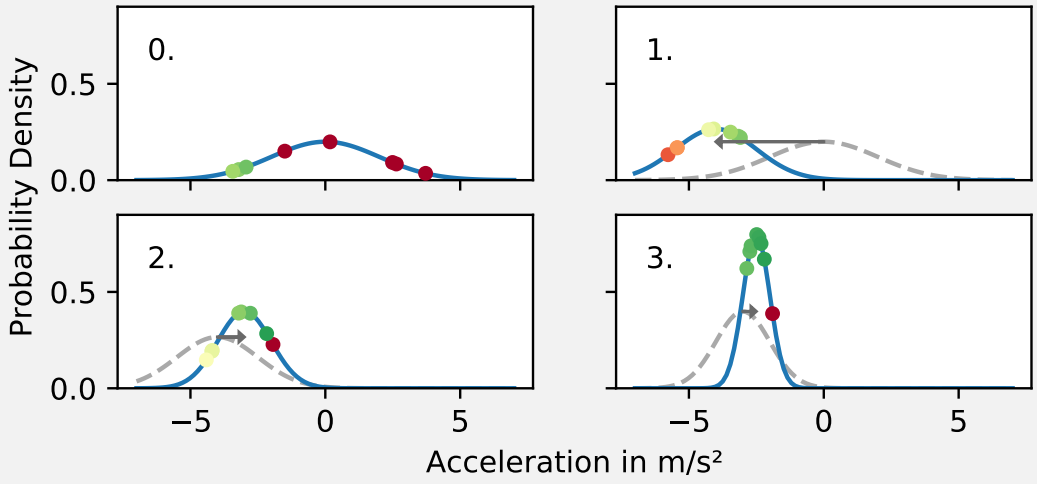


Figure 5.2: Four epochs of the simplest policy gradient

In the initial training epoch 0, the policy $\pi_{\theta_0}(\mathbf{a}) = \mathcal{N}(\mu_0, \sigma_0^2)$ has a large standard deviation and manages to avoid the collision in two cases, when the acceleration is

$-2\,\mathrm{m/s^2}$ or lower. Then, the policy parameters μ and σ are updated by performing one step into the direction of the gradient

$$\begin{aligned} &\frac{1}{|\mathcal{D}_{\text{RL}}|}\sum_{(a,r)\in\mathcal{D}_{\text{RL}}} \nabla_{(\mu,\sigma)} \ln \frac{\pi(\mathbf{a}=a;\mu,\sigma)}{[\mathrm{s^2/m}]} r \\ =&\frac{1}{|\mathcal{D}_{\text{RL}}|}\sum_{(a,r)\in\mathcal{D}_{\text{RL}}} \nabla_{(\mu,\sigma)} \left(-\frac{(a-\mu)^2}{2\sigma^2} - \frac{1}{2}\ln\frac{2\pi\sigma^2}{[\mathrm{m^2/s^4}]}\right) r \end{aligned} \tag{5.15}$$

to increase the likelihood of good actions and decrease the likelihood of bad actions. This is a simplified version of (5.13).
Here, the actual gradient

$$\nabla_{(\mu,\sigma)}(\dots) = \begin{pmatrix} \frac{\partial}{\partial\mu} \\ \frac{\partial}{\partial\sigma} \end{pmatrix}(\dots) = \begin{pmatrix} \frac{(a-\mu)}{\sigma^2} \\ \frac{(a-\mu)^2}{\sigma^3} - \frac{1}{\sigma} \end{pmatrix} \tag{5.16}$$

can be calculated by hand. Filling in the initial values $\mu_0 = 0, \sigma_0 = 2[\mathrm{m/s^2}]$ shows that for an action with positive reward, e.g., $a = -2[\mathrm{m/s^2}]$, the gradient is $\partial/\partial\mu(\dots)(\mu_0) = -1/2\ [\mathrm{s^2/m}]$ and $\partial/\partial\mu(\dots)(\sigma_0) = 0$, i.e., the mean action μ is shifted towards the good action and the standard deviation is unchanged. For an action with negative reward, e.g., $a = 1[\mathrm{m/s^2}]$, the gradient is $\partial/\partial\mu(\dots)(\mu_0) = 1/4\ [\mathrm{s^2/m}]$ and $\partial/\partial\sigma(\dots)(\sigma_0) = -3/8\ [\mathrm{s^2/m}]$, but weighted negatively by r in (5.15) and therefore effectively points away from the bad action, also towards a lower mean action and larger standard deviation.
The gradient for the ascent step is computed as the sum of the gradients of all experiences (5.13). After the gradient ascent step, the mean μ_1 of $\pi_1(\mathbf{a})$ is shifted towards the good actions, and the standard deviation σ_1 is decreased. The amount of change of the policy is entirely determined by the learning rate α in (5.2) and exaggerated here for illustration purposes. Typically, only small steps ($\alpha \approx 10^{-3}[\mathrm{m^2/s^4}]$) are taken towards the gradient to ensure stability in more complex settings. Hereby, dimensional analysis reveals that the unit of α must be the inverse squared unit of the gradient, i.e., $\mathrm{m^2/s^4}$.
In the next epoch, new actions are sampled from the updated policy $\pi_{\theta_1} = \mathcal{N}(\mu_1, \sigma_1^2)$, and the policy continues to be refined in epoch 2 and 3, as illustrated in Figure 5.2. After these four training epochs, the policy selects good actions with high probability. This example also illustrates the importance of starting the training with a large initial standard deviation of the policy, to ensure that good actions can be sampled by the policy.
Sampling from a stochastic policy is only a means to estimate the policy gradient (5.13). After the training is finished, instead of sampling from the policy distribution, the mean action is selected deterministically.

5.1.2 Advantage Estimates

Weighting the gradients $\nabla_\theta \ln \pi_\theta(a_k|o_k)$ with the total trajectory reward $R(\tau)$ in (5.12) to (5.14) leads to two practical problems: After the policy has been trained for a while and receives high returns $R(\tau)$ for all trajectories, the gradients for all trajectories are weighted approximately equal and the gradient towards the best trajectories does not stand out. Moreover, as actions are sampled from the policy in every simulation step, the returns have a large variance, requiring many repeated simulations to obtain a low-variance estimate of the policy gradient. Both effects slow down the optimization procedure.

Instead of weighting the pseudo loss

$$\ell_{\text{RL},\Psi}(\theta) = \frac{1}{|\mathcal{D}_{\text{RL}}|} \sum_{\tau \in \mathcal{D}_{\text{RL}}} \sum_{k=0}^{H} \ln \pi_\theta(a_k|o_k) \Psi_k \tag{5.17}$$

and gradients in (5.12) to (5.14) with $\Psi_k = R(\tau)$, other weighting factors can be used [Sch+18a]. By subtracting a baseline from $R(\tau)$, it can be ensured that the best trajectories stand out, compared to other good trajectories that receive almost equally high rewards on average. The baseline reflects the expected future rewards after observing o_k. If the received rewards exceed the expectations, the actions along the trajectory are advantageous and are to be reinforced; otherwise they are to be avoided.

For this, the sum of rewards after step k is computed for each trajectory according to

$$R(\tau_{k:H}) = \sum_{n=k}^{H} \gamma^{n-k} r_n \tag{5.18}$$

with the infinite horizon case $R(\tau_{k:\infty})$ abbreviated as $R(\tau_{k:})$.

The value function [SB18, Ch. 3.5]

$$V_{\pi_\theta}(o_k) = \mathbf{E}_{\tau_{k:} \sim \pi_\theta} \left\{ R(\tau_{k:}) | \mathbf{o}_k = o_k \right\}. \tag{5.19}$$

is the expected reward when following the policy π_θ starting at observation o_k.

With these two definitions, the baseline-adjusted gradient weight

$$\Psi_{\text{base},k} = R(\tau_{k:}) - V_{\pi_\theta}(o_k) \tag{5.20}$$

can be introduced for the gradient estimation in (5.17) [Ach18]. It is positive for trajectories that exceed the expected rewards and negative otherwise. Hence, it estimates how advantageous the action a_k is, which explains the name "advantage estimate" [Sch+18a].

While the true value function V_{π_θ} is infeasible to obtain in practical settings, because the expectation in (5.19) requires integrating over all possible trajectories, it can be approximated stochastically [Sch+18a]. For this, a neural network

$$\hat{V}_\psi : O \to \mathbb{R}$$

with parameters ψ is trained to predict the future rewards by minimizing the prediction error

$$\ell(\psi) = \frac{1}{|\mathcal{D}_{\text{RL}}|} \sum_{\tau_{k:} \in \mathcal{D}_{\text{RL}}} (\hat{V}_\psi(o_k) - R(\tau_{k:}))^2 \tag{5.21}$$

on the trajectory dataset obtained during one epoch of the RL training using the gradient descent technique Adam [KB15]. Each trajectory in the dataset is used with every starting point $k = 0, 1, \ldots$. Thus, the value estimate is trained not only for the initial observations o_0, but for any observation along the trajectories.

Early Termination In practice, the simulation that is executed to obtain the trajectory dataset $\mathcal{D}_{\text{RL}}$ for training the value network $\hat{V}_\psi$ is limited in several ways: First, the simulation terminates after a fixed number of steps H. Secondly, it terminates as soon as the agent leaves the map, or when it leaves the lateral road boundary.

Therefore, obtaining $R(\tau_{k:\infty})$, which is the prediction goal during the training of the value network (5.21), is infeasible. The simplest remedy would be to train to predict the finite-horizon return $R(\tau_{k:H})$ instead. However, this destabilizes the training, as example 5.2 illustrates.

Example 5.2: Training the value network with finite-horizon returns

Consider a vehicle that is driving on a completely straight road. The vehicle is driving with constant velocity along the track center. Due to this monotonous situation and behavior, it makes the same observation o and receives the same reward r at every time step. The simulation is executed for H steps with a discount factor $\gamma = 1$. The value estimate $\hat{V}_\psi$ is trained using all trajectory slices $\tau_{0:H}, \tau_{1:H}, \ldots, \tau_{H-1:H}$. Assuming that the reward is positive, the obtained return $R(\tau_{k:H}) = \sum_{n=k}^{H} r = (H - k + 1)r$ shrinks as the starting point k increases.
If the value estimate (5.21) is calculated with $R(\tau_{k:H})$ as the prediction target, this causes contradicting prediction targets: The same observation o leads to different returns $R(\tau_{k:H})$, depending on the evaluated number of simulation steps $H - k$. This phenomenon is an instance of state aliasing [WB91; Par+18a] and destabilizes the training of $\hat{V}_\psi$.

Clearly, in real world settings, the observations and rewards during different time steps vary. However, the fundamental problem of different lengths of trajectory slices remains: The value of an observation does not only depend on the observation, but also on the number of remaining simulation steps, which is unknown to the agent and value function.

For this reason, the infinite-horizon reward $R(\tau_{k:})$ is approximated in equation (5.20) and (5.21) using partial-episode bootstrapping [Par+18a]

$$\hat{R}(\tau_{k:}) = \begin{cases} R(\tau_{k:H}), & \text{if termination due to own fault} \\ R(\tau_{k:H}) + \gamma^{(H-k+1)} \hat{V}_\psi(o_{(H+1)}), & \text{otherwise.} \end{cases} \tag{5.22}$$

In the first case, termination due to own fault, for example because the agent crosses the lateral road boundary, no further rewards are assigned after the terminal state at time step H. The infinite horizon sum $R(\tau_{k:\infty})$ is reduced to a finite sum $R(\tau_{k:H})$.

The second case applies to terminations of the simulation for which the agent is not responsible. In these cases, the hypothetically obtained return during an infinite continuation of the simulation is approximated. For this, the obtained rewards $R(\tau_{k:H})$ during the remaining $H - k$ simulation steps are computed. Then, the estimated value of the final observation $\hat{V}_\psi(o_{(H+1)})$ is appended. Following the definition of the value function (5.19), this represents the rewards during an indefinitely continued simulation from the final simulated observation onward.

When minimizing (5.21) using gradient descent, it must be ensured that the gradient ∇_ψ is taken with respect to $\hat{V}_\psi$, but not with respect to $\hat{R}$, which now also depends on ψ as a consequence of the infinite horizon reward approximation (5.22). Otherwise, the loss $\ell(\psi)$ of the value function estimator could not only be reduced by adapting the value estimate of the current observation $\hat{V}_\psi(o_k)$ to match $\hat{R}(\tau_{k:})$, but also the other way around—by adapting the future reward approximation $\hat{R}(\tau_{k:})$ to match the current value estimate $\hat{V}_\psi(o_k)$. This would reverse the sequential nature of rewards along the trajectory.

Generalized Advantage Estimate While the baseline-adjusted gradient weight $\Psi_{\text{base},k}$ ensures that the trajectories that receive higher rewards than expected stand out compared to others, it still suffers from a large variance, because it evaluates the reward sum $R(\tau_{k:H})$ of the execution of a stochastic policy in a stochastic environment. Schulman et al. [Sch+18a] argue that a trade-off between the variance of the weight estimate and its systematic bias can be made by evaluating the reward sum $R(\tau_{k:k+n-1})$ for different numbers of steps n, and approximating the ensuing rewards with the value estimate $\hat{V}_\psi(o_{k+n})$, similar to the

partial-episode bootstrapping (5.22). This leads to the definition of the n-step advantage estimate

$$\begin{aligned}\hat{A}_k^{(n)} &= -\hat{V}_\psi(o_k) + R(\tau_{k:k+n-1}) + \gamma^n \hat{V}_\psi(o_n) \\ &= -\hat{V}_\psi(o_k) + r_k + \gamma^1 r_{k+1} + \cdots + \gamma^{n-1} r_{k+n-1} + \gamma^n \hat{V}_\psi(o_{k+n})\end{aligned} \tag{5.23}$$

with the special cases $n = \infty$, which evaluates the return of the infinite trajectory and is thus equal to $\Psi_{\text{base},k}$, and $n = 1$, which evaluates only the next-step reward r_k, and approximates all following rewards by $\hat{V}_\psi(o_{k+1})$. Any advantage estimate $\hat{A}_k^{(n)}$ can be used as the gradient weight Ψ. Hereby, $\hat{A}_k^{(\infty)}$ has the highest variance and the lowest bias, whereas $\hat{A}_k^{(1)}$ has the lowest variance and the highest bias. The bias of the estimation depends on the influence of the possibly erroneous value estimate $\hat{V}_\psi$, which has a large influence for $\hat{A}_k^{(1)}$ and a negligible influence for $\hat{A}_k^{(\infty)}$.

With this, [Sch+18a] proposes the Generalized Advantage Estimate (GAE)

$$\Psi_k^{\text{GAE}(\lambda,\gamma)} = (1-\lambda)(\hat{A}_k^{(1)} + \lambda \hat{A}_k^{(2)} + \lambda^2 \hat{A}_k^{(3)} + \dots), \tag{5.24}$$

the exponentially weighted average of the n-step estimators. Moreover, the authors provide a formula for the efficient recursive calculation along a simulated trajectory without the need to explicitly compute $\hat{A}_k^{(n)}$, omitted here for brevity. The GAE allows trading off between bias and variance of the gradient weight estimate through the parameter $\lambda \in [0, 1]$. The special cases can be shown to be $\Psi_k^{\text{GAE}(\lambda=0,\gamma)} = \hat{A}_k^{(1)}$ and $\lim_{\lambda \to 1} \Psi_k^{\text{GAE}(\lambda,\gamma)} = \hat{A}_k^{(\infty)}$ [Sch+18a].

The idea of bias-variance trade-off can not only be applied to the advantage estimate, but also to the value estimate: The current formulation of the value network (5.21) predicts the infinite return approximation $\hat{R}(\tau_{k:})$ (5.22). This contains the sum $R(\tau_{k:H})$, which has a high variance for different rollouts of the same policy. Instead, this sum can be evaluated for fewer steps, and the value estimate of the final observation is appended. The n-step return approximation is obtained by adding $\hat{V}_\psi(o_k)$ to the n-step advantage estimate (5.23). For $n < H$, this reduces the variance at the cost of increased bias. This leads to the definition of the GAE return

$$R_k^{\text{GAE}(\lambda,\gamma)} = \Psi_k^{\text{GAE}(\lambda,\gamma)} + \hat{V}_\psi(o_k), \tag{5.25}$$

which is used as the prediction target during training of the value network (5.21). As in the previous subsection, no further rewards are obtained after a termination due to own fault.

These improvements lead to the formulation of the GAE-enhanced policy gradient in Algorithm 3, which has demonstrated practical applicability in two popular RL tasks [Sch+18a].

Algorithm 3 GAE Policy Gradient

1: Initialize neural network for policy π_{θ_0} and value estimate $\hat{V}_\psi$ randomly.
2: **for** epoch $i = 0..N$ **do**
3: Collect trajectory dataset $\mathcal{D}_{\text{RL}}$ by executing the current policy π_{θ_i} in the environment.
4: Compute $\Psi_k^{\text{GAE}(\lambda,\gamma)}$ and $R_k^{\text{GAE}(\lambda,\gamma)}$ for every experience in the trajectory dataset.
5: Train value estimate $\hat{V}_\psi$ to predict $R_k^{\text{GAE}(\lambda,\gamma)}$ on $\mathcal{D}_{\text{RL}}$ by minimizing (5.21) using gradient descent.
6: $\hat{g} \leftarrow \nabla_\theta \ell_{\text{RL},\Psi^{\text{GAE}}}(\theta)$ ▷ Compute GAE policy gradient using pseudo loss (5.17)
7: $\theta_{i+1} \leftarrow \theta_i + \alpha \hat{g}$ ▷ Improve policy using gradient ascent
8: **end for**

5.1.3 Proximal Policy Optimization

The step width α is a key parameter in any gradient ascent or descent algorithm. Choosing too large steps bears the risk of leaving the area where the linearization of the optimized function through the gradient is approximately valid. On the other hand, too small steps lead to an increased number of steps until convergence, resulting in prolonged training duration.

More so, the optimal learning rate α is different for every problem and might even vary for different areas of the same optimization function. In supervised learning settings, this gives rise to advanced gradient descent techniques such as Adam [KB15], which effectively estimate the average and variance of the recent gradient updates to use large steps in monotonous areas and small steps in fluctuating areas.

However, in contrast to standard supervised learning, the dataset $\mathcal{D}_{\text{RL}}$ that is used to estimate the policy gradient (5.13) changes with the policy π_θ, rendering the pseudo-loss $\ell_{\text{RL},\Psi}(\theta)$ in (5.17) non-stationary. This is the fundamental reason why any policy gradient algorithm can only make a single (or few) steps towards the current maximum of $\ell_{\text{RL}}(\theta)$ before it is required to collect a new trajectory dataset $\mathcal{D}_{\text{RL}}$ by executing the updated policy π_θ. $\ell_{\text{RL}}(\theta)$ is a surrogate objective function that is only valid as long as the policy, and hence the dataset, only changes marginally.

This gives rise to the idea of Proximal Policy Optimization (PPO) [Sch+17a]: Schulman et al. [Sch+15b] propose to use importance sampling [RK08, Ch. 5.6]

$$\nabla_\theta J(\theta) = \mathbf{E}_{\tau\sim\pi_\theta}\left\{\sum_{k=0}^{H} \nabla_\theta \ln \pi_\theta(a_k|o_k)\Psi_k\right\} \tag{5.26}$$

$$= \mathbf{E}_{\tau\sim\pi_{\theta,\text{old}}}\left\{\sum_{k=0}^{H} \frac{\pi_\theta(a_k|o_k)}{\pi_{\theta,\text{old}}(a_k|o_k)} \nabla_\theta \ln \pi_\theta(a_k|o_k)\Psi_k\right\} \tag{5.27}$$

$$= \mathbf{E}_{\tau\sim\pi_{\theta,\text{old}}}\left\{\sum_{k=0}^{H} \nabla_\theta \frac{\pi_\theta(a_k|o_k)}{\pi_{\theta,\text{old}}(a_k|o_k)} \Psi_k\right\} \tag{5.28}$$

as a trick to continue using the old policy dataset for training. Sampling from the new distribution $\tau \sim \pi_\theta$ is approximated by sampling from the old distribution $\tau \sim \pi_{\theta,\text{old}}$ and weighting the samples according to the probability density ratio

$$q(\theta) = \frac{\pi_\theta(a_k|o_k)}{\pi_{\theta_\text{old}}(a_k|o_k)}. \tag{5.29}$$

Clearly, this works only as long as the distribution of the trajectories generated with the old policy $\pi_{\theta,\text{old}}$ is reasonably similar to the distribution of those generated with the new policy π_θ. For this reason, the deviation between the two is measured by the relative change in the probability density q.

This leads to the new pseudo loss function

$$\ell_\text{PPO}(\theta) = \sum_{\mathcal{D}_\text{RL}} \min(q(\theta)\Psi_k, \text{clip}(q(\theta); 1-\epsilon, 1+\epsilon)\Psi_k), \tag{5.30}$$

where the change q between the old and the new policy is bounded to be not larger than ϵ. The pseudo loss can safely be maximized during each RL epoch without risking destabilizingly large policy changes. This is in contrast to the previous policy gradient Algorithms 2 and 3, which only take one single step towards the maximum in each training epoch. In (5.30), the clip-function

$$\text{clip}(x; l, u) = \begin{cases} x, & \text{if} \quad l < x < u \\ l, & \text{if} \quad x \leq l \\ u, & \text{if} \quad x \geq u \end{cases} \tag{5.31}$$

limits the improvement in $\ell_\text{PPO}(\theta)$: If the probability density ratio $q(\theta) > (1+\epsilon)$ or $q(\theta) < (1-\epsilon)$, no additional advantage Ψ_k can be obtained. The min-function ensures that this holds only for increased advantage, but that there is no limit to how much a changed policy decreases the advantage. The maximum change ϵ is a hyperparameter of the algorithm, recommended being set to values between 0.2 and 0.3 [And+21]. Smaller values slow down the training unnecessarily, whereas larger values lead to too large deviations between the distributions $\tau \sim \pi_{\theta,\text{old}}$ and $\tau \sim \pi_\theta$, thereby destabilizing the training.

The combination of PPO and GAE that is used to train the agents in this thesis is described in Algorithm 4.

5.2 Multi-Agent Reinforcement Learning

Standard RL, as described in the previous section, is employed to learn a policy that maximizes the obtained rewards of a single agent in a POMDP, see p. 90. This suffices to train an agent to drive on a lonely map. If the interaction with other vehicles shall be learned, three

Algorithm 4 Proximal Policy Optimization with Generalized Advantage Estimate

1: Initialize neural network for policy π_{θ_0} and value estimate $\hat{V}_\psi$ randomly.
2: **for** epoch $i = 0..N$ **do**
3: Collect trajectory dataset $\mathcal{D}_{\text{RL}}$ by executing the current policy π_{θ_i} in the environment.
4: Compute $\Psi_k^{\text{GAE}(\lambda,\gamma)}$ and $R_k^{\text{GAE}(\lambda,\gamma)}$ for every experience in the trajectory dataset.
5: Train value estimate $\hat{V}_\psi$ to predict $R_k^{\text{GAE}(\lambda,\gamma)}$ on $\mathcal{D}_{\text{RL}}$ by minimizing (5.21) using gradient descent.
6: Maximize $\ell_{\text{PPO}}(\theta)$ with GAE $\Psi^{\text{GAE}(\lambda,\gamma)}$ using gradient ascent.
7: **end for**

requirements arise: First, the other vehicles must be part of the simulated world. Secondly, they must be part of the observation vector, such that an agent can react to them. And thirdly, the simulation environment must simulate the behavior of the other vehicles realistically, such that the agent can learn to realistically interact with them.

While the first two requirements are already covered by the simulation environment described in Chapter 3, the third requirement leads back to the beginning: Learning a behavior model that plausibly interacts with other vehicles requires a behavior model that simulates the other vehicles.

The simplest way to resolve this chicken and egg situation would be to control the surrounding vehicles by a simpler surrogate policy, e.g., driving with constant velocity along the track center. Then, the agent could be trained using PPO to successfully interact with these surrounding vehicles. However, the learned policy would reflect the experiences during training, implicitly encoding the assumption that other vehicles always drive with constant velocity. When the learned policy is later executed by all agents, the environment dynamics changes from the perspective of each agent. This can be considered another instance of distributional shift between training and execution [Bha+18].

Partially Observable Stochastic Game (POSG) Instead, the learning problem can be formulated as a POSG [HBZ04; BDK20], which is a generalization of a POMDP to multiple agents. It consists of the components listed in Table 5.2. The general setup is illustrated in Figure 5.3: Now, multiple agents interact with the environment simultaneously. At each time step k, each agent j makes an individual observation. Interaction between the agents becomes possible, because they can see relevant surrounding vehicles via their representation in the observation vector (see Table 3.1). Based on this observation, each agent selects an appropriate action using its behavior policy $\pi^j(a^j|o^j)$.

Equal to POMDPs, solving a POSG means to find a set of policies $\pi^1, \ldots, \pi^j$ such that each agent maximizes its individual return. No globally optimal set of policies exists, simply because optimality cannot be defined with respect to multiple individual returns [OA16].

Table 5.2: Components of a POSG

Symbol	Description
$\{1, \dots, N\}$	Set of N agents
$y_k \sim \mathbf{y}_k, \quad y_k \in Y$	Global environment state: realization, rand. var., and space
$a_k^j \sim \mathbf{a}_k^j, \quad a_k^j \in A^j$	Action: realization, rand. var., and space of agent j
$o_k^j \sim \mathbf{o}_k^j, \quad o_k^j \in O^j$	Observation: realization, rand. var., and space of agent j
$p(y_{k+1} \mid y_k, a_k^{1:N})$	State transition density
$p(o_k^j \mid y_k)$	Observation density of agent j
$\mathcal{R}^j : Y \times A^j \to \mathbb{R}, \quad \mathcal{R}^j(y, a^j) \mapsto r$	Reward function of agent j

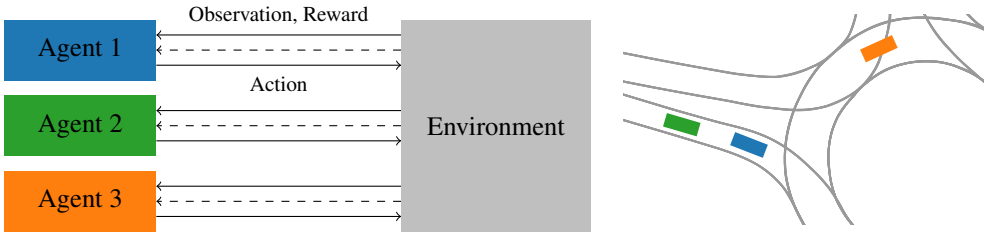


Figure 5.3: A Partially Observable Stochastic Game: Multiple agents interact with the environment simultaneously. At each step, each agent receives a local observation of its surrounding and selects an appropriate action. A reward is assigned to each of these transitions.

Instead, if the search for individually optimal policies converges, it must converge to a set of policies where no agent can improve its return by changing its policy while the other agents keep their policies fixed. This stable point in the policy space is called a Nash Equilibrium [Nas51; Bin07; OA16]. The existence of at least one such equilibrium can be proved only for simple games [Nas51].

Solving the POSG One approach to finding the return-maximizing policies is to switch to a centralized regime, where one "super agent" receives the concatenated observations from all agents, and selects an action for each to maximize the sum of returns of all agents [BBD08; GEK17]. Effectively, this converts the POSG into one large POMDP that can be solved with single-agent RL algorithms. However, assuming a centralized super agent that has access to all individual observations is an implausible model for a traffic situation, in which each driver observes, reasons, and acts individually. This would remove the need to coordinate actions between independent drivers, e.g., when negotiating right-of-way situations. Moreover, as the sum of returns is maximized, this approach fails to model uncooperative behavior where one driver selects actions that benefit himself, but harm the surrounding vehicles.

A more general solution approach to POSG is independent learning. The fundamental idea is to separate the multi-agent problem into N single-agent POMDPs, where each agent treats the other agents as a part of the environment [Tan93]. Then, standard RL algorithms can be applied to determine an optimal policy for each agent. However, the POMDPs resulting

from this separation are not stationary [BBD08; Low+17; GEK17]: As the policies of the surrounding agents change, the dynamics of the simulated environment changes. This prohibits the use of off-policy RL algorithms, such as Soft Actor Critic (SAC) [Haa+18], because they use experiences from an experience buffer of earlier policy rollouts, which become invalid when the environment dynamics change.

PPO, as described in Section 5.1.3, is an on-policy method that optimizes the policy exclusively on the last policy rollout and is thus robust to changing environment dynamics. A recent work by de Witt et al. [dWit+20] suggests that an independent variant of PPO dubbed Independent Proximal Policy Optimization (IPPO) achieves state-of-the-art performance on the challenging StarCraft Multi-Agent Challenge (SMAC) benchmark [Sam+19]. In SMAC, a real-time strategic video game, multiple agents from one team need to cooperate to overpower the opposing team. Additional experiments by Yu et al. [Yu+22] also indicate competitive performance of IPPO on three more MARL benchmarks.

While SMAC is fully cooperative, i.e., all agents try to maximize a joint reward, the agents in a traffic situation have individual and sometimes competing goals, e.g., when a driver squeezes into a roundabout and forces another vehicle to brake. Still, the results from [dWit+20; Yu+22] inspire this thesis to apply IPPO to the problem of learning a driving policy for multiple interacting agents.

Parameter Sharing For simplicity, it is assumed that the vehicles in the traffic situation are homogeneous, i.e., all vehicles follow the same kinematic bicycle model described in Section 3.1 with equal parameters, and their action ranges (acceleration and steering) have equal boundaries. Moreover, each agent is assigned rewards through the same reward function. Intuitively, if all agents have equal properties and goals, then they should also behave equally when faced with the same observation. Otherwise, if one behavior would result in higher expected returns, then all agents should act according to this return-maximizing behavior. With this presumption, instead of training one policy per agent, it suffices to train only one single policy that is executed by each agent. This technique is referred to as parameter sharing [GEK17] and has been subject of research for more than three decades [Tan93].

During training, all agents interact simultaneously with the environment by making observations, selecting actions according to the same policy $\pi_\theta(a|o)$ and receiving rewards. The only difference to the single-agent case listed in Algorithm 4 is that the dataset $\mathcal{D}_{\text{RL}}$ is constructed from the collective experience of all agents. Compared to concurrent learning of individual policies based on individual experience datasets, this has the additional advantage that much more experiences are available for training the policy.

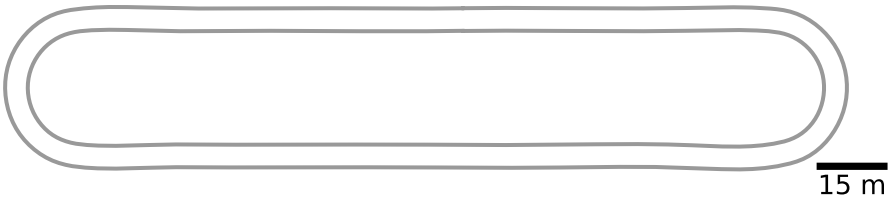


Figure 5.4: The oval track used for single-agent RL consists of two straight segments of $150\,\mathrm{m}$ length and two circular segments with radius $r = 15\,\mathrm{m}$. The track width is $5\,\mathrm{m}$.

5.3 Experiments: Single-Agent Reinforcement Learning

This section applies the ideas of single-agent RL from Section 5.1. The goal is to demonstrate the presented combination of PPO and GAE (Algorithm 4) with a simple example to gain an insight into its functioning. To this end, a single agent is trained to drive on the oval track shown in Figure 5.4, while maximizing the reward. The simulation from Chapter 3 is used. As this section is concerned with single-agent RL, the observation vector consists only of features relevant to this setting. Concretely, the observation (see Table 3.1) consists of the current speed, the distances to both road boundaries, the heading angles $\psi_{0...20}$, and the road curvatures $c_{0...20}$.

The central questions investigated in this experiments section are:

- How can the previously introduced algorithms PPO and GAE be applied to learn a driving policy?
- What are the terms and relative weights of a reward function to teach an agent to drive with RL?
- Which factors have to be considered to accelerate the training process?

5.3.1 Reward Function

The rewards are chosen to encourage progress along the track as well as avoiding discomfort due to accelerations. The agent should drive as fast as possible, while maintaining comfortable accelerations. The general reward function for this scenario

$$\mathcal{R}(v, a_{\text{lon}}, a_{\text{lat}}) = \omega_{\text{vel}}\mathcal{R}_{\text{vel}}(v) - \omega_{\text{acc,lon}}\mathcal{R}_{\text{acc,lon}}(a_{\text{lon}}) - \omega_{\text{acc,lat}}\mathcal{R}_{\text{acc,lat}}(a_{\text{lat}}) + \mathcal{R}_{\text{offtr}} \quad (5.32)$$

encourages higher speed v and penalizes longitudinal and lateral accelerations a_{lon} and a_{lat}.[2] The weights ω are used to balance the importance of the different reward terms. $\mathcal{R}_{\text{offtr}}$ is a constant penalty that is assigned when the vehicle leaves the track.

The terms of the reward function are deduced under the following set of assumptions:

- Drivers want to make progress along the track, hence their speed must be rewarded.
- The lateral acceleration is penalized according to the squared magnitude a_{lat}^2.
- There exists an optimal lateral acceleration a_{lat}^* independent of the road curvature. The trade-off between the reward gained by the speed of the vehicle and the penalty received for the lateral acceleration should be optimal for this lateral acceleration, regardless of the road curvature. Hence, a driver would adapt its speed to maintain $a_{\text{lat}} = a_{\text{lat}}^*$ on a curved track, driving faster on larger radii and slower on smaller radii.

To deduce reward function that fulfills these properties, consider a vehicle driving on a circular track with radius r. The velocity that corresponds to the optimal lateral acceleration a_{lat}^* on a perfectly circular track is $v^* = \sqrt{a_{\text{lat}}^* r}$. What are the terms of a reward function that is maximum for this velocity on a circular track?

With $a_{\text{lat}} = v^2/r$, the reward function for this special case can be written as

$$\mathcal{R}(v, a_{\text{lon}}) = \omega_{\text{vel}}\mathcal{R}_{\text{vel}}(v) - \omega_{\text{acc,lon}}\mathcal{R}_{\text{acc,lon}}(a_{\text{lon}}) - \omega_{\text{acc,lat}}\mathcal{R}_{\text{acc,lat}}(v^2/r) + \mathcal{R}_{\text{offtr}}. \tag{5.33}$$

In the stationary case, the vehicle is driving with a constant velocity, such that the reward for the longitudinal acceleration can be disregarded. The maximum reward with respect to the velocity is obtained, when

$$\frac{\mathrm{d}}{\mathrm{d}v}\mathcal{R}(v, a_{\text{lon}}) \stackrel{!}{=} 0 \tag{5.34}$$

is satisfied. This can be written as

$$\omega_{\text{vel}}\frac{\mathrm{d}\mathcal{R}_{\text{vel}}(v)}{\mathrm{d}v} \stackrel{!}{=} \omega_{\text{acc,lat}}\frac{\mathrm{d}\mathcal{R}_{\text{acc,lat}}(v^2/r)}{\mathrm{d}v}. \tag{5.35}$$

With this, appropriate functions for $\mathcal{R}_{\text{vel}}$ and $\mathcal{R}_{\text{acc,lat}}$ can be chosen, such that the initial requirement, maximum reward for a specific lateral acceleration, is satisfied. It is common to assign a quadratic penalty for the acceleration [Nau+20],

$$\mathcal{R}_{\text{acc,lat}}(v^2/r) = (v^2/r)^2/[\mathrm{m}^2/\mathrm{s}^4]. \tag{5.36}$$

[2]The reward function $\mathcal{R}(y, a)$ was previously defined to depend on the state y and action a. As the speed and accelerations are constituents of these vectors or can be derived from them, they are directly used as the arguments of the reward function here for clarity.

From (5.35), it follows that the reward for the velocity must be logarithmic,[3]

$$\mathcal{R}_{\text{vel}}(v) = c_1 \ln(v/[\text{m/s}]) + c_2. \tag{5.37}$$

Only this $\mathcal{R}_{\text{vel}}(v)$ ensures that the maximum of the total reward function is obtained for a specific lateral acceleration, as the following shows. The constants are set to $c_1 = 1, c_2 = 0$, because c_2 does not influence the position of the maximum and c_1 can be considered a part of $\omega_{\text{acc,lat}}$.

Differentiating both rewards with respect to v according to (5.35) equates to

$$\omega_{\text{vel}}/v \stackrel{!}{=} \omega_{\text{acc,lat}} 4v^3/r^2/[\text{m}^2/\text{s}^4]. \tag{5.38}$$

It follows, that the maximum reward is obtained when the comfortable acceleration

$$v^2/r/[\text{m/s}^2] = \sqrt{\frac{\omega_{\text{vel}}}{4\omega_{\text{acc,lat}}}} = a^*_{\text{lat}}/[\text{m/s}^2] \tag{5.39}$$

is reached. In other words, by setting the weights according to

$$\frac{\omega_{\text{vel}}}{\omega_{\text{acc,lat}}} = 4(a^*_{\text{lat}})^2/[\text{m}^2/\text{s}^4], \tag{5.40}$$

the maximum reward on a curved track is obtained, when driving with v^*, such that the lateral acceleration a^*_{lat} occurs. Driving faster would lead to a higher acceleration penalty, and driving slower would lead to lower rewards for the velocity.

The penalty for the longitudinal acceleration can be set independently. Assuming that the driver is equally averse to longitudinal and lateral accelerations, as indicated by [Nau+20], the weights $\omega_{\text{acc,lon}} = \omega_{\text{acc,lat}} = \omega_{\text{acc}}$ for both penalties are chosen equally and the longitudinal acceleration is also penalized quadratically.[4] With this, the reward function used in the following is

$$\mathcal{R}(v, a_{\text{lon}}, a_{\text{lat}}) = \omega_{\text{vel}} \ln(\max(v/[\text{m/s}], \varepsilon)) - \omega_{\text{acc}}(a_{\text{lon}}/[\text{m/s}^2])^2 - \omega_{\text{acc}}(a_{\text{lat}}/[\text{m/s}^2])^2 + \mathcal{R}_{\text{offtr}}. \tag{5.41}$$

The weights are set to $\omega_{\text{vel}} = 1/\ln(10)$, $\omega_{\text{acc}} = 1/(9\ln(10))$, and leaving the track is penalized with $\mathcal{R}_{\text{offtr}} = -100$. Following (5.40), this corresponds to an optimal acceleration of $a^*_{\text{lat}} = 1.5\,\text{m/s}^2$ in curves. As only the ratio $\omega_{\text{vel}}/\omega_{\text{acc}}$ is relevant for the optimal lateral acceleration,

[3] To avoid the indefiniteness of the logarithm at 0, the reward $\mathcal{R}_{\text{vel}}$ is clipped to a minimum value of $\ln(\varepsilon)$ with $\varepsilon = 0.1$ in practice, i.e., $\mathcal{R}_{\text{vel}}(v) = \ln(\max(v/[\text{m/s}], \varepsilon))$. The ensuing case discrimination is ignored in the following for brevity, as it can be assumed that the optimal speed v^* is larger than $0.1\,\text{m/s}$.

[4] In fact, any other exponent $k \geq 1$ for the absolute acceleration terms could be used and also leads to a maximum reward for a specific constant lateral acceleration. The calculations are omitted here for brevity.

the weights ω_{vel} and ω_{acc} are divided by $\ln(10)$ to ensure that a reward of 1 is obtained when the vehicle is driving with $10\,\mathrm{m/s}$ at no accelerations.

5.3.2 Learning to Drive

With the reward function (5.41), the training can be started. All training parameters are listed in Table C.3, and described in the following. In each training epoch, 50 trajectories are simulated and aggregated in $\mathcal{D}_{\text{RL}}$ to evaluate and improve the current policy. For the training, PPO is used in combination with GAE (Algorithm 4). Given an observation o, the policy neural network

$$f_\theta : o \mapsto (\mu_{\text{acc}}, \mu_\delta, \ln(\sigma_{\text{acc}}), \ln(\sigma_\delta)) \tag{5.42}$$

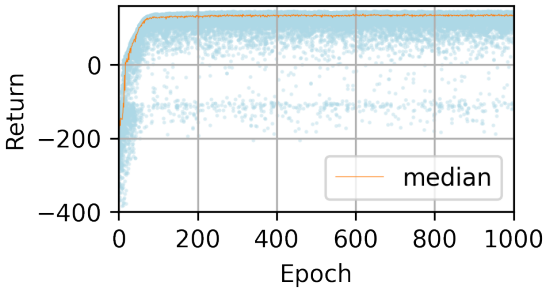
emits the mean $(\mu_{\text{acc}}, \mu_\delta)$ and standard deviation $(\sigma_{\text{acc}}, \sigma_\delta)$ of the acceleration and steering angle, respectively. The standard deviation is independent of the observation; it exclusively depends on θ and typically decreases during training. Since the output of the neural network is not limited to positive values, the standard deviation output is represented as a logarithmic value. To guarantee positive standard deviations, this value is then processed through an exponential function. Apart from the additional output of $(\ln(\sigma_{\text{acc}}), \ln(\sigma_\delta))$ and the reduced number of inputs, the network architecture is identical to the networks used in the experiments on BC. The architecture is depicted in the appendix in Figure C.1.

The mean action vector $\mu_{\mathbf{a}} = (\mu_{\text{acc}}, \mu_\delta)^\top$ and the diagonal covariance matrix $\Sigma_{\mathbf{a}} = \mathrm{diag}(\sigma_{\text{acc}}^2, \sigma_\delta^2)$ parameterize the policy distribution

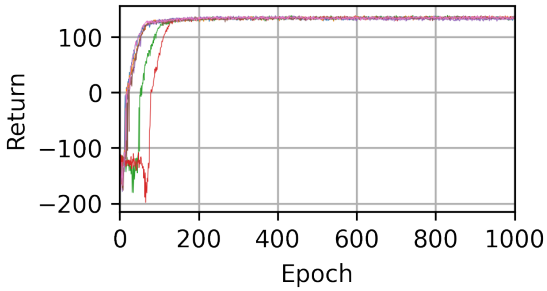
$$\pi(\mathbf{a}|\mathbf{o} = o) = \tanh(\mathcal{N}(\mu_{\mathbf{a}}, \Sigma_{\mathbf{a}})), \tag{5.43}$$

which is a squashed Gaussian distribution. Applying the $\tanh$ function to the normal distribution ensures that all actions are bounded in $(-1, 1)$. The properties of the squashed Gaussian distribution are described in detail in Appendix B.2.

As the covariance matrix is diagonal, the distribution of the acceleration and the steering is treated as two separate univariate distributions with scalar μ and σ in the following. The initial policy standard deviations σ are set to 1, such that the squashed Gaussian distribution is approximately a uniform distribution on $(-1, 1)$ for both, acceleration and steering, see Appendix B.2. During training, the standard deviation typically decreases. For $\sigma \ll 1$, the resulting distribution resembles a Gaussian distribution $\mathcal{N}(\tanh(\mu), \sigma)$. Finally, the actions are scaled and shifted in the simulation environment: The acceleration a_{lon} is scaled and shifted to be between $-7\,\mathrm{m/s^2}$ and $3\,\mathrm{m/s^2}$, and the steering angle δ is scaled to be in $(-\pi/7, \pi/7)$ rad.



(a) Median $\overline{R(\tau)}$ and single-epoch returns $R(\tau)$ (scattered, cyan) of one single training run.



(b) Median returns $\overline{R(\tau)}$ of 50 trajectories during seven independent training runs.

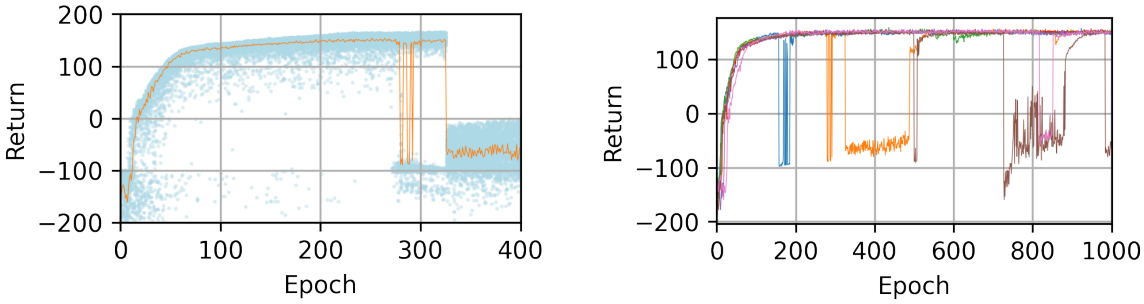
Figure 5.5: Undiscounted returns during RL training. The undiscounted finite return is the sum of rewards along one simulated trajectory. During each epoch, 50 trajectories are simulated with the returns scattered in (a). For a comparison of different training runs, only the median return is displayed in (b). The median is used instead of the mean, because it is more robust to a fluctuating number of outliers (leaving the track).

To ensure a diverse set of experiences, the vehicles are initialized at a random position along the track, with a random heading, speed and lateral offset.[5] Each simulation is executed for $H = 200$ steps, or until the agent leaves the track. The simulation step size is $\Delta t = 0.2\,\mathrm{s}$. Empirically, shorter simulation durations prevent the agent from experiencing the long-term effects of their actions, e.g., accelerating leads to an immediate penalty, but higher long-term rewards for the velocity.

The training is inherently stochastic, because the policy and value network are initialized randomly, the initial situations are randomized, actions are sampled from the policy, and the experience samples on which the training is performed are also sampled randomly. To ensure reproducibility, the training is repeated seven times. The resulting reward curves are shown in Figure 5.5b. Training is performed for 1000 epochs, which takes approximately $1\,\mathrm{h}$ on a single core of an Intel i7-9700 @ $3\,\mathrm{GHz}$. After approximately 300 training epochs, all training runs converge to similar values between $\overline{R(\tau)} = 130$ and $\overline{R(\tau)} = 135$ for the median return. The returns do not increase significantly during the additional training epochs. The average reward per step is $\bar{r} = \overline{R(\tau)}/H \approx 0.66$. Most of the training time is spent for running the simulation, whereas the PPO step is relatively fast. Thus, there is no benefit in executing the neural network training on a GPU. With 50 trajectories of $40\,\mathrm{s}$ duration, the RL agent experiences $33\,\mathrm{min}$ of driving per training epoch.

Following the recommendation from [Sch+17a], training is performed by executing multiple optimization steps for the PPO pseudo loss $\ell_{\mathrm{ppo}}(\theta)$ with randomly drawn samples ("mini-batch") from the experience dataset $\mathcal{D}_{\mathrm{RL}}$ per training epoch. Compared to an optimization on the full dataset, this has been shown to yield policies with better performance [And+21].

[5] Concretely, the initial heading is distributed according to $\psi_{\mathrm{init}} \sim \mathcal{N}(0; \sigma = 0.1\mathrm{rad})$, the initial speed is uniformly distributed, $v_{\mathrm{init}} \sim \mathcal{U}(0, 20\,\mathrm{m/s})$, and the lateral offset is $\sim \mathcal{N}(0; \sigma = 0.15\,\mathrm{m})$.



(a) Median $\overline{R(\tau)}$ (orange) and single-epoch returns $R(\tau)$ (scattered, cyan) of one single training run.

(b) Median returns $\overline{R(\tau)}$ of 50 trajectories during seven independent training runs.

Figure 5.6: Undiscounted returns during RL training with lower policy standard deviation. Allowing for lower action noise destabilizes the training. For better visibility, only the first 400 epochs are shown in (a). During each epoch, 50 trajectories are evaluated and scattered in the plot. After approximately 250 epochs, the policy stops colliding at all. Few epochs later, the policy makes more collisions than before and the returns collapse entirely after approximately 320 epochs. The phenomenon can be observed in multiple independent training runs whose median returns are displayed in (b).

Figure 5.5a shows the return of each of the 50 simulated trajectories during each epoch of one training run, as well as the median return. During early training, most trajectories leave the track before the end of the simulation. In these cases, the trajectories receive a reward of $\mathcal{R}_{\text{offtr}} = -100$. As the actions are selected randomly during early training, typically additional negative rewards are aggregated before leaving the track due to the penalties on the lateral and longitudinal acceleration. This leads to episodes with returns that are significantly lower than -100. Within the first 30 epochs, the agent learns to stay on the track in most cases. In the following epochs, the policy is refined to collect additional rewards through higher speeds and reduced accelerations.

Exploration Noise Throughout the training, some episodes continue to leave the track. Leaving the track at later training epochs can be explained by a combination of an adverse initial state of that vehicle and bad action samples. For the training runs in Figure 5.5, the effective minimum standard deviation of steering angles selected by the policy is restricted to be above $e^{-2}\pi/7 \approx 0.06\,\text{rad}$.[6] Allowing for smaller standard deviations ($e^{-4}\pi/7$) leads to the complete prevention of vehicles leaving the track, as Figure 5.6a shows: Only six vehicles leave the track between training epoch 200 and 270. Then, the training collapses. The reason is that the policy has learned to approach dangerously close to the road boundary,

[6]The value can be explained as follows: The policy neural network emits a logarithmic standard deviation, that is exponentiated to ensure that the standard deviation is positive. The exponent -2 was determined empirically as a value that produces stable behavior. Finally, the steering angle action is passed through a $\tanh$ function and multiplied by $\pi/7$ to ensure that the policy can only select steering angles within $(-\pi/7, \pi/7)$.

and eventually crosses it. It can only slowly recover from this behavior, because it only rarely samples good actions when σ is small, and thus cannot undo its mistake. This phenomenon is illustrated in example 5.3.

Example 5.3: Effect of low exploration noise

Consider again example 5.1. After many training epochs, the policy standard deviation σ is tiny. Due to a too large step size α, the mean action has shifted into a region where a collision cannot be avoided anymore, as shown in the left plot of Figure 5.7. When not a single good action is sampled, the gradient (5.15) does not lead towards an improved policy.

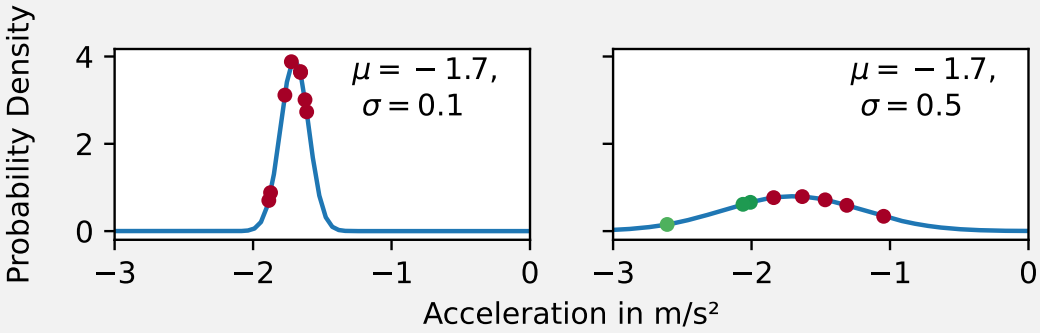


Figure 5.7: When the policy standard deviation σ is too low, the policy can get trapped in areas with low rewards.

One solution to this problem is to simply enforce a minimal policy standard deviation σ_{min}. Then, as shown in the right plot of Figure 5.7, it can be assured that the policy continues to sample good actions. This stabilizes the training, but comes at the price that the policy needs to ensure a certain distance from bad actions.

Most of the training runs in Figure 5.6b eventually collapse due to this phenomenon. To counteract this issue, this work always enforces a minimum action noise of $\sigma_{\text{min}} = e^{-2}$ for both, acceleration and steering.

An alternative interpretation on this can be given through the lens of the policy gradient estimation mechanism. When the exploration noise is low, the gradient of the returns with respect to the policy parameters is approximately the analytical gradient, whereas larger exploration noise means that the gradient is effectively estimated in a larger area around the policy operating point, similar to the distinction between the extended and unscented Kalman filter [WV00]. While the analytical gradient is locally more accurate, it can get trapped in local minima during gradient descent. In contrast, the stochastic gradient is less prone to this, because the larger effective area of the gradient estimate facilitates escaping local minima.

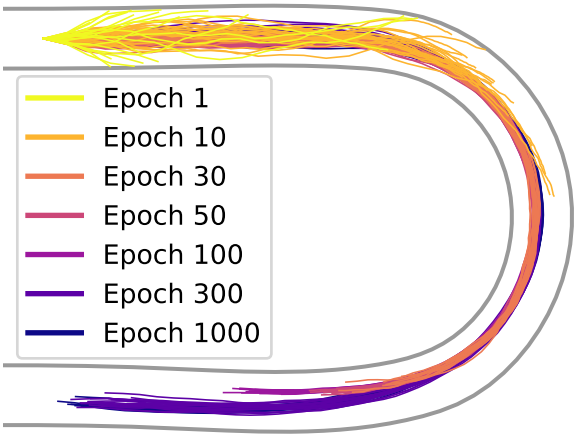
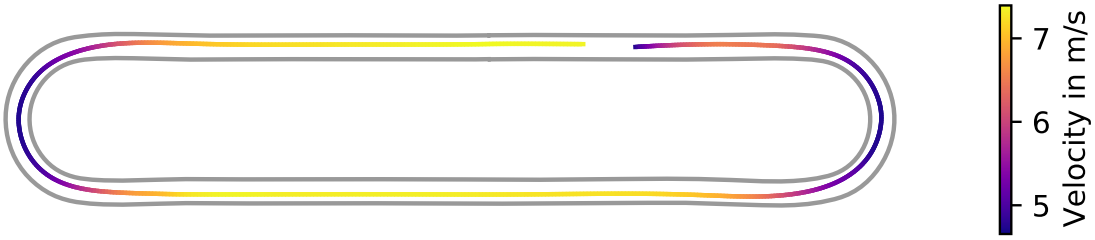


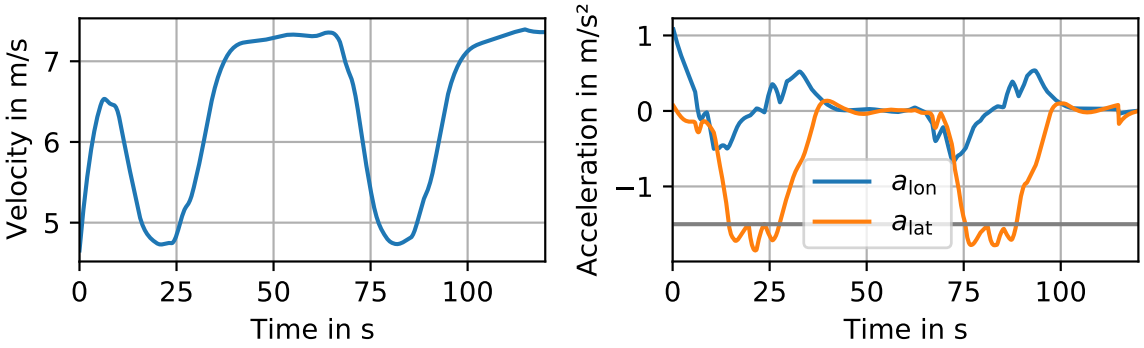
Figure 5.8: Trajectories from different training epochs.

Training Progress in Trajectory Space Another perspective on the training progress is given in Figure 5.8, where realizations of policies from different training epochs are shown. For visualization purposes, all trajectories are initialized at the same position with equal initial speed, heading and lateral offset. During early training, the policy acts randomly and often leaves the track. The best trajectories determine how the policy changes. After 30 training epochs, all trajectories stay on the track. Then, the speed is increased to maximize the reward. Training has converged after approximately 300 training epochs, and the trajectories after 1000 epochs look similar.

As sampling from action distribution, parameterized by the policy, is a trick to obtain the gradient estimate required in all presented policy gradient algorithms, sampling from the policy is disabled when executing the policy for testing. Instead, the mean action μ_a is selected. With this, no vehicle leaves the track in 200 random test situations for any final policy from the 7 training runs. One deterministic realization of the policy from Figure 5.8 is shown in Figure 5.9. The policy has learned to approximately comply with the optimal acceleration a_{lat}^* induced by the reward function by adapting its velocity before the curves, and by maximizing the turn radius through the selected path. On the straight segments, it drives faster to obtain higher rewards for the velocity. The applied longitudinal accelerations are below $0.5\,\mathrm{m/s^2}$, except from the starting point. Both accelerations are not smooth but exhibit small jumps. This can be explained by the policy acting purely reactive and without memory, based exclusively on the current observation. A jump in any of the observation features can cause a jump in the actions. Thus, the learned policy would probably not be suited to control a real-world vehicle without any smoothing. However, the kinematic model integrates the actions and therefore acts as a low-pass, rendering this point uncritical for the positional prediction.



(a) Vehicle track and speed: The agent reduces lateral accelerations by braking before curves. Also, by entering at the outside edge of the turns it effectively increases their radius, thereby further reducing the lateral acceleration.



(b) Speed, lateral and longitudinal accelerations. The lateral acceleration in the turns is close to the ideal $1.5\,\mathrm{m/s^2}$ (dark gray).

Figure 5.9: An agent drives one lap through the oval track with the final, deterministic policy. The vehicle drives clockwise and starts in the upper right corner.

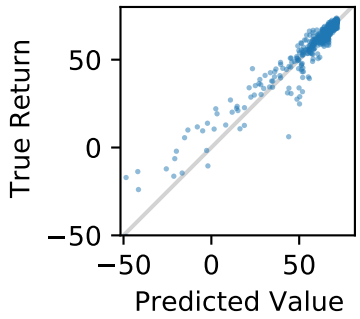


Figure 5.10: Predicted values $\hat{V}_\psi(o)$ and true infinite-horizon discounted return $R(\tau_{k:\infty})$. The value network predicts values close to the true return in most cases. This means that it can anticipate the expected future returns based on the current observation. Hence, the related GAE provides a good guidance during training on which actions to select to reach promising states and the corresponding observations.

Value Network and GAE The value network and the GAE significantly influence the training progress. The value network $\hat{V}_\psi(o)$ is trained to predict the future infinite-horizon GAE-smoothed returns (5.25) after making an observation o. Based on this, the GAE (5.24) is computed to assess whether an action is beneficial, compared to the estimated value, or not. To assess whether the value network has learned to correctly predict the expected return after an observation, Figure 5.10 shows the predicted value and the true return from the initial observation of 500 simulated trajectories. The trajectories are simulated for 500 time steps to approximate the infinite-horizon return. The value network $\hat{V}_\psi$ accurately predicts the true return in most cases. The remaining differences are caused by both, the stochastic sampling of actions during the policy execution, and erroneous value estimates. Adverse initialization with unsuitable velocities can lead to low returns, as the policy needs to apply large accelerations to reach an appropriate speed.

For one vehicle driving on the circular track according to the learned policy, Figure 5.11 shows the reward r_k of each transition along the trajectory, the value estimate $\hat{V}_\psi(o_k)$, the GAE return $R_k^{\mathrm{GAE}(\lambda,\gamma)}$, and the generalized advantage estimate $\Psi_k^{\mathrm{GAE}(\lambda,\gamma)}$. The GAE return (5.25) is an approximation of the infinite-horizon discounted sum of future rewards. During training, the value network $\hat{V}_\psi$ is trained to predict the GAE return.

The left plot shows a monotonous driving situation. The GAE return is approximately constant, as the expected infinite-horizon discounted return in the test track only differs marginally, depending on whether a vehicle is approaching the next curve, or has just left it. The value network $\hat{V}_\psi(o_k)$ predicts the GAE return $R_k^{\mathrm{GAE}(\lambda,\gamma)}$ relatively accurately. The GAE $\Psi_k^{\mathrm{GAE}(\lambda,\gamma)}$ is the difference between the GAE return and the predicted value, see (5.25). When the return is higher than the expected value, the GAE is positive, and the corresponding action is reinforced during training. In the right plot, the vehicle leaves the track at time step 310, and therefore receives a reward of -100 at that time step. The value network detects the

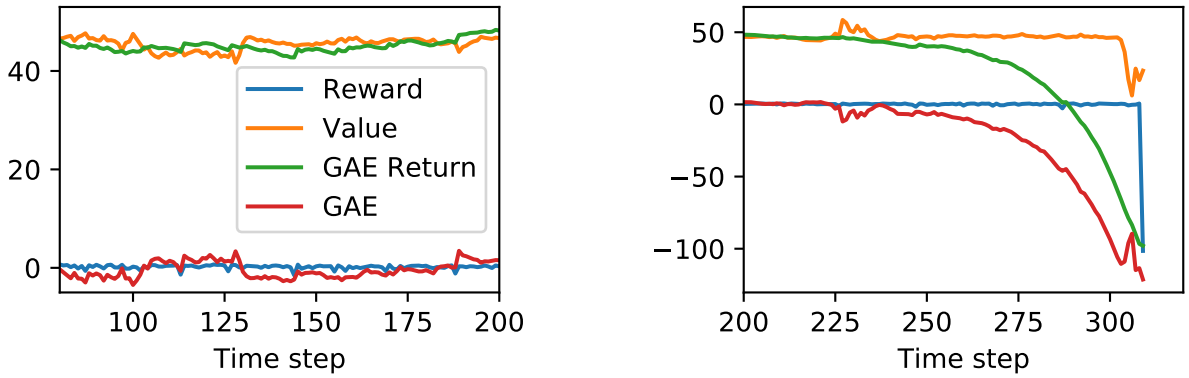


Figure 5.11: Rewards r_k, value estimate $\hat{V}_\psi(o_k)$, GAE return $R_k^{\text{GAE}(\lambda,\gamma)}$ and GAE $\Psi_k^{\text{GAE}(\lambda,\gamma)}$ along one trajectory. The left figure shows the values during regular driving, whereas the right figure shows the values before a collision at time step 310. The y-axes are scaled differently for better visibility. The collision is foreseen by the value network (orange) a few time steps before it happens, leading to a reduced predicted value. The GAE return is computed recursively backwards from the point of the collision, and penalizes all states that lead to the collision. Hence, the policy is discouraged from executing similar actions in the next training run. Similarly, the value network is trained to predict the imminent danger earlier, as its training goal is to predict the GAE return.

imminent lane departure a few steps before, as the vehicle is approaching the lane boundary, and predicts low values. The GAE penalizes all actions from approximately time step 240, such that the policy learns to avoid the actions that lead to the departure. Depending on the discount factor γ, the GAE penalizes more or fewer time steps before the departure. This mechanism is fundamental for the policy to learn the long-term effects of its actions, such that it learns to avoid actions that lead to low rewards multiple time steps ahead. Similarly, the policy learns to select actions that increase the obtained rewards in the future.

Implementation Multiple implementations for PPO and GAE exist and are available under permissive licenses, e.g., [Lia+18; Raf+21]. For the following reasons, the algorithms have been implemented independently in this thesis nevertheless: First, this enables gaining a better insight into the internals of the algorithms, for example demonstrated in Figure 5.10 and 5.11. Secondly, this allows for an extendable implementation of the algorithms that not only supports single-agent RL, but also multi-agent RL as well as the integration of the algorithms into the IRL framework presented in the next chapter. Still, both implementations [Raf+21] and [Lia+18] served as a reference during the implementation in this work.

5.3.3 Reducing the Training Time

One important aspect of RL training is the overall time required to obtain a policy. Two main factors influence the training duration: The number of training epochs and the duration of one

epoch. The following paragraphs conclude the most important choices made in this thesis to reduce the training time.

Number of Training Epochs Policy gradient methods iteratively approach the optimal policy by adapting the parameters according to the current gradient estimate.

A good initial policy reduces the iterations required until the maximum is reached. To this end, the policy is initialized to select on average an acceleration of $0\,\mathrm{m/s^2}$ and a steering angle of $0\,\mathrm{rad}$.

To minimize the number of iterations, the gradient estimate needs to be accurate. Therefore, the idea of advantage estimates is used to clearly identify the actions that lead to higher returns than the current policy. To compare the expected return to the actual return, it is required to maintain an estimate of the value of the current observation. GAE is used to reduce the variance of the standard advantage estimate, as described in Section 5.1.2.

Lastly, the step in the direction of the gradient needs to be as large as possible, without leaving the area where the current gradient estimate is valid. The PPO algorithm achieves this goal by ensuring that the change in the policy parameters leads to a limited change in the actions that the policy selects, as discussed in Section 5.1.3.

Duration of One Training Epoch The majority of computational resources while executing Algorithm 4 is spent while executing the current policy in the simulation to collect the experience dataset $\mathcal{D}_{\mathrm{RL}}$. Training the policy and value networks is comparatively fast, because of the relatively small size of the networks, c.f. Figure C.1 and Table C.3 in the appendix.

Thus, reducing the duration of one training epoch amounts to reducing the time spent for collecting experiences during each training epoch. This is achieved by selecting an appropriate number of simulated trajectories, simulation steps and simulation step size. More simulated trajectories improve the robustness of the learned policy, and enable a more accurate gradient estimate. More simulation steps allow the policy to experience the long-term effects of its actions. For example, an agent learns that applying a longitudinal acceleration is penalized immediately, but results in a higher return due to the accumulated rewards for the higher speed. Clearly, increasing the number of simulated trajectories and time steps increases the simulation duration. Finally, a larger simulation step size also allows the agent to experience the long-term effects of its actions, but reduces their ability to react to observations by effectively increasing the reaction time. The values used for training are also listed in Table C.3.

Moreover, to generate experience samples as fast as possible, the simulation environment is implemented such that multiple simulations are effectively running in parallel, as discussed in

Section 3.4. Compared to a sequential execution of the simulation, this reduces the required simulation time by approximately two orders of magnitude.

5.4 Experiments: Multi-Agent Reinforcement Learning

After demonstrating how the combination of PPO and GAE solves a single-agent RL problem, this section is concerned with transferring the method to a multi-agent problem. To this end, a POSG is formulated in which each agent in a traffic situation has the goal to maximize its reward. For the solution, parameter sharing is used to learn one single policy that is applied by all agents. Training takes place in a roundabout situation with a high degree of interaction between the agents.

The questions investigated in this experiments section are:

- Which modifications are required to adapt the previously demonstrated single-agent approach such that a policy that can control multiple vehicles in a roundabout traffic situation is learned?
- Which modifications to the cost function are required to produce a policy that makes progress but avoids collisions with other vehicles?
- Which further modifications are required to produce more plausible behavior, i.e., respecting right-of-way rules and maintaining safe distances to other vehicles?

5.4.1 Setup of the Partially Observable Stochastic Game

To foster interesting interactions between agents, a roundabout situation is used for training. For this, the map of the right roundabout in Figure 3.4 is used. The right-of-way situations at the roundabout entrance require incoming agents to decide whether they enter before or after inner-roundabout vehicles, and inner-roundabout agents need to adapt their velocity when an incoming vehicle squeezes in before them. To enable the agents to interact with each other, other vehicles need to be represented in the observation. Thus, the full observation vector from Table 3.1 is used.

Handling Collisions Each agent is assigned rewards via the same reward function (5.41) that was used for single-agent training. In the multi-agent setting, an additional reason for early termination of the simulation of one agent exists besides leaving the road: A collision with another agent. In this case, the two colliding agents are removed from the simulation. The simulation is continued for the remaining agents, which is efficient because they can continue to gather experience.

This leads to the question of how the collision should be penalized. Often, one agent causes the collision, whereas the other agent is not to blame, for example, when a vehicle that

waits at the entrance of the roundabout is hit rear-end by another vehicle. Considering the core idea of policy gradient RL—iteratively increasing the likelihood of selecting actions with high rewards compared to those with low rewards—penalizing the stationary vehicle in this situation would be harmful, as it introduces noise into its rewards: Usually, stopping at the roundabout is a reasonable action that yields average rewards, but when a rear vehicle strikes the stationary vehicle, the same sequence of observations and actions would lead to a penalty. This creates contradicting training signals and therefore potentially destabilizes the RL training.

For this reason, only agents that could have avoided a collision by acting differently are penalized. Agents that are not at fault in a collision receive no penalty, and their return and advantage estimate are computed as if their simulation continues indefinitely, as described in Section 5.1.2. These early terminations are treated equally to other non-culpable early terminations, such as when the simulation horizon is reached.

In concrete terms, this requires an implementation of simplified traffic rules in the simulation to determine whether an agent is culpable in a collision. The rules are simple: In a rear-end collision, only the rear agent is culpable. In a right-of-way collision within $5\,\mathrm{m}$ after one vehicle entering the roundabout, both agents are culpable.[7] Despite the formal precedence of one agent over the other, both should act to avoid a collision.

The penalty assigned in case of a culpable collision

$$\mathcal{R}_{\text{coll}}(v) = -20 - \omega_{\text{coll,vel}}|v|/[\mathrm{m/s}] \tag{5.44}$$

is dependent on the speed of the agent. Compared to a constant penalty, this leads to an agent preferring collisions at lower speeds to collisions at higher speeds, and thereby provides a direction of change for the policy to ultimately avoid the collision entirely. If only constant penalty was used, the agent would only learn to brake before a collision once it experiences that it can avoid a collision by braking, after selecting a sequence of sufficiently large braking actions by chance. Thus, this pseudo gradient on the collision penalty speeds up the training.

How large should the weight $\omega_{\text{coll,vel}}$ of the speed-dependent collision penalty be? Even if a collision cannot be avoided, emergency braking should be beneficial to collide with the lowest possible speed. Emergency braking with $-7\,\mathrm{m/s^2}$ is penalized via the longitudinal acceleration penalty in (5.41) with $\omega_{\text{acc,lon}} a_{\text{lon}}^2/[\mathrm{m/s^2}] = -2.36$ per step, where the weight is $\omega_{\text{acc,lon}} = 1/(9\ln(10))$. The step size is $\Delta t = 0.2\,\mathrm{s}$. The decrease in speed during this time is $\Delta v = \Delta t\; a_{\text{lon}} = -1.4\,\mathrm{m/s}$. To ensure that braking is always beneficial, the penalty $\omega_{\text{acc,lon}} a_{\text{lon}}^2$ due to the braking must be overcompensated by the reduced penalty for a collision with

[7] Initially, only the agent that violated the right-of-way was penalized. However, this lead to inner-roundabout agents completely ignoring incoming vehicles, even when slightly braking would resolve the situation. To keep things simple, introducing different penalties depending on the degree of culpability, similar to traffic law, is not explored here.

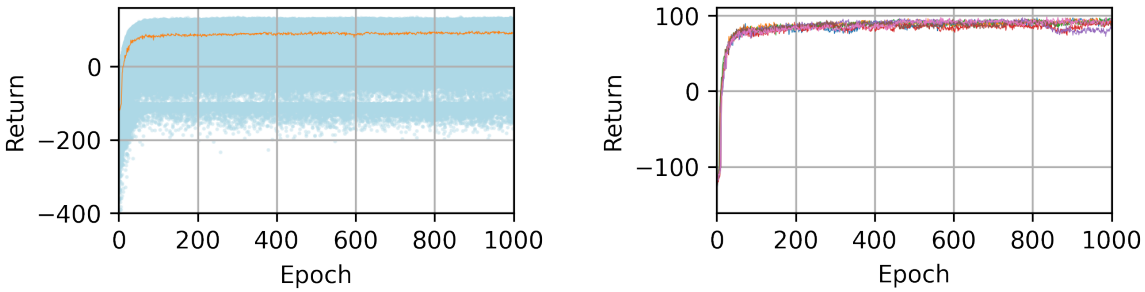
reduced speed. It follows, that $\omega_{\text{coll,vel}}$ must be larger than $-2.36/\Delta v \approx 1.7/[\mathrm{m/s}]$. Then, the reduction in reward by braking is immediately compensated by the decreased collision penalty. For simplicity, it is set to $\omega_{\text{coll,vel}} = 2/[\mathrm{m/s}]$.

Situation Initialization To train a robust policy, it must be ensured that agents are confronted with a large variety of situations during training. During each training epoch, 50 situations are simulated. The situations are initialized randomly at each epoch. To capture different traffic densities, situations are initialized with a random number of 1 to 17 agents. The exit that each agent needs to take is also selected randomly. Equal to the single-agent case, the initial heading and lateral offset from the lane center is drawn randomly. Agents are placed on fixed positions along the four lanes leading to the roundabout every $16\,\mathrm{m}$, plus a random offset within $\mathcal{U}(0,7)\mathrm{m}$. The resulting minimum initial bumper-to-bumper distance is approximately $4\,\mathrm{m}$. In 70% of all situations, each vehicle is assigned a speed from the uniform distribution $\mathcal{U}(0,20)\mathrm{m/s}$. As a result, vehicles are occasionally initialized in situations with low distances and large delta velocities to their preceding vehicle, which enables them to learn how to act in these critical situations. To train the agents to resolve situations where everyone is in standstill, all vehicles start with an initial speed of $0\,\mathrm{m/s}$ in 15% of all situations and with $\mathcal{U}(0,3)\mathrm{m/s}$ in the remaining 15%. Some exemplary initial situations are shown in Figure C.2.

5.4.2 Learning to Drive

Most of the settings for training correspond to the single-agent experiment described in Section 5.3.2. The policy and value networks have a larger number of inputs, because more observation features are used. Except from this, their architecture remains unchanged and is shown in the appendix in Figure C.1. In contrast to single-agent learning, more experience from an average number of 500 agents is aggregated during each training epoch. The approximate tenfold increase in simulated trajectories compared to single-agent learning leads to an increased training time of 3 to $4\,\mathrm{h}$ for 1000 epochs. The training is repeated seven times to ensure reproducibility. All training parameters are listed in Table C.4.

The training progress is visualized in Figure 5.12. The performance between epochs fluctuates more strongly than in the single-agent case due to the non-stationarity of the multi-agent problem. Still, the training reproducibly reaches high returns after approximately 400 epochs. Remarkably, all training runs learn approximately at the same speed during early training. Compared to the single-agent training shown in Figure 5.5b, this can be explained by the much larger number of experiences aggregated per epoch, which makes the estimation of the gradient direction more robust.



(a) Median $\overline{R(\tau)}$ and single-epoch returns $R(\tau)$ (scattered, cyan) of one single training run.

(b) Median returns $\overline{R(\tau)}$ during seven independent training runs.

Figure 5.12: Undiscounted returns during MARL training. During each epoch, the returns of approximately 500 trajectories are evaluated and scattered in (a). To investigate the reproducibility, the median returns of seven independent training runs are shown in (b).

Collisions occur throughout the training. At later training stages, these collisions occur mostly due to adverse initializations, e.g., when one vehicle is initialized with little distance to its preceding vehicle at a much higher speed and cannot avoid the collision anymore. Decreasing the maximum initial speed during the simulation avoids almost all collisions. However, this should not be implemented at training time, as the robustness of the models crucially depends on agents experiencing collisions to learn to avoid them. As the reward function is unchanged except for collisions, the return values can be directly compared to those in the single-agent case depicted in Figure 5.5. The median return is significantly lower in the multi-agent case, because agents need to drive slower or wait at the roundabout entry to avoid collisions. Executed in the test and unseen situation from Section 4.3, the learned policy shows lower failure rates than the best multi-step policies. The collision rate is between 0.5 and 1.2% in both situations. No vehicle leaves the track in both situations. A more thorough comparison of multi-step and RL-based policies follows in the next chapters.

Demonstration of the Learned Policy A demonstration of the learned policy is provided in Figure 5.13. The images show a top view of the simulated traffic situation in chronological order. The simulation time is shown in the top right corner. The agents have learned to stay on the track, and accelerate and steer to follow their intended route. Moreover, they brake to avoid rear-end collisions. Close to the four roundabout entries, both, the inner-roundabout and the entering vehicle slow down to resolve the merge situation. Interestingly, as visible for vehicles #5, #10, and #6, the agents have learned to negotiate the right of way differently than in real traffic: Here, the inner-roundabout vehicle brakes to let an oncoming vehicle enter. From a macroscopic perspective, this leads to a congestion at the roundabout.

Considering the reward function, this observation is not surprising: Each agent is rewarded for its own progress along the track. Cooperation emerges exclusively from the self-interest of not colliding. If the agents are to handle right of way as they do in the real world, they must be

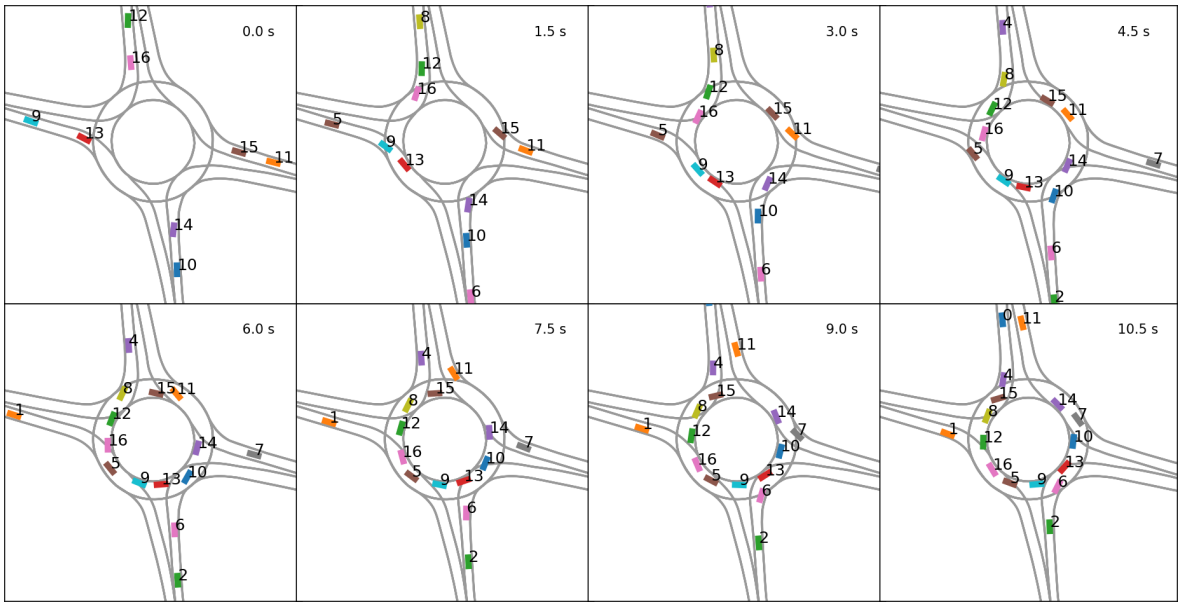


Figure 5.13: Execution of the multi-agent reinforcement learning policy by all agents. While the agents have learned to follow their route and avoid collisions, they have learned an inverted right-of-way logic, where incoming vehicles have priority over inner-roundabout vehicles.

rewarded for doing so. In addition to the reversed right of way, the agents only maintain little distances to other vehicles. A more realistic modeling should reflect that drivers typically do not fall below certain time gaps.

Improved Traffic Modeling For a more accurate model of traffic situations, three additional reward terms are introduced: 1.) If the time gap to the preceding vehicle falls under a threshold, an agent is penalized. 2.) If the absolute distance to the preceding vehicle falls under a threshold, an agent is penalized. 3.) Agents are rewarded for respecting right of way rules. Vehicles about to enter the roundabout receive a penalty when the distance or time gap of the closest vehicle with right of way to them is below a threshold.

Specifically, this leads to the reward function

$$\mathcal{R}(v, a_{\text{lon}}, a_{\text{lat}}) = \omega_{\text{vel}} \ln(v/[\text{m/s}] + \epsilon) - \omega_{\text{acc}}(a_{\text{lon}}/[\text{m/s}^2])^2 - \omega_{\text{acc}}(a_{\text{lat}}/[\text{m/s}^2])^2 + \mathcal{R}_{\text{offtr}} + \mathcal{R}_{\text{coll}}(v) + \mathcal{R}_{\text{tg}} + \mathcal{R}_{\text{dist}} + \mathcal{R}_{\text{row}} \quad (5.45)$$

with the newly introduced penalty for the timegap Δt

$$\mathcal{R}_{\text{tg}} = -10 \text{ if } \Delta t < \Delta t_{\text{min}}, \quad (5.46)$$

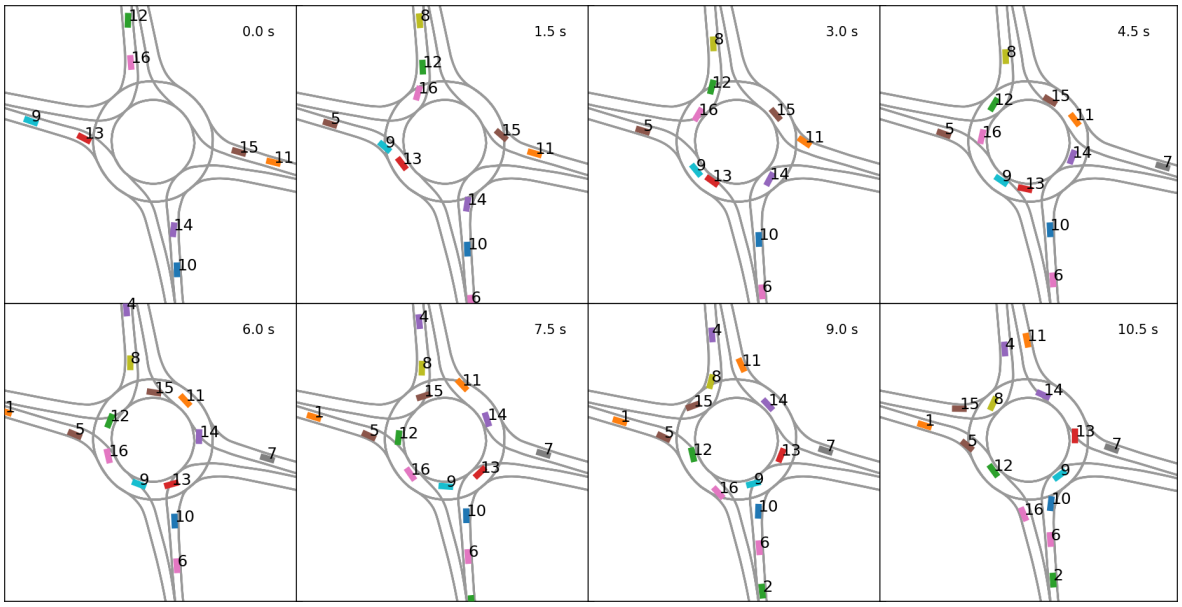


Figure 5.14: Execution of the cooperative MARL policy by all agents. With the cooperative reward function, the agents learn the correct right-of-way rules, compared to Figure 5.13.

the penalty for the distance to the preceding vehicle d_{pre}

$$\mathcal{R}_{\text{dist}} = -10 \text{ if } d_{\text{pre}} < d_{\text{min}}, \tag{5.47}$$

and the right-of-way penalty,

$$\mathcal{R}_{\text{row}} = -10 \text{ if } d_{\text{ccv,ego}} < d_{\text{ccv,ego,min}} \text{ or } \Delta t_{\text{ccv,ego}} < \Delta t_{\text{ccv,ego,min}}, \tag{5.48}$$

which is assigned when the distance of the closest conflicting vehicle $d_{\text{ccv,ego}}$ to the agent is too low or when the time gap of the closest conflicting vehicle to the agent $\Delta t_{\text{ccv,ego}}$ is too low.

This cooperative reward function replaces the previously used reward function. Starting from the same initial situation as in Figure 5.13, the evolution of a traffic situation simulated with the cooperative policy is depicted in Figure 5.14. Now, the agents respect the right of way and only enter the roundabout if a sufficiently large gap exists, as for example visible for vehicle #5 and #8. Moreover, agents keep larger distances and time gaps to each other.

5.5 Modeling Individual Driver Traits

The result of all previous experiments in this chapter is one single policy that maximizes the reward function. When a situation is simulated, the same policy is executed by all drivers. The resulting behavior is homogeneous: Faced with the same observation, every agent behaves the same. However, real drivers exhibit a variety of behaviors, because they have differing

preferences. Some drivers prefer to keep more distance to the vehicle in front than others. Aggressive drivers might prefer higher accelerations and velocities than careful drivers.

This leads to the last question investigated in this chapter:

- How can multi-agent RL be extended to model *heterogeneous* driving behavior?

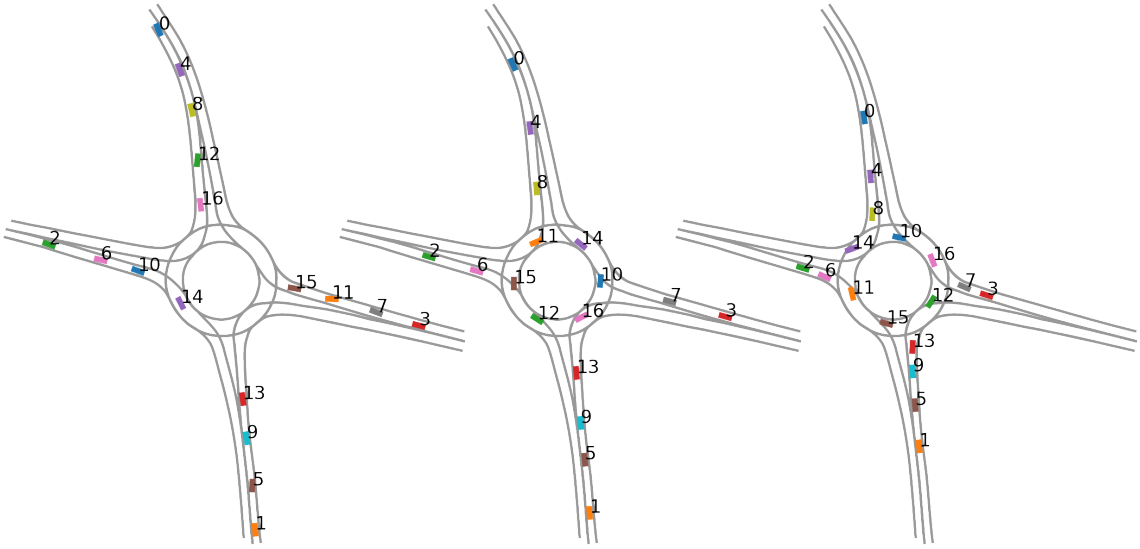
To this end, the following section proposes an approach of learning a heterogeneous driving policy that can adapt its behavior to differing preferences.

One way to obtaining different behaviors is to use Independent Proximal Policy Optimization (IPPO) without parameter sharing. Thereby, multiple policies that maximize different reward functions are trained by letting the agents with different policies interact in the simulated traffic situation. However, this reduces the amount of experiences available for training per policy, compared to training one single policy using parameter sharing. If N different policies are trained by letting every agent execute one of the policies, only $1/N$ of the total simulated experiences are generated by each policy and can be used for its improvement. Moreover, this approach is susceptible to instabilities, because each agent influences the perceived environment dynamics of the other agents, and bad policies can thereby impair the learning of the other agents.[8]

Instead, this thesis proposes to learn one single flexible policy, which exerts different behaviors depending on its inputs. Three preferences $\rho = (\Delta t_{\text{min},i}, d_{\text{min},i}, \omega_{\text{acc},i})$ are assigned to each agent. These define the individual thresholds for the minimum timegap in (5.46) and minimum distance in (5.47), and the individual weight of the acceleration cost in (5.45). Depending on their values, the reward function changes, and with it the optimal behavior. To inform the policy about the preferences, these are used as additional input variables besides the observation features of the traffic situation from Table 3.1. This enables the policy to learn the relation between the preference variables and desirable behavior, e.g., to avoid time gaps below the individual threshold $\Delta t_{\text{min},i}$.

During training, each agent is assigned a random preference vector with $\Delta t_{\text{min},i} \sim \mathcal{U}(0.5, 2)\text{s}$, $d_{\text{min},i} \sim \mathcal{U}(1, 6)\text{m}$, and the optimum lateral acceleration $a^*_{\text{lat},i} \sim \mathcal{U}(1.5, 4)\text{m/s}^2$, which is converted to the weight of the acceleration $\omega_{\text{acc},i}$ according to relation (5.40). The preferences of each agent remain constant throughout one simulated traffic situation and are reshuffled before the next simulation in the next training epoch. In this way, a flexible policy is learned on the basis of the experiences of all agents.

[8] In multi-agent settings, agents are often competing for the same resources, and the policy which figures out the fastest has a strong and continuous advantage over the other policies. If for example one policy learns to ignore the right of way by entering the roundabout and forcing inner-roundabout vehicles to brake, the inner-roundabout vehicles cannot learn to insist on their right of way, as they would collide otherwise. This is a Nash-equilibrium, from which it is hard to escape, as no agent can improve its return by changing its policy when the other agent sticks to its policy.



(a) Initial situation **(b)** Careful parameters **(c)** Aggressive parameters

Figure 5.15: Effect of simulating the same initial situation with aggressive and careful driving parameters for every agent: After $15\,\mathrm{s}$ of simulation, the distance between the vehicles waiting at the entry lanes is clearly distinguishable. Vehicles have made more progress in the aggressive case, as they tolerate higher velocities and smaller time gaps.

After training, the behavior of the policy can be adapted at runtime via the preference input of the policy. To demonstrate the effect, the evolution of 50 initial traffic situations is simulated twice. The first set of simulations is performed with aggressive driving preferences $(\Delta t_{\text{min}} = 0.5\,\mathrm{s}, d_{\text{min}} = 1\,\mathrm{m}, a^*_{\text{lat}} = 4\,\mathrm{m/s^2})$ for each agent, the second set of simulations is performed with careful driving preferences $(\Delta t_{\text{min}} = 2\,\mathrm{s}, d_{\text{min}} = 6\,\mathrm{m}, a^*_{\text{lat}} = 1.5\,\mathrm{m/s^2})$ for each agent.

Figure 5.15a shows one initial traffic situation in which every agent has the goal of leaving the roundabout at the exit where they entered. Figure 5.15b shows how the situation evolves when every agent executes the policy with the careful preferences. Figure 5.15c shows the situation when every agent behaves aggressively. Clearly, the policy has learned to adapt its standstill distance to the preference value.

A histogram of the time gaps that emerge during the simulation of the 50 traffic situations is depicted in Figure 5.16. The agents have learned to drive with time gaps above their minimum time gap. The preference value has a clear impact on their behavior. The learned policy maintains a relatively large safety margin towards the threshold. This can be explained by the large penalty of -10 that is assigned when the time gap falls under the threshold. Using a smaller penalty during training leads to policies that tolerate undercutting the threshold briefly, e.g., to merge into the roundabout directly behind another vehicle.

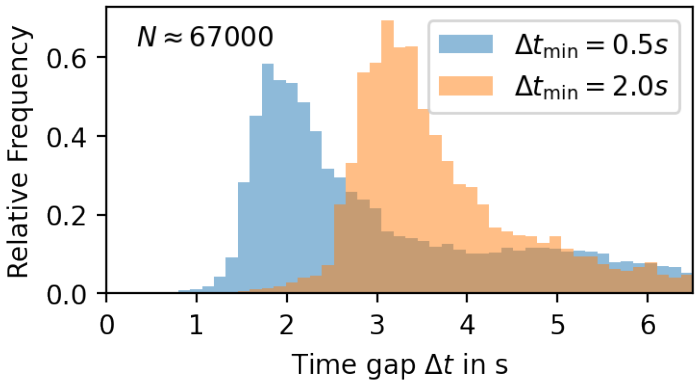
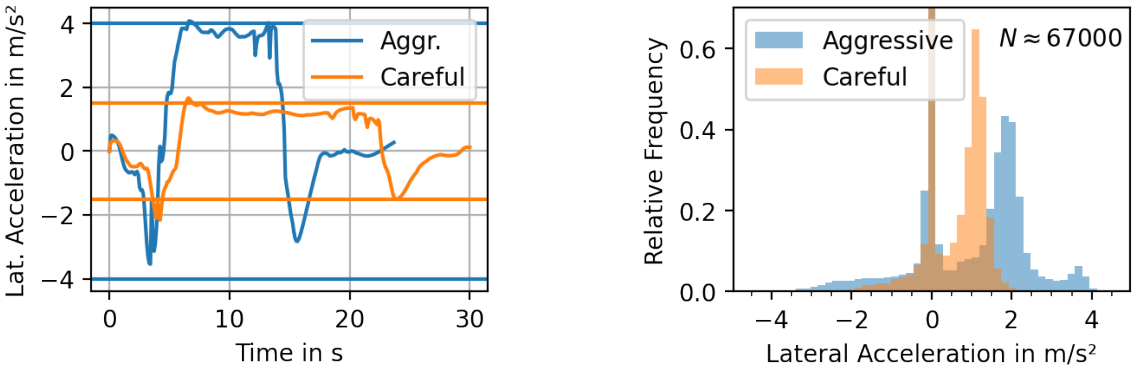


Figure 5.16: Time gaps during simulation with careful and aggressive driving parameters



(a) Lateral acceleration of one agent driving one lap through a lonely roundabout

(b) Histogram of lateral accelerations of all simulated agents

Figure 5.17: Lateral acceleration

Finally, the effect of the lateral acceleration penalty on the behavior is analyzed. A situation is considered where one lonely vehicle drives a full circle through the roundabout. The lateral accelerations that the agent experiences during this simulation is plotted in Figure 5.17a. For the aggressive agent, the weight of the lateral acceleration penalty $\omega_{\text{acc,lat}}$ is set such that the tradeoff between velocity reward and lateral acceleration penalty is optimal at a lateral acceleration of $4\,\text{m/s}^2$ according to (5.40). The weight for the careful agent is set such that its optimal acceleration is $1.5\,\text{m/s}^2$.

During the simulation, the aggressive agent drives with an average speed of $7.6\,\text{m/s}$ inside the roundabout, leading to a lateral acceleration slightly below the optimal $4\,\text{m/s}^2$. The careful agent drives approximately $4.3\,\text{m/s}$, leading to a lateral acceleration of $1.3\,\text{m/s}^2$, also slightly below its optimum. Due to the higher speed, the aggressive agent completes its lap through the roundabout significantly faster than the careful agent.

Figure 5.17b shows a histogram of the lateral accelerations that all agents experience during the simulation of the 50 traffic situations. The situations are initialized with varying traffic

densities, as shown in Figure C.2. The effect of the aggressive or careful parameterization is clearly visible. In the traffic situations with low traffic density, the agents drive such that their lateral acceleration approaches the optimal value. At higher traffic densities, the lateral acceleration is typically below the optimal value, because the vehicles cannot select their speed freely due to the presence of other vehicles. The lateral accelerations rarely exceed the optimal value of $4\,\mathrm{m/s^2}$ or $1.5\,\mathrm{m/s^2}$.

These three experiments show that the proposed method enables learning a flexible driving policy that can exert different behaviors at execution time. The policy can be used for modeling different types of drivers. The implementation of additional preference variables is conceivable, but omitted here for brevity.

To model uncertainty about the evolution of a traffic situation, multiple predictions of the same initial situation can be made with random preference parameters for each agent. Moreover, using a Bayesian filter such as an unscented Kalman filter or a particle filter, the preference vector of an observed vehicle could be estimated online by comparing predictions under different preferences to the actual behavior of a driver. This could be used to obtain a driver-specific prediction model. For example, a driver that has been observed to drive carefully could be predicted to continue driving carefully in the future. Similar ideas have been explored in [HSD17; **Bey+21a**], where the parameters of the car-following IDM [THH00] are estimated. Sadigh et al. [Sad+18] goes one step further and proposes to actively gather information on another driver by selecting actions oneself that clarify the preferences of the other driver. For example, the authors present a driving strategy where an automated vehicle first nudges into the lane of another vehicle to probe its attentiveness, and only merges when the other vehicle reacts. Compared to the simplistic driver models used in [HSD17; Sad+18; **Bey+21a**], this chapter proposes a method to learn policies that handle longitudinal and lateral control while interacting with multiple relevant surrounding vehicles simultaneously.

5.6 Conclusion

RL is a group of methods with the goal of finding policies that maximize the aggregated reward of an agent that interacts with an environment. This chapter described the idea of policy gradient RL, which forms the basis of algorithms that learn policies in partially observable environments with continuous observation and action spaces. Based on this, two improvements from the literature are described: GAE reduces the variance of the gradient estimate and PPO determines the maximum step that can safely be used during the gradient ascent procedure.

The key contribution in this chapter lies in the application of GAE and PPO, which are typically used in robotics control tasks, to the problem of driver behavior modelling. To model traffic situations, a multi-agent variant of PPO is implemented from scratch. The implementation

enables the training of a cooperative policy that controls all vehicles. Moreover, a method to model heterogeneous driving behavior is proposed, whereby a single flexible policy is learned that can change its behavior, depending on special preference inputs. The implementation also forms the base for further extensions of the algorithms that are presented in the next chapter.

Besides demonstrating the application of the algorithms to driver behavior modeling, the following describes insights into the algorithms that are important for successful, stable and fast training.

Single-Agent Reinforcement Learning The combination of PPO and GAE is applied to the problem of learning a single-agent driving policy. To this end, all ingredients to the RL problem are described: The simulation environment and the observation vector remain unchanged, compared to the previous BC chapter. A *new reward function* is derived from the assumption that a comfortable lateral acceleration exists as a tradeoff between the reward for progress along the track and the penalty for the lateral acceleration. Hence, the optimal behavior under this reward function would be to adapt one's velocity to the curvature of the road, such that this comfortable lateral acceleration is maintained.

The experiments visualize how the policy iteratively learns to stay on the track and to increase its rewards. The central mechanism behind this is

- the random selection of actions,
- evaluation of the rewards of the resulting trajectory sequences,
- computation of the GAE, i.e., which actions lead to higher long-term rewards than expected, and finally
- increasing the probability of selecting actions with positive GAE values and decreasing the probability of selecting actions with negative GAE.

It is shown that by repeatedly executing this procedure, the policy learns to adapt its velocity in curves such that the lateral acceleration is close to the theoretical optimum.

Thereby, an insight into the importance of *exploration noise* is gained: If the variance of the probability distribution according to which the actions are selected is too low, the policy eventually stops to experience bad behavior such as leaving the track during training. Shortly after, its performance collapses. Similar to BC, the lesson to be learned here is that it is important to make good and bad experiences during training—good experiences are important to further improve the performance, and bad experiences such as leaving the track are important to ensure the robustness of the policy.

Alternatively, using smaller magnitudes of the exploration noise can be interpreted as making a more *local* estimation of the policy gradient. In contrast, larger noise leads to a more *global*

gradient estimate. This is similar to the difference between the extended and the unscented Kalman filter. The more global gradient estimate is better equipped to handle regions of sharp change in the reward, such as when a vehicle is either penalized heavily for crossing the road boundary or moderately rewarded for staying within it.

The training progress is visualized in the trajectory space. Moreover, the mechanism of GAE is demonstrated: For regular driving of the trained policy, the GAE value fluctuates around 0, meaning that the policy will only be adapted marginally for the corresponding observations. In case of a collision, the GAE becomes strongly negative multiple steps before the incident, meaning that the policy avoids select all actions that lead to the crash after the next training epoch.

Multi-Agent Reinforcement Learning The second part of the chapter is concerned with MARL. Teaching one vehicle to drive and interact with other vehicles introduces a chicken-and-egg problem: To learn a policy that properly interacts with other vehicles, the other vehicles need to follow a policy that properly interacts with oneself. To resolve this dilemma, the idea of *parameter sharing* is picked up: All vehicles learn simultaneously in the same simulation, and each vehicle is controlled by a copy of the same policy. The policy, in turn, is trained based on the experiences collected by all agents in this simulation. It is demonstrated that parameter sharing enables the application of the single-agent algorithm to the multi-agent problem. Hence, the intuition gained from single-agent RL can be directly transferred to MARL.

Nevertheless, the multi-agent problem is theoretically more demanding, as it entails *non-stationary environment dynamics* during training: From the perspective of one agent, the environment dynamics change between training epochs, because the changing policy controls all surrounding agents. To ensure stable training, one key takeaway is that the policy should change in small steps during training, such that the environment dynamics can be considered quasi-stationary. The PPO algorithm has been selected because it explicitly limits the amount that the policy changes between training epochs via the parameter ϵ. Moreover, a small learning rate during the optimization of the policy and value neural networks reduces the change of the policy between training epochs.

The reward function is adapted to penalize vehicles that are involved in a collision. Another important insight is to distinguish between vehicle trajectory *terminations due to own fault or due to external circumstances*. For example, a vehicle that crosses the lateral road boundary or collides with another vehicle terminates at own fault, but a vehicle that is hit in a rear-end collision is not at fault. For vehicles that are not at fault, the return is computed as if the trajectory was continued indefinitely, whereas vehicles that are at fault receive a final penalty and no further rewards after their termination.

With this, a policy is trained for controlling multiple vehicles in roundabout situations. The experiments show that it is essential for the reward function to not only reward the progress of each agent, but also the compliance with right-of-way rules. Otherwise, the training result is sometimes a policy with an inverted right-of-way logic, where vehicles that enter the roundabout have priority over vehicles that are inside the roundabout. Under this policy, traffic jams form at high simulated traffic densities.

After adapting the reward function to adhere to right-of-way rules and to maintain safety distances to other vehicles, a safe policy is learned that stays on the track, avoids rear-end collisions, and handles right-of-way situations correctly.

Modelling Heterogeneous Behavior Finally, the question of how to model heterogeneous driving behavior is addressed. To this end, a method to learn a flexible driving policy that can represent different driving styles is proposed. This is achieved by introducing *preference* parameters that change the reward function for individual agents. By making these parameters also part of the observation vector that is fed into the policy, the agents can observe their preferences and learn to adapt their behavior. This is demonstrated for the example of preferences for the minimum time gap, the minimum distance to a preceding vehicle, and the penalty for the lateral acceleration: The simulated vehicles rarely fall below the individual safety distance, and adapt their velocity in curves to experience their individual comfortable acceleration.

Comparison to Multi-Step Training There are two central differences between policy gradient RL and multi-step training from Section 4.2. First, multi-step training directly tries to find a policy that produces trajectories similar to a dataset of ground truth trajectories, whereas RL generally aims to maximize a manually defined reward function. While direct imitation is a straightforward way to learning a policy for predicting driver behavior, it is inherently limited because it can only be trained in situations for which a set of real world trajectories has been recorded. This is the central motivation of this thesis to investigate IRL approaches in the next chapter, which build upon the ideas and implementation presented in this chapter.

Secondly, multi-step training computes the gradient with respect to the policy parameters analytically, whereas policy gradient RL methods approximate the gradient stochastically. This stochastic approximation has the advantage that it does not require a differentiable simulation environment. Moreover, while the analytic gradient computation is clearly more accurate, it becomes numerically instable for large simulation horizons, approximately larger than $10\,\mathrm{s}$ in Section 4.3.2, and requires pre-training the policy with shorter horizons for stabilizing the training. In the context of deep learning, the numerical instability can manifest as vanishing or exploding gradients [Zha+22, Ch. 5.4 and 9.7]. In contrast, the stochastic

gradient approximation used in the experiments from this chapter is not prone to instabilities and can be used for arbitrary long simulation horizons.

6 Reconstructing the Rewards: Inverse Reinforcement Learning

Parts of this chapter have been published in [**Sac+22b**].

Learning a policy from rewards using RL, as described in the previous chapter, has one major drawback: A reward function is required. As the goal of this thesis is to find a policy that accurately models human behavior, the reward function needs to accurately describe the incentive structure of real-world drivers. This reward function is generally unknown. Therefore, this chapter is about methods to establish a reward function that explains the behavior of human drivers. This is desirable, as the reward function is a compact representation of the goals of the drivers and can be used to infer a driving policy even for situations where no training data is available. In the behavior triangle in Figure 1.2, this chapter closes the last link: deriving the reward function from a dataset of observed trajectories.

One approach to finding an appropriate reward function has been practically applied in Section 5.4.2: Making an initial guess of the reward function, training the policy with it, observing the resulting outcome, and then adapting the reward function by adapting the terms and varying their weights. Guessing the reward function involves many degrees of freedom, for example: Should the acceleration be penalized quadratically, or linearly? Is the weight of longitudinal acceleration equal to that of lateral acceleration? How should progress along the track be rewarded? And how should a violation of the minimum distance be penalized?

Before observing the effect of a new reward function on the resulting behavior, the policy needs to be trained again. This makes the procedure tedious, as many iterations are required until a policy that produces realistic behavior is found.

This procedure is automatized by Inverse Reinforcement Learning (IRL) [NR00]: Given expert demonstrations (trajectories) of the behavior of an agent and a simulation model of the environment, the goal is to determine the reward function that the agent optimizes. Classical IRL methods such as [NR00; AN04; Zie+08] aim to reconstruct the weights ω_i of a linear combination of known reward-features, i.e., $\mathcal{R}(y) = \omega_1 \mathcal{R}_1(y) + \omega_2 \mathcal{R}_2(y) + \dots$. One key insight used by [AN04] is that if the policy and expert trajectories are similar, the values of the reward terms must also be similar.

The learning procedure in [AN04] works as follows: A reward function is formulated, then a policy is learned using standard RL to maximize it. Next, the reward of trajectories coming from the policy is compared to the rewards of the expert demonstrations. A new set of reward

weights is constructed, such that the expert trajectories receive high rewards whereas the policy trajectories receive low rewards under this weight vector. Then, a new policy is learned and the cycle continues. When a new policy is learned, the obtained rewards are maximized, whereas they are minimized when a new set of reward weights is established. As the difference between the policy rewards and the expert rewards shrinks after each iteration, the learned policy increasingly resembles the expert policy and can match its rewards arbitrarily close under ideal conditions [AN04].

This method has three fundamental disadvantages: First, it requires the reward function to be linear and the components of the reward function to be known. Secondly, it requires a full RL training for each reward function guess, making the procedure very time-consuming [HE16; FLA16]. And thirdly, the resulting policy becomes ambiguous when the expert demonstrations are sub-optimal [Zie+08].

Therefore, the next section explores the theory behind Generative Adversarial Imitation Learning (GAIL) [HE16] and Adversarial Inverse Reinforcement Learning (AIRL) [FLL18], two recent approaches to IRL which address these issues. Both approaches realize the reward function as a neural network instead of a linear combination of reward terms, allowing for a more expressive representation. Moreover, GAIL and AIRL reduce the training time by intertwining the optimization of the policy and the reward neural network. Unlike conventional IRL methods, where these functions are optimized sequentially, this leads to a significantly shorter training time. The algorithms target a single-agent setting; Section 6.2 describes modifications to apply them to the multi-agent problem of traffic situation prediction, along with approaches to stabilize the training.

An overview on related works which use IRL in the context of trajectory prediction is given in Section 6.3. In contrast to the related works, the central novelty in this chapter is the application of AIRL to the problem of trajectory prediction. Compared to GAIL, the reward function reconstructed by AIRL can be used for RL training in additional situations that are different from the original training situations. This is exploited by confronting the AIRL policy with artificial critical situations on a new map to increase its robustness and versatility. Finally, different variants of GAIL and AIRL policies are trained and evaluated in Section 6.4.

6.1 Adversarial Learning

Finn et al. [Fin+16] remark that the back and forth between maximizing the rewards via RL and adapting the reward function to distinguish between expert trajectories and policy-generated trajectories can be interpreted as a minimax game between reward function and policy, similar to a Generative Adversarial Network (GAN) [Goo+14]. Based on this idea, Ho et al. [HE16] propose GAIL, which transfers the ideas from GAN to IRL: The policy continues to be learned by RL, but the reward function is replaced by a *discriminator*, which has the task of

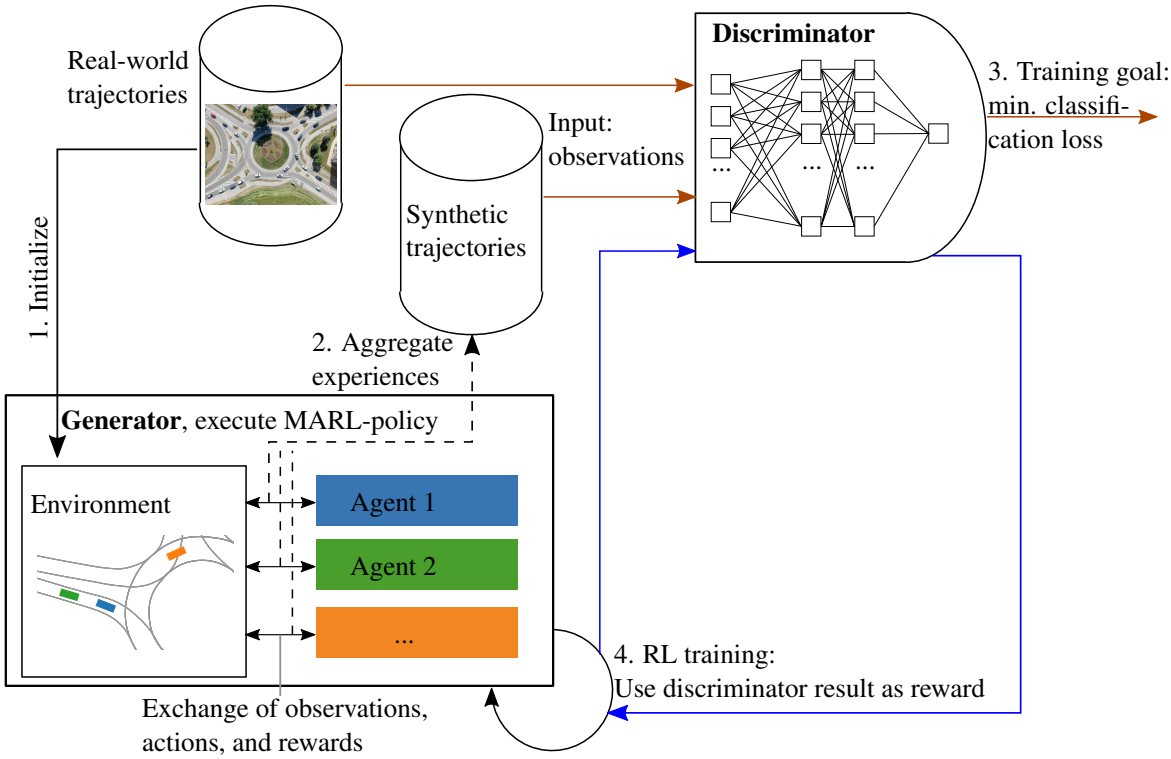


Figure 6.1: Visualization of GAIL:
1. The simulation environment is initialized from real-world traffic situations.
2. The generator, i.e., the combination of simulation environment and agent policies, is executed to build up a dataset of simulated trajectories.
3. The discriminator is trained to distinguish between real and simulated trajectories.
4. During RL training, the reward signal for improving the policy is derived from the probability of the RL trajectories originating from real data, which is estimated by the discriminator.
These steps are repeated until the trajectories generated by the learned policy closely resemble the real trajectories.

distinguishing between expert and policy trajectories. Compared to classical IRL approaches, GAIL bears the advantage that the discriminator is a neural network, which removes the need to manually engineer reward terms. Moreover, the training of the discriminator happens inside the RL training, such that the overall training time is dramatically reduced.

Fundamentally, GAIL consists of two components depicted in Figure 6.1: A generator, which is the combination of a policy and the simulation environment. It is called generator, because it generates trajectories, consisting of a sequence of observations and actions. The second component is the discriminator, which is tasked with deciding whether its input originates from the generator or from an expert demonstration. Often, the input to the discriminator is an observation, equal to the policy input.

The underlying assumption is that the expert demonstrations are a result of the execution of an unknown expert policy. To find a policy that resembles this expert policy, the discriminator

neural network is trained to distinguish between inputs from expert trajectories and inputs from synthetic trajectories by the generator. Next, one epoch of a standard RL algorithm is executed to improve the policy in the generator. The trick is that the discriminator acts as the reward function—the RL agent is rewarded, when the discriminator considers it to be an expert demonstration, and it is penalized when the discriminator detects that it is synthetic.

6.1.1 Theoretical Background: Generative Adversarial Imitation Learning

Concretely, the discriminator neural network $D_\phi : B \to [0,1]$ with parameters ϕ maps from its input $b \in B$ to the probability that the input stems from the expert. While the input features to the discriminator are often the observation vector ($B = O$), this work experiments with different inputs and therefore uses the separate symbol B for the discriminator input features. Based on a set of discriminator input features extracted from expert trajectories $\mathcal{D}_E$ and from trajectories generated by the most recent policy execution, stored in the synthetic dataset $\mathcal{D}_S$, the discriminator is trained to minimize the binary cross entropy classification loss

$$\ell(\mathcal{D}_E, \mathcal{D}_S) = -\frac{1}{|\mathcal{D}_E|} \sum_{b \in \mathcal{D}_E} \ln(D_\phi(b)) - \frac{1}{|\mathcal{D}_S|} \sum_{b \in \mathcal{D}_S} \ln(1 - D_\phi(b)). \tag{6.1}$$

Besides the use in GAIL [HE16], the cross entropy is commonly used in classification tasks [Zha+22, Chapter 22.11] and originates from information theory [CT06]. It measures how well the predicted probabilities align with the actual probabilities. The binary cross entropy becomes minimal when the predicted distribution is equal to the true distribution. Here, this is the case when every item from the expert dataset $\mathcal{D}_E$ is assigned a probability of 1, and every item from the synthetic dataset $\mathcal{D}_S$ is assigned a probability of 0.

Next, the policy is trained with the reward function [HE16]

$$\mathcal{R}_1(b) = \ln(D_\phi(b)), \tag{6.2}$$

which assigns a reward of 0 when the discriminator considers its input to stem from the expert. A negative reward up to $-\infty$ is assigned, when the discriminator estimates a low probability of the data stemming from the expert.[1]

An improved reward function

$$\mathcal{R}_2(b) = \ln(D_\phi(b)) - \ln(1 - D_\phi(b)) + c \tag{6.3}$$

is proposed by [Fin+16; FLL18] with $c = 0$ for now. Both functions are depicted in Figure 6.2. Compared to the original reward function $\mathcal{R}_1$, it additionally assigns high positive rewards

[1] For numerical stability, the discriminator outputs are clipped to $(e^{-5}, 1 - e^{-5}) \approx (0.007, 0.993)$ in this work.

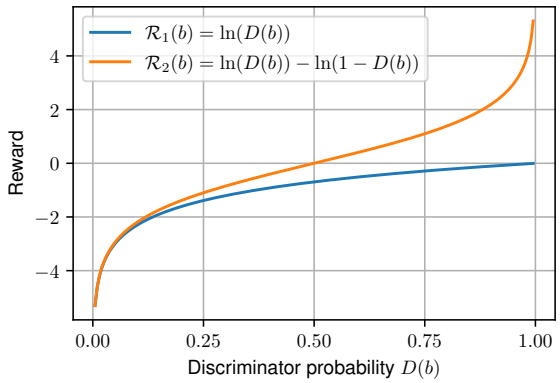


Figure 6.2: Reward functions $\mathcal{R}_1$ and $\mathcal{R}_2$: The rewards depend on the probability $D_\phi(b)$ estimated by the discriminator. If the discriminator assigns a low probability of b stemming from the expert, both rewards can assume values of up to $-\infty$; the corresponding states are therefore to be avoided by the policy. On the other hand, only $\mathcal{R}_2$ assigns high positive rewards if the policy successfully outwits the discriminator, such that the discriminator estimates a high probability of b stemming from the expert. Hence, the policy is strongly incentivized to reach these states.

when the policy manages to outwit the discriminator and receiving a high probability of stemming from the expert via $\ln(1 - D_\phi(b))$. Empirically, this reward function leads to a faster convergence of the learning algorithm.

With this, Algorithm 5 can be defined. Compared to the original GAIL algorithm presented in [HE16], two modifications are made: 1.) The improved reward function $\mathcal{R}_2$ is used. 2.) The original GAIL paper uses Trust Region Policy Optimization (TRPO) [Sch+15b] as the policy gradient algorithm in step 5, whereas the implementation in this work uses the successor PPO with GAE, as described in Section 5.1.3.

Algorithm 5 Generative Adversarial Imitation Learning, modified from [HE16]

1: Initialize neural nets for policy π_{θ_0}, value estimate $\hat{V}_\psi$, and discriminator D_ϕ randomly.
2: **for** epoch $i = 0..N$ **do**
3: Collect trajectory dataset $\mathcal{D}_S$ by executing the current policy π_{θ_i} in the environment, as in standard RL.
4: Train discriminator by making a gradient descent step in the direction of $\nabla_\phi \ell(\mathcal{D}_E, \mathcal{D}_S)$.
5: Continue standard RL algorithm with rewards assigned by $\mathcal{R}_2(b)$. Here, the combination of PPO and GAE described in Algorithm 4 on page 104, steps 4 to 6 is used: Compute GAE and return, train value estimate, improve policy using policy gradient estimate.
6: **end for**

End of Training Training GAIL can be interpreted as a two-player minimax game[2]

$$\min_{\phi} \max_{\theta} -\mathbf{E}_{\tau \sim \mathcal{D}_E} \left\{ \ln(D_\phi(b_k)) \right\} - \mathbf{E}_{\tau \sim \pi_\theta} \left\{ \ln(1 - D_\phi(b_k)) \right\}. \tag{6.4}$$

between generator and discriminator [Goo+14]: The generator tries to maximize the rewards obtained by the discriminator, while the discriminator tries to minimize the rewards assigned to the generator. The targeted stable point of this game is a Nash equilibrium, where none of both can improve any further by changing its outputs. At this point, the learned policy is equal to the expert policy. The discriminator can no longer distinguish between the expert and the learned policy and must therefore output a probability of $1/2$ for any input.

6.1.2 Adversarial Inverse Reinforcement Learning

In theory, upon reaching the Nash equilibrium in GAIL, the discriminator becomes useless, as it always outputs $1/2$ irrespective of its input. Thus, GAIL learns to imitate the expert policy, but does not recover the reward function of the expert. However, recovering the reward function is desirable, because it is a more distilled form of describing behavior: While the reward functions in Chapter 5 consist only of few interpretable terms, the policies that maximize the reward functions are significantly more complex, as they need to determine the behavior for every conceivable situation. Applying the learned policy in situations different from the training situation may not succeed due to distributional shift. This is also the reason why behavior planning in automated vehicles is typically formulated as an optimization problem of a reward or cost function as opposed to directly programming the behavior for all situations.

Modifying GAIL to also recover the reward function is the motivation behind AIRL [FLL18]. AIRL maintains the same structure of GAIL, but imposes a special structure on the discriminator. Apart from using the modified discriminator, all steps in Algorithm 5 are identical and the structure in Figure 6.1 remains unchanged.

Fu et al. [FLL18] show that the AIRL discriminator, which classifies based on observations o and actions a, can be formulated as

$$D(o, a) = \frac{p_E(a|o)}{p_E(a|o) + \pi(a|o)}. \tag{6.5}$$

with the unknown expert policy $p_E(a|o)$. The idea is that given an observation o, the decision whether one observes expert or simulated behavior can be made based on the probability that an expert would select this action $p_E(a|o)$ and the probability that the policy would select

[2]The equation depends on the reward function for the generator. Here, $\mathcal{R}(b) = -\ln(1 - D_\phi(b))$ from [Goo+14] is used. The other reward functions introduced in this section yield a similar minimax games that require two separate equations for their formulation.

this action $\pi(a|o)$. If the prior probability for both options is $1/2$, then the decision rule (6.5) follows directly from Bayes' theorem. While GAIL learns the full discriminator function D_ϕ directly, AIRL only learns an approximation of p_E to evaluate the discriminator term.

The underlying idea is that the unknown reward function of the expert $\mathcal{R}_E(o,a)$ induces a probability distribution $p_E(a|o)$. Under this distribution, states with high rewards should be more likely than states with low rewards, because the expert acts to maximize its rewards. Among all conceivable distributions that fulfill this property, the maximum entropy distribution [Fin+16; FLL18]

$$p_E(a|o) = \exp(\lambda \mathcal{R}_E(o,a))/Z(o), \tag{6.6}$$

is the distribution that expresses the maximum uncertainty with λ and $Z(o)$ chosen such that the integral of the distribution is 1. Any other distribution is biased, as it assigns higher probabilities to some states than what would be reasonable. A detailed explanation of the maximum entropy principle including a proof can be found in Appendix B.3.

AIRL approximates p_E by introducing a neural network g_ϕ that replaces p_E in (6.5) with

$$\hat{p}_E(a|o) = \exp(g_\phi(o,a)), \tag{6.7}$$

such that g_ϕ approximates $\lambda \mathcal{R}_E(o,a) - \ln(Z(o))$ and can therefore be interpreted as a scaled and shifted approximation of the expert advantage function, whose maximum indicates the best action a for a given observation o. The optimal behavior is invariant to shifting or scaling the rewards with $\lambda > 0$.

As with GAIL, the discriminator neural network $g_\phi(o,a)$ is trained by maximizing the binary cross entropy classification loss $\ell(\mathcal{D}_E, \mathcal{D}_S)$ introduced in (6.1). To train the policy, the reward function $\mathcal{R}_2$ introduced for GAIL is maximized using RL. Inserting the AIRL discriminator $D(o,a)$ of (6.5) with the approximated expert policy probability density function $\hat{p}_E$ (6.7) into

$$\begin{aligned}
\mathcal{R}_2(o,a) &= \ln(D_\phi(o,a)) - \ln(1 - D_\phi(o,a)) && (6.8)\\
&= \ln\left(\frac{\hat{p}_E(a|o)}{\hat{p}_E(a|o) + \pi(a|o)}\right) - \ln\left(\frac{\pi(a|o)}{\hat{p}_E(a|o) + \pi(a|o)}\right) && (6.9)\\
&= \ln(\hat{p}_E(a|o)) - \ln(\pi(a|o)) && (6.10)\\
&= g_\phi(o,a) - \ln(\pi(a|o)) && (6.11)
\end{aligned}$$

reveals that $g_\phi(o,a)$ can be interpreted as the reward function that the agent maximizes. The second term $-\ln(\pi(a|o))$ of (6.11) has an information theoretic interpretation [Fin+16]: It rewards the information that an action a bears, given the current policy and observation. The expected information is minimal, when the policy always selects the same action; it is maximal, when the policy selects all feasible actions with equal probability. The expected

information is known as the entropy [CT06]. Rewarding the policy for higher entropy is a common practice in RL [Zie+08; Haa+18]. Policies with high entropy are desirable, because they act as randomly as possible while striving for high rewards. This makes them robust to disturbances and ensures better exploration, compared to policies with low entropy.

6.2 Adaptions for Behavior Prediction

To apply GAIL and AIRL in the simulation environment presented Chapter 3, multiple adaptions are required. First, the adversarial training needs to be stabilized such that it reproducibly converges to good results. Approaches to this are discussed in the following. Secondly, as the algorithms are formulated for single-agent tasks, they need to be modified to handle multi-agent situations, which is outlined in the last part of this section.

Training Difficulties GAN-based architectures are known for being notoriously difficult to train [Sal+16; AB17], as the training is characterized by the competition between generator and discriminator. There is no natural balance between the two; once the discriminator learns to correctly classify most of the samples, it starts establishing a sharp decision boundary between expert and synthetic samples to minimize the discriminator loss (6.1). In this case, the discriminator assigns all inputs from the expert dataset probabilities close to 1 and all inputs generated by the policy probabilities close to 0. As a consequence the gradient of the discriminator output with respect to its input is almost everywhere close to 0. The generator loses the chance to improve, as changing its actions does not increase its score assigned by the discriminator, hence there is no signal in which direction to improve [AB17; ACB17].

To mitigate this phenomenon, multiple measures are taken to weaken the discriminator:

1. Following a suggestion from [AB17], additive Gaussian noise is applied to the discriminator inputs during the discriminator training. Due to the additional noise, the expert samples and the synthetic samples might overlap and the decision boundary of the discriminator cannot be as sharp. This effect is illustrated in Figure 6.3
2. Following a suggestion from [FLL18] and [Bha+18], the synthetic dataset $\mathcal{D}_S$ during discriminator training contains not only experiences from the most recent policy execution, but also experiences from 20 random earlier training epochs. Thus, the discriminator cannot overfit to the most recent policy execution.
3. The discriminator input b is only a subset of the full observation vector. The idea behind this is that in a lower-dimensional space, it is harder to draw a clear decision boundary between the expert and synthetic experiences. This effect is illustrated in Figure 6.4.

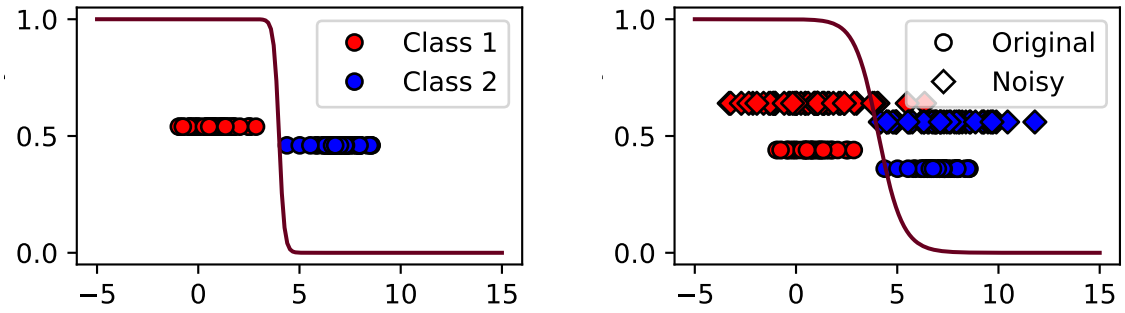


Figure 6.3: Classification with noisy 1D training data: On the left, a classifier is trained to distinguish between items from the two classes. The estimated probabilities of an item belonging to class 1 (red line) are shown. As the classes are clearly separable, a sharp boundary exists between the two. On the right, the estimated probabilities from a classifier which was trained on the same dataset with additional Gaussian noise is shown. Due to the additional noise, the two classes now overlap, and the decision boundary must be smoother. This logic extends to classification in higher dimensions.

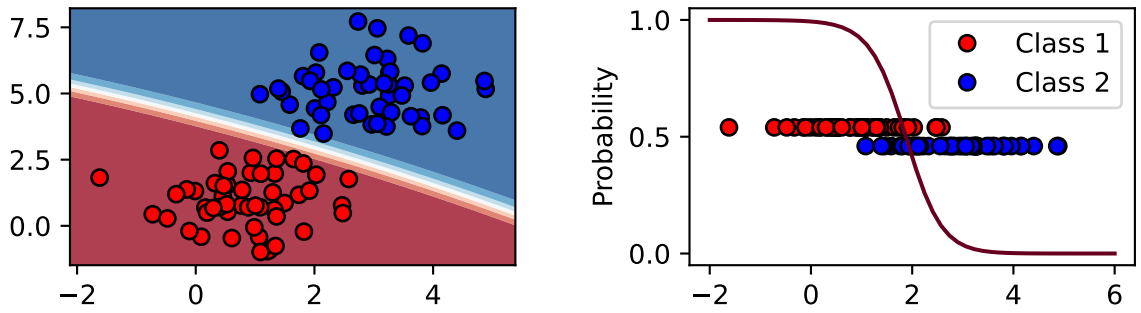


Figure 6.4: Classification in 2D and 1D: While a sharp decision boundary can be drawn in the 2D classification problem on the left, the two classes overlap when projecting the data to one dimension. As a result, the classification boundary of the 1D-problem cannot be as sharp. The probability of an item belonging to class 1 (red line) smoothly transitions from 1 to 0 in the overlapping region.

Table 6.1: Components of the policy observation vector o, the full discriminator input vector, and the restricted discriminator input vector. Equal to Table 3.1, all features are standardized before processing them with a neural network.

Feature	Policy	Full Discriminator	Restricted Discriminator
Speed	✓	✓	✓
Longitudinal acceleration	✗	✓	✓
Lateral acceleration	✗	✓	✓
Last steering angle	✗	(✓)	(✓)
Distance to left and right boundary	✓	✓	✗
Heading relative to lane in $\{0, 5, 10, 20\}\mathrm{m}$	✓	✓	✗
Road curvature in $\{0, 5, 10, 20\}\mathrm{m}$	✓	✓	✗
Speed of preceding vehicle	✓	✓	✓
Distance to preceding vehicle	✓	✓	✓
Distance to next yield line	✓	✓	✓
Speed of conflicting vehicle	✓	✓	✓
Distance of conflicting vehicle to merge zone	✓	✓	✓
Angle of conflicting vehicle to merge point	✓	✓	✗
Speed of 2nd conflicting vehicle	✓	✓	✗
Distance of 2nd conflicting vehicle to merge zone	✓	✓	✗
Distance to next priority merge zone	✓	✓	✗
Speed of non-priority vehicle	✓	✓	✗
Distance of non-priority vehicle to merge zone	✓	✓	✗

Legend: ✓: Feature included, ✗: Feature excluded, (✓): Feature optional, see experiments section

Restricted Discriminator Inputs Besides weakening the discriminator, restricting its input space has further advantages: Ho et al. [HE16] argue that the policy learns to match the occupancy of the real world data in the discriminator feature space. Through this mechanism, it can be controlled in which regard the distribution of the policy features is similar to the expert demonstrations.

This idea is used to select the features that form the restricted discriminator inputs, listed in Table 6.1: First, the policy should exhibit similar kinematics to the experts, i.e., the speed and longitudinal and lateral acceleration. As the distance between the front and rear axle to the center of gravity, l_f and l_r, are assumed to be fixed in this thesis, this renders the steering angle redundant, which can be seen when filling in (3.4) into (3.6). To test whether the steering angle can therefore be omitted as a discriminator input, it is optionally included during training.

Secondly, the relation to the preceding vehicle should be similar between policy and experts, such that the policy exhibits realistic car-following behavior. For this purpose, the distance to and the speed of the preceding vehicle are included.

The MARL experiments in Section 5.4.2 have shown that the reward function must reward the compliance with right-of-way rules. These are not explicitly specified to the discriminator, but the required information to deduce them should be contained in the input vector. Therefore, thirdly, the distance of the agent to the next yield line, the distance of the closest conflicting vehicle (with priority) to the merge zone and its speed are part of the feature vector.

These eight features form the restricted discriminator input vector. The remaining features listed in Table 6.1 continue to be observations, i.e., inputs to the policy. They are essential for controlling the vehicle, such as the future road curvatures or the distance to the boundary, but are in part determined by the environment: The policy cannot control the curvature or the width of the road. While the policy is trained in the same environment where the real world data was captured for now, this might become important when the training environment differs. For example, the discriminator could learn to distinguish between the training and the expert environment instead of the features exhibited by the behavior policies.

For this reason, and to reduce the dimensionality of the discriminator input, training is performed on the proposed restricted discriminator input vector. To assess the impact on the prediction performance, the training is also performed with a full discriminator input, which contains all features listed in Table 6.1. All features are standardized using the values listed in Table 3.1 before feeding them into the policy, value, or discriminator neural network to ensure that the training is not hampered by the different scales of the original features.

Multi-Agent Adaptions IRL algorithms are typically formulated for the single-agent case, including GAIL and AIRL. To adapt the methods to multi-agent settings, the ensuing challenges and required modifications are similar to MARL, as described in Section 5.2, and briefly recapitulated in the following.

The simulation environment is populated with multiple independent agents, as described in Chapter 3. Agents can interact with each other, as they are represented in each other's observation vector. Parameter sharing is used, such that each agent is controlled by the same policy. The parameter-sharing variant of PPO, as lined out in Section 5.2, is used for the policy training. Hereby, the GAIL or AIRL discriminator acts as the reward function. The set of synthetic trajectories, which is used for training the discriminator, is constructed from the trajectories of all agents in the traffic situation.

The original GAIL and AIRL algorithms do not consider the case of an agent being able to willingly terminate its simulation. However, this is possible in the simulation framework used in this work, as the simulation of an agent is terminated when it leaves the track or collides with another agent. This leads to a curious effect: During training, the discriminator typically has the edge over the generator and assigns a probability of less than 0.5 to the majority of the generator samples, thereby correctly identifying them as stemming from the generator. As a result, the rewards (6.3) assigned by the discriminator are on average negative, which becomes clear when evaluating the reward function $\mathcal{R}_2(b)$ for a probability assigned by the discriminator $D(b) < 0.5$. To avoid the pain of permanently receiving negative rewards, agents learn to drive off the track as fast as possible. A simple remedy is to add a constant positive offset c to the rewards, as hinted in (6.3). If c is sufficiently large, the agents are

encouraged to remain in the simulation, despite being exposed by the discriminator.[3] Besides introducing this "survival instinct" to the policy, the optimal behavior is unchanged with respect to the discriminator reward.

6.3 Related Works

Conventional IRL, Linear Rewards One early example of applying IRL to the problem of predicting human trajectories is presented by Ziebart et al. [Zie+09]. In a single-agent setting, the goal is to predict the future movement of pedestrians in an office environment. A grid map of the situation is established that contains different reward features per cell, i.e., a constant reward of that position, an indicator whether the cell is free or occupied, and indicators of the distance to the closest obstacle. The reward function is a linear combination of six terms $\mathcal{R}_i$, weighted with ω_i and evaluated at cell y, i.e., $\mathcal{R}(y) = \omega_1 \mathcal{R}_1(y) + \omega_2 \mathcal{R}_2(y) + \dots$. Using maximum entropy IRL [Zie+08], the weights of the features are determined to maximize the probability of observed trajectories under the resulting reward function. Later, predictions are made by planning under this reconstructed reward function. This idea is picked up and extended by [Kit+12], who use a semantic perception of the environment using computer vision to determine different features that influence human behavior in a parking lot, such as vehicles, pavement, grass or sidewalks.

With the goal of controlling the motion of an automated vehicle on a highway similar to how humans would control it, Kuderer et al. [KGB15] use maximum entropy IRL to determine the weights of a reward function that incorporates a total of nine terms, rewarding or penalizing the kinematic state as well as being close to surrounding vehicles. Using the estimated reward function, they later demonstrate that a motion planner optimizing for this reward function displays acceleration profiles and performs lane changes comparable to the demonstrations. However, the work only considers low-level motion planning and assumes the tactical decisions, e.g., whether to change lanes to overtake, to be given. A similar idea, extended to combined behavior and motion planning, is presented in [Ros+19].

GAIL and AIRL, Neural Network Based Rewards All previously discussed works approach the problem with maximum entropy IRL [Zie+08], which recovers the weights of a linear reward function. This imposes a limit on the class of reward functions that can be recovered, as the shape of the reward terms (e.g., quadratic penalty on accelerations, linear reward for velocity) needs to be known in advance. Moreover, it is very time-consuming, because it requires repeatedly performing a full RL training with different weights for the reward function terms. Hence, Kuefler et al. [Kue+17] explore GAIL for the purpose of

[3] In practice, this work uses $c = 3$, which ensures positive rewards when the discriminator assigns a probability $> .05$.

predicting and simulating driver behavior on highways. The environment representation for the policy and the discriminator is constructed by sending out 20 virtual beams around the simulated vehicle, and measuring the distance until the beam hits an obstacle. To adequately represent the environment to the policy, this requires the use of a RNN to fuse observations over multiple timesteps, thus moving away from the theoretical foundation of the Markov assumption, which is a fundamental simplification made in most RL algorithms. Otherwise, the policy could not react to surrounding objects when they are briefly not hit by a beam.

All previously mentioned works consider only the single-agent problem, which leads to unrealistic behavior: Applying the learned policy in a multi-agent environment, [Kue+17] reports collision rates above 10% for their model, as only one vehicle is controlled by the policy and the surrounding vehicles are played back from the dataset. Hence, the surrounding vehicles are incapable of reacting to the behavior of the simulated vehicle. Bhattacharyya et al. [Bha+18] aims to address this problem by extending the work from [Kue+17]: Using a training curriculum to stabilize the training progress, the policy successively controls larger portions of the vehicles in the simulated traffic scene, while the remaining vehicles are played back from the data. Finally, all vehicles are controlled by the policy. To realize this approach, a parameter sharing variant of TRPO proposed in [GEK17] is combined with GAIL. While this improves the prediction RMSE compared to single-agent training, the collision rate is still larger than 10%. To rectify this deficiency, a followup work [Bha+19] augments the GAIL reward by a term that penalizes off-track driving, collisions and hard braking. This pushes the reported collision rate down to around 3%. An overview of this line of research by a Stanford group is given in [Bha+23].

In the context of automated driving, AIRL has been applied by [Wan+21]. Thereby, the goal is to learn a high-level policy for an automated vehicle that selects one of three gaps for merging into a neighboring lane in a highway scenario. However, the goal of the work is to control an automated vehicle, not behavior modelling. Only the high-level lane-change decision is made by the policy, while the low-level trajectory is determined two controllers for lateral and longitudinal movement. This approach is unsuitable for behavior modelling, because the nuances of driving behavior cannot be captured, e.g., how the vehicle accelerates before changing lanes, or how fast the lane change is performed. Moreover, the policy is learned only for one vehicle, whereas the surrounding vehicles are controlled by a traffic simulation.

Novelties in This Work This work, along with the associated publication [**Sac+22b**], is the first to use AIRL for low-level trajectory control in a multi-agent setting. With this, the most relevant related works by Bhattacharyya et al. [Bha+18; Bha+19; Bha+23] are extended in the following ways: First, the AIRL discriminator can be interpreted as a reward function, whereas the GAIL discriminator has no comprehensible meaning. In the following, this is used to generate an insight on the reward function that AIRL maximizes. This helps to

interpret the learned behavior, because it explains the goals that the policy tries to achieve. Moreover, this work proposes to use the learned AIRL reward function to train in additional fictional situations on a new map. The situations are initialized to contain many critical situations, e.g., near collisions. It is shown that this helps learning a policy that has a lower collision rate and generalizes better to untrained situations.

Secondly, the observation vector with semantically meaningful features, known from previous chapters, is used for training the policy and the discriminator. Besides using AIRL, this is another prerequisite for being able to interpret the discriminator as a reward function, because each discriminator input has a semantic meaning. Moreover, this allows for the targeted limitation of the information that the discriminator uses for its decision. This is used for deciding in which regards the learned policy should resemble the real-world driver behavior, and which features can be ignored. Both would not be possible with the abstract beam-based environment representation used in the works by Bhattacharyya et al.

Thirdly, in contrast to most surveyed works which operate on highway scenarios, the experiments in this section are performed in highly interactive urban roundabout situations. Together with the previous chapters, this allows for a systematic comparison of different approaches to behavior modeling under equal conditions. The ideas from this chapter have also been transferred to highway scenarios in [**Rad+23**].

6.4 Experiments

Training both GAIL and AIRL involves a standard RL training. For this, the same simulation environment is used as for RL and multi-step BC training, described in Chapter 3. In contrast to multi-step training, the simulation environment is not required to be differentiable, as the gradient of the discriminator rewards with respect to the policy parameters is approximated stochastically.

The experiments are designed to shed light on the following research questions:

- Are the proposed multi-agent variants of GAIL and AIRL suitable for learning a driving policy that accurately models human behavior?
- Can the failure rate be reduced compared to BC approaches, by explicitly penalizing collisions and driving off track?
- Does a restricted set of inputs which are deemed relevant for the discriminator facilitate the learning task and lead to better policies?
- Can the AIRL discriminator be interpreted as a reward function, and what is the structure of the reconstructed reward?
- Further, can the AIRL discriminator be used for training in additional fictional situations, and does this improve the learned policies?

Equal to BC, a dataset of trajectories is required for training the discriminator and thereby indirectly the policy. To this end, the same training, validation, and test datasets as for BC are used, as described in Section 3.3. Besides using the training dataset for training the discriminator, the simulation for learning the policy is initialized with situations from the dataset, such that the initial simulation state is always a realistic state from the real world.

The problem of finding the optimal policy is now non-stationary in three ways: First, the expected returns of the experience collected during one epoch change while the policy changes, which is addressed by the PPO algorithm, as lined out in Section 5.1.3. Secondly, due to the multi-agent setting, the environment dynamics change when the policy changes, as explained in Section 5.2. For both GAIL and AIRL, additionally, a third non-stationarity is introduced by the discriminator, which amounts to the goal of maximizing a reward function that is changing during the course of training. Despite this challenging problem setup, the Adam optimizer [KB15] that is also employed for training all other models in this work empirically delivers good and reproducible results.

The prediction performance of the policy is evaluated after each epoch. Again, two key performance criteria are used: First, the failure rate (collisions and off track driving), which should be close to 0 and measures the plausibility of the model. Secondly, the along-track prediction RMSE of $8\,\mathrm{s}$-predictions is evaluated, which measures the precision of the predictions. Both are evaluated on the training set, which is used to train the discriminator, and on the validation set, which is not used for training.

With two criteria to be minimized, there is no longer a clearly defined best result. Here, the best policy is defined as the policy that exhibits the lowest failure rate on the validation set. Often, there are multiple policies with equal failure rate, as the validation set contains only approximately 300 vehicles, and many policies achieve close to zero failures. Therefore, if a policy achieves the same minimum number of failures multiple times during the training, the epoch with the lowest RMSE is considered the final result of that training run.

For a fair comparison with the performance of the BC algorithms in Section 4.3, the policy operates on the same normalized observation vector, listed in Table 3.1, and has the same network architecture, depicted in Figure C.1. The additional discriminator and value neural network have the same architecture, but output only one single value.

Unlike in BC, no pre-training of the policies is performed or required. Training always starts from scratch with initially random policy, value and discriminator neural networks. Due to this, as well as the random sampling of actions during the policy rollouts in each RL epoch, the training procedure is stochastic. To ensure significance of the results, each training run is repeated seven times.

6.4.1 Generative Adversarial Imitation Learning

To better understand the GAIL algorithm, this section first illustrates the training progress for one single successful training run. In the second part of this section, the effect of different design decisions is investigated.

The training is performed for 200 epochs, but good results often emerge much earlier, after approximately 50 epochs. One training run with 200 epochs takes approximately $45\,\mathrm{min}$ on a single core of an Intel i7-9700 @ $3\,\mathrm{GHz}$. Interestingly, the training duration per epoch is approximately equal to RL,[4] as the overhead due to the discriminator training is small compared to the remaining steps that are also performed in the original RL algorithm. This is a large benefit over classical IRL approaches such as [Zie+08], which repeatedly execute full RL training runs until a suitable reward function is reconstructed.

Both the discriminator and the policy make multiple steps along the gradient per training epoch, as they are trained on minibatches of size 2048 that are drawn from the full set of experiences. Since the full set contains about $50\,000$ experiences per epoch,[5] the discriminator neural network is updated approximately 25 times per epoch for a pass over the full batch of experiences. The policy is trained with 20 passes over the full experience batch, leading to a total of approximately 500 parameter updates per training epoch. Improving the policy by making multiple passes over the experience batch reduces the amount of required simulations, which is a key benefit of the underlying PPO algorithm, as discussed in Section 5.1.3. All training parameters are listed in the appendix in Table C.5.

Training Progress Equal to RL, the policy is initialized with random parameters and an initial policy standard deviation[6] $\ln(\sigma) = 0, \sigma = 1$, such that the policy initially selects accelerations and steering angles approximately widely scattered among the available options, c.f. Appendix B.2. As the feasible longitudinal accelerations are not centered around 0, i.e., $a_{\text{lon}} \in (-7, 3)\mathrm{m/s^2}$, this leads to an initial tendency to brake, explaining the initial large negative mean displacement in the top left of Figure 6.5. Executing this random policy leads to high initial failure rates, because many vehicles collide or leave the track, shown in the bottom right of Figure 6.5. During training, the policy is executed stochastically, i.e., its actions a are drawn from

$$a \sim \pi(\mathbf{a}|\mathbf{o} = o) = \tanh(\mathcal{N}(\mu_{\mathbf{a}}, \Sigma_{\mathbf{a}})), \tag{6.12}$$

[4] Here, training is performed with approximately 1000 trajectories. In the MARL experiments, training with 500 trajectories for 1000 epochs took 3 to $4\,\mathrm{h}$. In both cases, one training epoch takes between 11 and $14\,\mathrm{s}$.

[5] 1007 vehicles simulated for 50 steps, but some simulations terminate early due to vehicles colliding, leaving the track laterally or leaving the map. This amounts to approximately 2.8 hours of simulated driving per training epoch.

[6] See Section 5.3.2 for a recap of the policy standard deviation and the description of how the policy neural network parameterizes the action distribution.

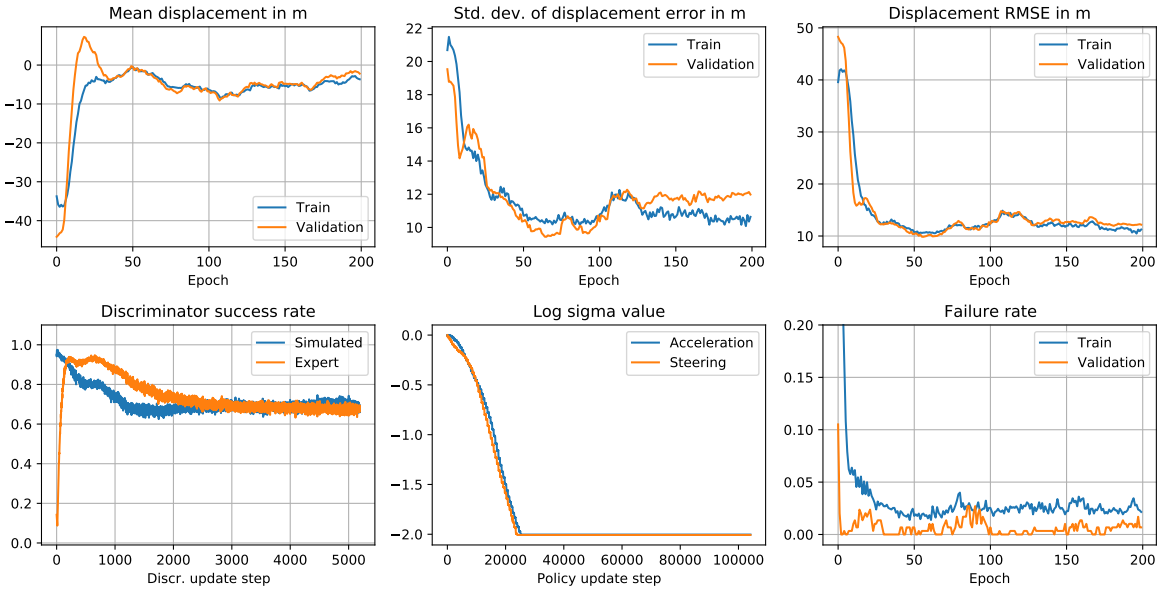


Figure 6.5: Performance measures during one single GAIL training run. Prediction errors and failure rates are measured for $8\,\mathrm{s}$-predictions. Plots from top left to bottom right: The mean displacement is the along-track distance between the predicted position and the true position after $8\,\mathrm{s}$, averaged over the set of training and validation trajectories. Negative values mean that the prediction is behind the ground truth. For this displacement, also the standard deviation and the RMSE are shown. Bottom: The discriminator success rate shows how frequently the discriminator assigns a probability of more than 50% to the correct class, which is displayed separately for the set of policy (simulated) and ground truth (expert) trajectories. The log sigma value indicates the exploration noise of the policy during training. As discussed on p. 112, the exploration noise is limited to a minimum value of $\ln(\sigma) = -2$ to ensure a stable training. Finally, the failure rate measures the percentage of vehicles that collide or leave their route. It is lower in the validation situations, because the actions are selected deterministically, whereas they are drawn from the policy distribution in the training situations.

as this is required to estimate the policy gradient. Later, when the policy is used for making predictions, it is executed deterministically with $a = \tanh(\mu_{\mathbf{a}})$. As this is the use case, the policy is also executed deterministically on the validation dataset. This explains why the policy achieves lower failure rates on the validation data than on the training data.

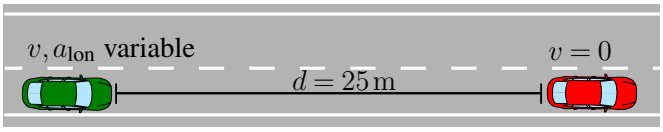
At the beginning of the training, the randomly initialized discriminator emits meaningless probabilities, and thus provides no useful training signal to the policy. After the random initialization, it labels the majority of all inputs as stemming from the policy. As a result, it initially has a high success rate in detecting inputs from the policy and a low success rate in correctly labeling inputs from the expert, shown in the bottom left of Figure 6.5. A classification is deemed successful when the discriminator assigns more than 50% probability to the correct class.

The offset $c = 3$ dominates the rewards (6.3) assigned by the discriminator to the policy. Thus, the implicit goal of all agents during early training is to not terminate early by avoiding collisions or driving off the track. The policy rapidly learns a failure-proof method to ensure that: braking to standstill and waiting for the end of the simulation. As shown in the bottom right of Figure 6.5, the failure rate on the validation data rapidly drops to 0 during the first few training epochs. Simultaneously, the RMSE prediction error is very large, as this is clearly a strongly biased model for driver behavior, even worse than the initial policy that acts completely randomly.

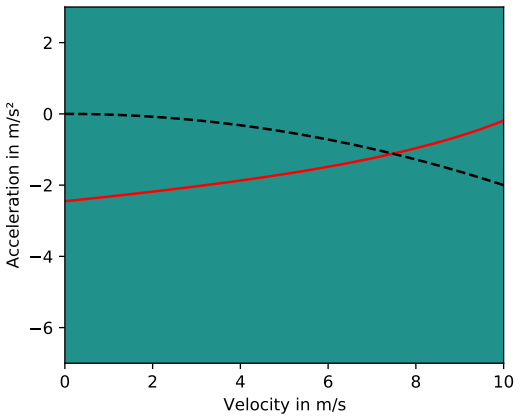
However, the discriminator quickly learns to recognize this behavior. After few training steps, it correctly identifies more than 90% of all expert and simulated trajectories. In turn, this improves the training signal for the policy, providing it with information of how expert-like its observations look to the discriminator. Thereby, the discriminator guides the policy towards better imitating the expert. Consequently, the displacement error decreases rapidly. As the policy becomes better and better at imitating the expert behavior, the discriminator becomes worse at distinguishing between the policy and expert observations, such that its success rate decreases. After half of the training time has elapsed, an equilibrium between generator and discriminator is reached, where neither of both further improves in fooling the other. At this point, the discriminator correctly identifies approximately 70% of its inputs, i.e., assigns a probability of more than 50% to the correct class. The best possible result for the policy would be that the discriminator correctly classifies its inputs in 50% of all cases, i.e., that it can only guess. The reason why the discriminator has a higher success rate is that the policy does not perfectly imitate human driving, especially as it can only exhibit one behavior style, whereas real driving styles captured in the dataset are diverse and behave differently in the same situation. In terms of displacement RMSE and failure rate, the policy reaches its optimum earlier, after approximately 50 training epochs, or $10\,\mathrm{min}$ of training time. In this work, the epoch that achieves the lowest RMSE among all epochs with minimum collision rate is selected as the training result.

To get a deeper insight into the functioning of the discriminator, its outputs after different training epochs for a simple traffic situation are depicted in Figure 6.6. To generate the plots, the discriminator is trained on the roundabout dataset. After different training epochs, it is queried to classify the situation depicted in (a), where a vehicle is approaching another standing vehicle with varying initial speed v and longitudinal acceleration a_{lon}. Based on the information at this moment, the discriminator needs to decide whether it is looking at a situation experienced by a real driver, or at a situation that the policy has caused.

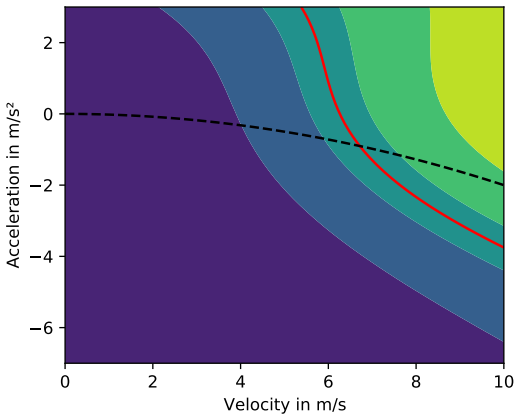
Initially (b), the yet untrained discriminator outputs a probability close to 50%, regardless of its input. After 12 epochs (c), it has learned to differentiate mostly upon the speed in this situation: it assumes that any vehicle driving slower than $5\,\mathrm{m/s}$ most likely stems from the simulated data. As shown in Figure 6.5, the policy has a tendency to brake during early



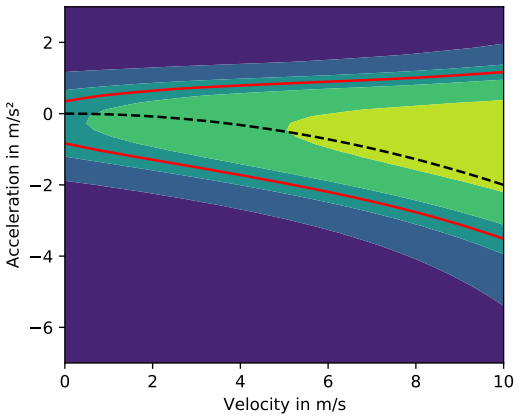
(a) Situation: The agent approaches a standing vehicle in $25\,\text{m}$ on an otherwise empty, straight road with variable speed and acceleration



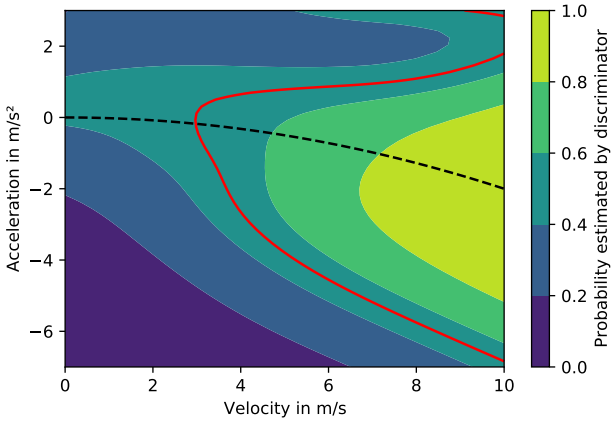
(b) Initial



(c) After 12 epochs



(d) After , because they 50 epochs



(e) After 124 epochs

Figure 6.6: GAIL discriminator output for the situation depicted in (a). In the plots, the red line indicates the decision boundary, i.e., where the estimated probability of the sample stemming from real or simulated data is 50%. The overlaid dashed line shows the minimal constant acceleration required to avoid a collision. If the goal is to avoid a collision while applying minimal acceleration, this is the optimal behavior. See the accompanying text for an interpretation.

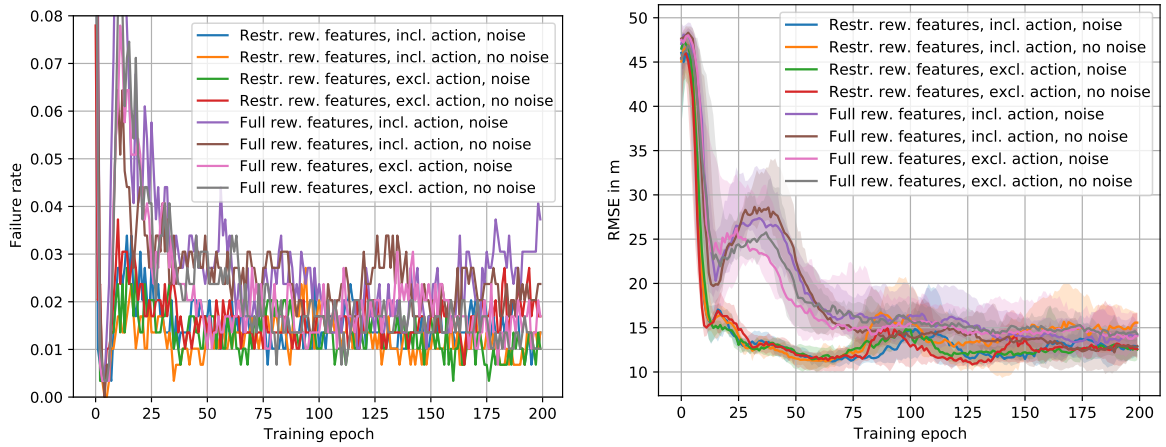
training, rendering low velocities and strong braking a reliable decision criterion to identify policy behavior. However, as the training continues, the policy starts driving faster, and the discriminator must find a new decision criterion. Without explicitly knowing the optimal braking behavior, after 50 epochs (d), the discriminator estimates a high probability of the data stemming from an expert, when the behavior is close to the optimal braking line. If the agent accelerates or brakes much stronger than optimal, the discriminator assigns a low probability. After 124 epochs (e), the policy has become more refined and acts similar to a real world-driver. This makes the classification task for the discriminator harder: It can no longer output a high probability when the behavior looks plausible, and output a low probability when the behavior looks unrealistic. Now, a large area of plausible behaviors, i.e., slight braking for low velocities, or stronger braking for higher velocities, can be expected to occur under both, the policy and in the real data. Hence, the discriminator assigns a probability between 40% and 60% to plausible behaviors of velocities between $0\,\mathrm{m/s}$ and $5\,\mathrm{m/s}$ and an adequate deceleration, indicating that it cannot make a clear decision in these cases.

Design Decisions This work compares eight different training configurations for GAIL that result from the Cartesian product of three binary decisions: 1.) Using a discriminator with full or restricted features, c.f. Table 6.1. 2.) Training the discriminator with clean inputs or with additional Gaussian noise. 3.) Using a discriminator with or without access to the steering action.

The first two points are targeted to investigate whether the measures to smoothen the decision boundary of the discriminator introduced in Section 6.2 are effective. The third point investigates whether the last steering action is required as an input to the discriminator. This work hypothesizes that this is a redundant feature, because the policy likely steers to control the lateral acceleration, which is also included in the discriminator inputs.

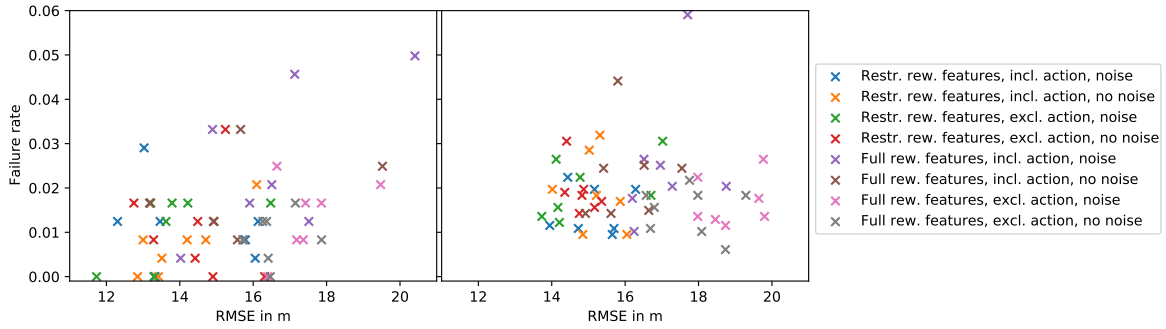
Each training configuration is executed seven times to ensure significance of the results. The resulting median failure rate and the minimum, median and maximum RMSE of all seven runs for every configuration are displayed in Figure 6.7. Equal to the single training run shown in Figure 6.5, all policies first learn to brake to standstill after few epochs, thereby pushing their failure rate to 0 but increasing their RMSE. Then, the failure rate of all models increases again as the policies learn to better imitate the expert. At the end of the training, most configurations achieve similar results with a median failure rate between 1% and 3% and a median RMSE between $12\,\mathrm{m}$ and $16\,\mathrm{m}$ in the unseen test situations. The failures can be attributed almost exclusively ($> 95\%$) to collisions and rarely to vehicles leaving the track.

All training configurations whose discriminator has access to the full observation vector require significantly more epochs until the minimum RMSE is reached, compared to those with restricted reward features. This might be explained by the larger dimensionality of the input space of the discriminator. As a consequence, the decision problem becomes more



(a) Median failure rate **(b)** Median, min. and max. RMSE

Figure 6.7: GAIL: median failure rate and RMSE of 8 s-predictions on the validation dataset of different configurations during seven training runs.



(a) Validation performance **(b)** Test performance

Figure 6.8: GAIL: Final model performance of each of the seven repeated training runs and the eight configurations. The evaluation is performed for 10 s predictions on the validation dataset (a) and the unseen test dataset (b).

difficult and requires more training data until the discriminator can confidently distinguish between expert and synthetic inputs. Even at the end of the training, the configurations with access to the full observation vector do not perform better than the configurations with the restricted discriminator feature vector. This leads to the conclusion that no relevant information is contained in the additional inputs and that the training can safely be performed with the restricted discriminator inputs.

Another perspective on the training success is given in Figure 6.8, where the performance of the best model of each of the seven repetitions and each of the eight configurations is displayed for 10 s-predictions on the validation and test set. The best run is the epoch with the lowest RMSE among all epochs with minimal collision rate on the validation data. Among the total of 56 runs, 9 achieve 0 collisions on the validation dataset. Most models achieve a collision rate below 2% and a RMSE below 17 m on the validation data. Among the 15 runs

with the lowest RMSE below $14\,\mathrm{m}$, only one uses the full discriminator features. This is a clear indicator that using the restricted discriminator inputs is beneficial. This observation is confirmed on the test set: While the RMSE and the failure rate is generally higher than on the validation set, most runs with the lowest RMSE and failure rate rely on the restricted discriminator features.

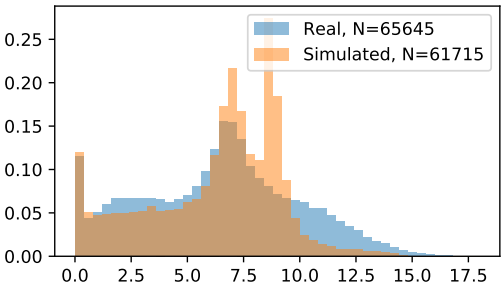
Concerning the decision of including or excluding the steering action in the discriminator inputs, no clear difference can be made out in Figure 6.7. The evaluation on the test data in Figure 6.8b with restricted discriminator features also shows no clear preference, as the results from both approaches are close to each other. However, when using the full discriminator feature vector, the configurations that include the action exhibit approximately $3\,\mathrm{m}$ lower RMSE values.

Finally, the effect of additional noise during discriminator training is investigated. While the best models on the test data in Figure 6.8b use discriminator noise, the corresponding noise-free models exhibit only slightly higher RMSE and failure rates. Hence, this improvement proves to be unnecessary. Adding noise to the discriminator inputs is for example proposed in [AB17] to fix vanishing gradient issues in GAN, typically used for image generation. As the dimensionality of feature-based policy learning in this thesis is significantly lower, this might explain why no discriminator noise is required for successful training.

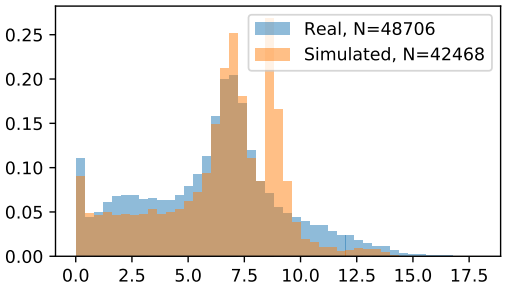
With this, it can be concluded that the most effective way of improving the model performance is to use a discriminator that operates on the proposed restricted input vector, instead of the full observation vector used by the policy. For the restricted input vector, the model performance is barely affected by the additional steering input, such that it can also be left out. The additional noise during discriminator training also improves the final model performance only slightly.

Feature Matching As previously described, the discriminator bases its decisions on a restricted feature vector b, listed in Table 6.1. To illustrate the impact on the learned policy, the features are evaluated for every timestep and every vehicle in all $10\,\mathrm{s}$-situations in the unseen test set. The same situations are predicted using the learned policy, starting from the initial situation state. Again, the discriminator features are evaluated for every timestep and vehicle. The histogram of each feature in the real-world dataset and in the predicted dataset is depicted in Figure 6.9.[7] The histograms are largely overlapping, which shows that the policy exhibits behavior that resembles the real-world drivers. However, some features by the policy are more concentrated than the corresponding real-world features. For example, the

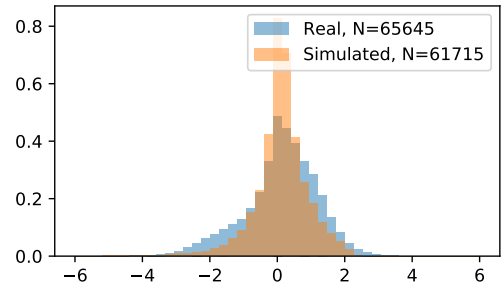
[7]The number of elements used for computing the histograms varies, because not every feature is available at every timestep (e.g., the distance to the preceding vehicle) and because the trajectories are evaluated in a limited area, which the real-world vehicles leave after a different number of steps than the simulated vehicles. Therefore, the histograms display the estimated density instead of the absolute number of occurrences.



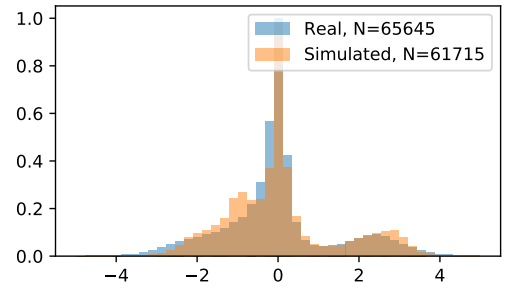
(a) Speed in m/s



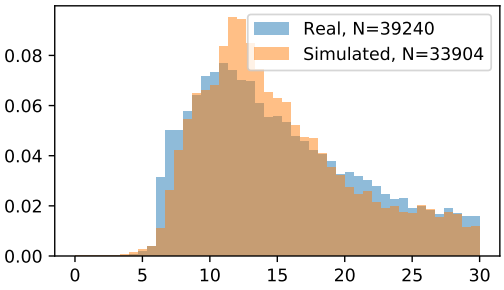
(b) Speed of preceding vehicle in m/s



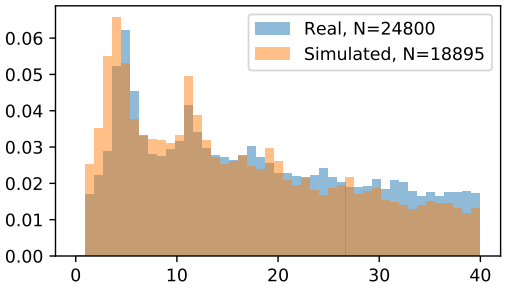
(c) Longitudinal acceleration in m/s^2



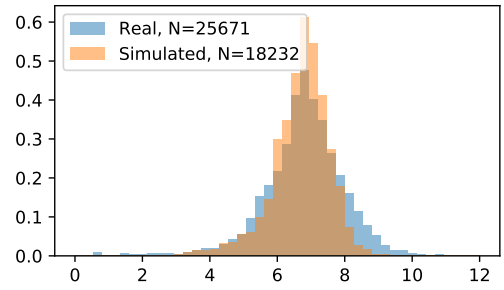
(d) Lateral acceleration in m/s^2



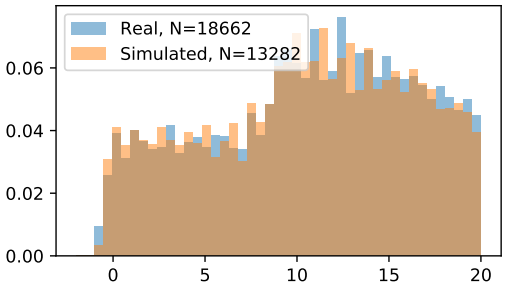
(e) Distance to preceding vehicle in m



(f) Distance to yield point in m



(g) Speed of the closest conflicting vehicle in m/s



(h) Dist. of closest conflicting vehicle to yield in m

Figure 6.9: Feature matching in GAIL: The normalized histograms show the distribution of the restricted feature vector (Table 6.1) that the discriminator uses to decide whether it is confronted with data from the real world recording or from the simulation. The histograms for the simulated data are constructed by executing the simulation starting from the initial situations of the real world test data set. It becomes clear that the policy has learned to exhibit features that are similarly distributed to the real world data.

real-world speed distribution has a peak around $7\,\mathrm{m/s}$, the typical speed of vehicles inside the roundabout. For higher velocities, the density fades out slowly, as different drivers tend to leave the roundabout at different velocities. The speed feature from the policy exhibits a similar peak at around $7\,\mathrm{m/s}$. However, it also has an additional peak around $9\,\mathrm{m/s}$, which is due to the agents accelerating to this velocity when leaving the roundabout. As there is only one homogenous policy that is executed by all agents, there is no variety in this behavior, unlike in the real world driving behavior.

Conclusion GAIL manages to learn a policy that successfully imitates the behavior from real-world recordings for all surveyed training configurations. The training is shaped by the competition between the policy and the discriminator, illustrated in the bottom left plot of Figure 6.5. After few training epochs, the discriminator learns to identify features of real-world driving, and thereby encourages the policy to also exhibit these features. At the end of training, the features of real-world drivers and the learned policy largely overlap, as illustrated in Figure 6.9.

All training runs converge within 50 to 150 training epochs to $8\,\mathrm{s}$-RMSE values between 12 and $20\,\mathrm{m}$ and failure rates below 5%, shown in Figure 6.7. The best models achieve failure rates of approximately 1% on unseen test data and a $10\,\mathrm{s}$-RMSE of about $14\,\mathrm{m}$.

The experiments in this section show that applying prior knowledge, i.e., training the discriminator with a manually selected feature vector instead of the full observation vector, accelerates the training progress without compromising the prediction performance. Further, the experiments show that the discriminator input vector does not need to include the steering action, as this is a redundant feature that can also be deduced from the lateral acceleration. Thirdly, using additional noise on the discriminator inputs during training does not significantly improve the final policy performance and hence is not required.

6.4.2 Adversarial Inverse Reinforcement Learning

Next, policies trained with AIRL are examined. As described in Section 6.1.2, the algorithm is based on GAIL, but imposes a special structure on the discriminator that allows it to be interpreted as a reward function. Apart from the changed discriminator, everything else remains unchanged and the same training parameters as described in Section 6.4.1 and listed in Table C.5 are used.

The training progress is similar to GAIL. For one single run with equal configuration to the GAIL-training from Figure 6.5, it is depicted in Figure 6.10. At the beginning of the training, the policy learns to brake to standstill to avoid collisions and the failure rate rapidly drops to 0. The discriminator quickly learns to distinguish between policy and real world demonstrations and correctly classifies approximately 90% of all inputs. This signal, in turn, teaches the RL

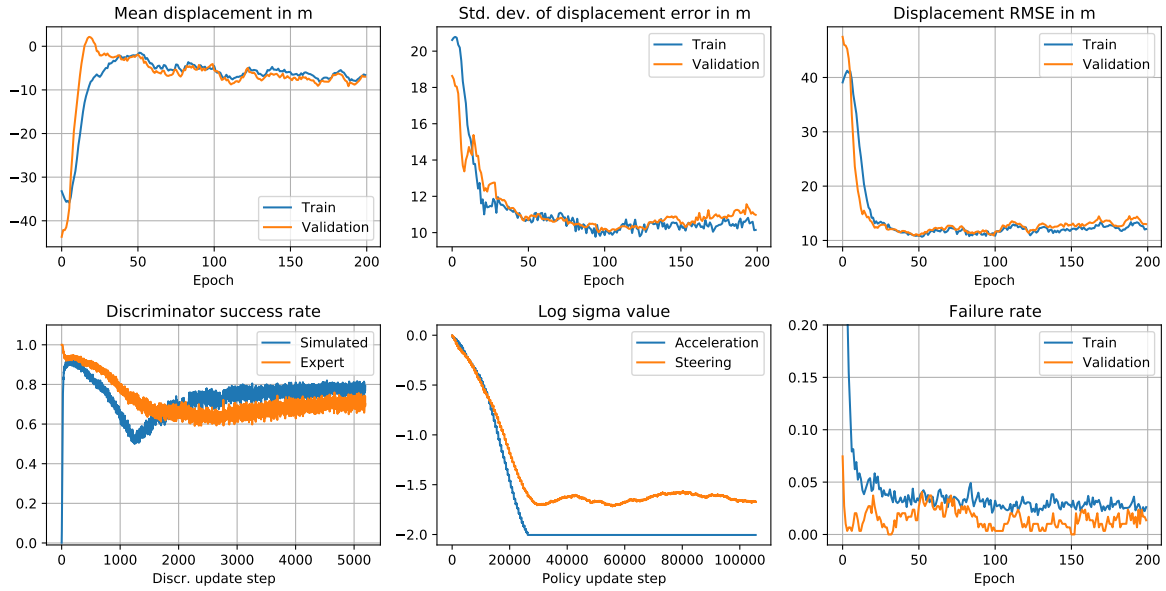
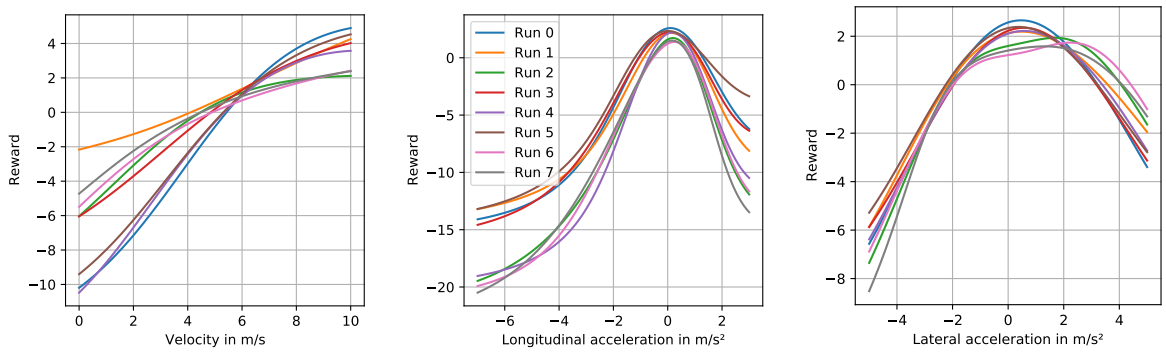


Figure 6.10: Performance measures during one single AIRL training run. Prediction errors and failure rates are measured for $8\,\mathrm{s}$-predictions. Further descriptions in text.

policy to act more realistically and leads to a decreasing RMSE of the displacement between predicted and real trajectories. In this training run, the minimum of RMSE and failure rate is reached after approximately 40 epochs. During further training, both measures fluctuate but do not continue to improve. The minimum displacement RMSE and failure rate is similar to GAIL.

Unlike in GAIL, the exploration noise of the standardized steering angle action does not reach its lower bound $\sigma = e^{-2}$, as shown in the middle bottom plot of Figure 6.10. This means that the distribution from which the steering actions are drawn from is more dispersed than the action distribution of the GAIL policy, c.f. Figure 6.5. The reason for this phenomenon is that the AIRL policy is not only rewarded for resembling the ground truth data as closely as possible via the reward approximator $g_\phi(o, a)$ inside the discriminator in (6.11), but also simultaneously for doing this with maximum entropy with $-\ln(\pi(a|o))$, i.e., for selecting its actions as randomly as possible during training.

Interpreting the Learned Reward Function As previously stated, the AIRL discriminator can be interpreted as a RL reward function. To investigate the learned reward function, eight repeated training runs with restricted discriminator inputs are executed, c.f. Table 6.1. As the previous experiments have shown that the steering angle is a redundant feature, it is not part of the restricted feature vector. For the experiment, the reward-approximating neural network $g_\phi(b, a)$ is evaluated for different variations of its inputs. To this end, an uncritical traffic situation, in which there is no preceding nor conflicting vehicle present, is simulated. Hence, the kinematic discriminator features are set to the following values:



(a) Speed **(b)** Longitudinal acceleration **(c)** Lateral acceleration

Figure 6.11: Rewards reconstructed with AIRL discriminators of eight different training runs. The plots are constructed by varying one feature at the input of the discriminator neural network g_ϕ while keeping the remaining features fixed to the values described in the text.

$v = 7\,\mathrm{m/s}$, $a_\mathrm{lon} = a_\mathrm{lat} = 0\,\mathrm{m/s^2}$. The features that describe the interaction with the non-existing surrounding vehicles are set to their default values listed in Table 3.1. With all other features fixed to these values, one kinematic feature at a time is varied: First, the speed v, then the longitudinal acceleration a_lon, and finally the lateral acceleration a_lat is varied in Figure 6.11.

The training result is reproducible and produces similar results for all eight repetitions. The reconstructed rewards resemble the manually defined reward function in Section 5.3.1: Progress along the track is encouraged by a monotonically increasing reward for the speed. The reward for the longitudinal and lateral acceleration is maximum when their value is $0\,\mathrm{m/s^2}$. It decreases smoothly and symmetrically around the maximum, similar to a quadratic function. This result carries more information: Despite having the possibility of learning complicated functions with the discriminator neural network, the reward of the longitudinal acceleration is symmetrical, i.e., accelerating is penalized equally hard as braking. The penalty for braking very hard eventually levels off after $a_\mathrm{lon} = -3\,\mathrm{m/s^2}$, which indicates that the model has learned that braking strongly is undesirable, but tolerable if necessary. Also, unlike the manually defined reward function, longitudinal and lateral accelerations are weighted differently: The penalty for longitudinal accelerations increases approximately 2-3 times as fast as the penalty for lateral accelerations.

However, there are limitations: In Figure 6.11c, four training runs have learned a slightly unsymmetrical reward function for the lateral acceleration. The model expresses a preference for lateral accelerations around $2\,\mathrm{m/s^2}$. Considering the lateral accelerations experienced by real drivers in Figure 6.9d, the reason becomes clear: This is the typical acceleration that vehicles experience when driving through the roundabout. As there are many vehicles driving with this acceleration, some models seem to have learned that it is desirable to drive with that specific lateral acceleration. The more intuitive causal explanation—that this acceleration

emerges as a tradeoff between the desire to make progress and the aversion to experience accelerations—has not been learned in these training runs.

Transfer to New Situations Obtaining real-world trajectory data is an expensive and time-consuming process. Further, some important situations are so rare that it is practically impossible to design a measurement campaign for them. One example are collisions and close collisions. This section addresses the issue by using the reconstructed reward function from AIRL to train in additional fictional situations, besides the hitherto used situations initialized from the real-world drone recordings. To this end, the AIRL algorithm is slightly altered: The discriminator is left unchanged, learning to classify between real-world observations and observations obtained by executing the policy for all agents from an initial real-world situation on. The policy, however, is trained to maximize the rewards assigned by the discriminator reward function in both, the situations initialized from the real world and the additional fictional situations.

To demonstrate the idea, the fictional situations are set up on a map of the right roundabout from Figure 3.4, for which no real-world training data is used. Further, to experience roads with different curvatures, some vehicles drive on a meandering course, a slightly curved track, and a clockwise circular track. Finally, some vehicles are set up to interact with each other in standard unsignalized right-of-way crossings. The map of the fictional situation is depicted in Figure 6.12. Similar to Section 5.4, the traffic density and the initial speed, position and heading of the vehicles is randomly initialized for each simulation. The vehicles are placed with high traffic densities in a way to maximize interaction between them and to stimulate critical situations. For example, some vehicles are placed up to $10\,\mathrm{m}$ behind other vehicles with a delta velocity of up to $14\,\mathrm{m/s}$, making a collision inevitable with the maximum possible deceleration of $-7\,\mathrm{m/s^2}$. Most situations are less dramatic, but require strong braking to avoid a collision.

Altogether, the training is now executed on the real-world training dataset encompassing 1007 vehicles, also used for training all other models, and additionally in 50 fictional $10\,\mathrm{s}$-situations with an average of 25 vehicles per situation. The number of vehicles varies randomly between the different situations to simulate high and low traffic densities. The total training duration for 200 epochs with the additional fictional situations is approximately $1\,\mathrm{h}$, compared to $45\,\mathrm{min}$ without the additional situations.

To assess the improvement, seven training runs are performed with additional training in the fictional situations and compared to training runs without the additional training. Equal to the GAIL experiments, the experiments are performed once with access to the full discriminator feature vector, and once with the restricted features. The results are plotted in Figure 6.13. Overall, the plots resemble the training progress of GAIL in Figure 6.7. In all cases, the failure rate rapidly drops to low values. The models obtain low RMSE values faster when

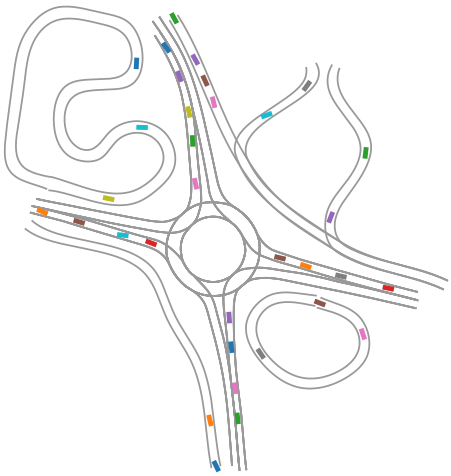
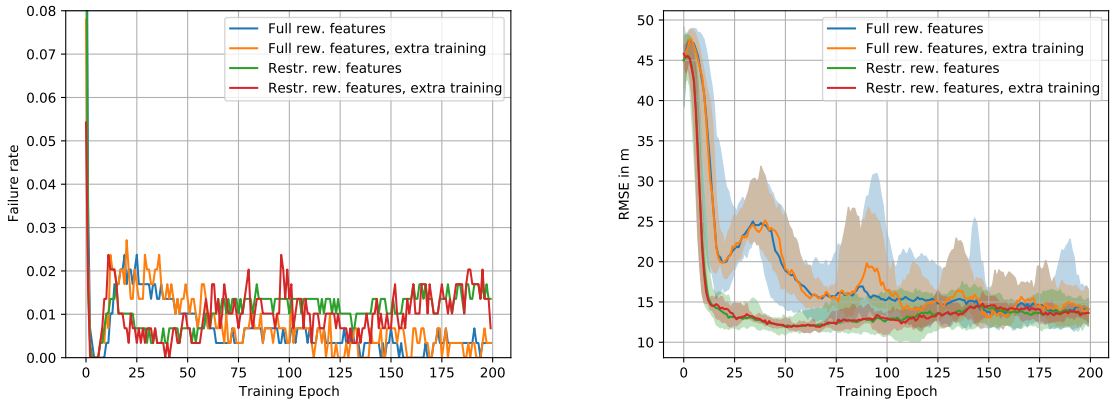


Figure 6.12: The fictional track used for training with vehicles placed according to the random initialization scheme



(a) Median failure rate

(b) Median, min. and max. RMSE

Figure 6.13: AIRL: median failure rate and RMSE of 8 s-predictions on the validation dataset during seven training runs. The plots show results for training with full or restricted discriminator input (reward) features, and for training with or without additional fictional situations.

the discriminator has access to the restricted features. Most configurations achieve prediction errors between $10\,\mathrm{m}$ and $15\,\mathrm{m}$.

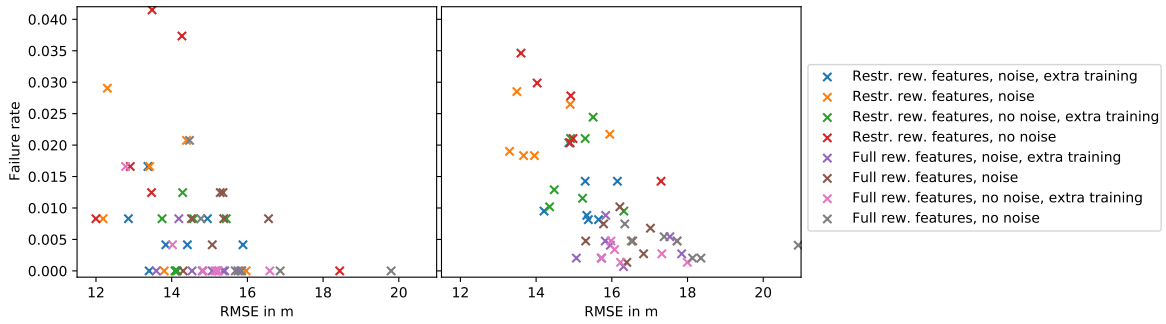
While the configurations with access to the restricted discriminator feature vector exhibit the lowest failure rates and RMSE after about 50 epochs before the failure rate increases again, the configurations with the full discriminator feature vector improve more slowly during the course of the training and achieve a median failure rate of 0 during multiple training epochs. The slower learning can be explained by the more challenging task for the discriminator of making its decision in a higher dimensional feature space, hence requiring more training data and epochs until it can reliably classify its inputs.

Contrary to the expectation, training in the additional critical situations does not improve the failure rate in the validation situations. One reason for this could be that empirically, most collisions occur between a vehicle that enters the roundabout and an inner-roundabout vehicle. The fictional situations in this section however are initialized to mainly provoke rear-end collisions. However, the additional training decreases the failure rate in the unseen test situations and hence can be seen as a method to improve the generalization of the model, as the evaluation in the next paragraph shows.

Comparison of All Training Configurations Similar to GAIL, different parameters are varied to determine the most successful training configuration. To this end, again, three binary decisions are compared: 1.) Providing the discriminator with access to the full or restricted feature vector. 2.) Whether additional noise on the discriminator inputs is used during training, to stimulate a smoother discriminator decision boundary. 3.) Training with or without the additional fictional training situations. The combination of these decisions leads to eight configurations. Each configuration is trained seven times to ensure the reliability of the evaluations. The final model of each training run is determined by selecting the model that has the lowest failure rate on the validation dataset. If there are multiple models with equal minimum failure rate, the model with the lowest RMSE is selected among those.

The results are depicted in Figure 6.14. As the majority of the models achieves a failure rate of 0 on the validation dataset, the analysis focuses on the performance in the unseen test situations in Figure 6.14b. Overall, many models now achieve failure rates lower than 1%, which is significantly lower than the corresponding GAIL models in Figure 6.8b. The final RMSE of most models is between $13\,\mathrm{m}$ and $18\,\mathrm{m}$, similar to GAIL. It appears that an inverse relationship exists between the RMSE and the failure rate of the examined models. Specifically, those models exhibiting lower prediction RMSE tend to demonstrate higher failure rates, and vice versa.

Similar to GAIL, the models which have access to the restricted discriminator features have lower RMSE values than those with the full discriminator features. However, the failure rate



(a) Validation performance **(b)** Test performance

Figure 6.14: AIRL: Final model performance of each training run of the eight configurations. The evaluation is performed for 10 s predictions on the validation dataset (a) and the unseen test dataset (b).

of these models is comparatively large, indicating that the model has not learned to respond appropriately to critical situations. Within the models with restricted discriminator features, the additional training in fictional situations decreases the failure rate at the expense of a higher RMSE. Training the discriminator with or without additional noise on its inputs does not seem to influence the training significantly.

The models with access to the full discriminator feature vector demonstrate the lowest failure rates, but have a slightly larger prediction RMSE than the models with restricted reward features. Due to the generally low failure rate, the visual improvement of training in additional fictional situations is only subtle in Figure 6.14. Comparing the median failure rate of the runs with full discriminator features with and without extra training, the failure rate is lowered from 0.47% to 0.27%, almost halving the remaining number of collisions.

For AIRL, it can therefore be concluded, that the most successful training runs are obtained when the discriminator has access to the full feature vector, and that the training in additional fictional situations further lowers the failure rate and improves the ability of the model to generalize to unseen situations.

6.5 Conclusion

This chapter demonstrates the application of IRL methods to the problem of modeling driver behavior. Hereby, the reward function that best explains the behavior of drivers in a dataset is reconstructed by iteratively adapting the reward function and training a policy that maximizes the rewards until the trajectories produced by the policy resemble the recorded trajectories.

Despite the algorithms originally being targeted at single-agent scenarios, this chapter shows that GAIL and AIRL can be adapted to the multi-agent task of predicting interacting traffic situations. For this purpose, the idea of parameter sharing from the previous chapter is

revisited. The indirection of reconstructing a reward function and performing RL with it enables the approaches in this chapter to pursue two goals: Learning a policy that closely resembles the behavior of the demonstrated trajectories, while observing the manually specified requirements, i.e., avoiding collisions or leaving the track.

Novelties Compared to the related works, the central innovation in this chapter is the *application of AIRL* to the problem of trajectory prediction. In contrast to conventional IRL methods with a linear combination of reward features, AIRL represents the reward function as a neural network. This is a more flexible and expressive representation that not only determines the relative importance of reward terms with respect to each other, but also learns the shape of the terms. Compared to GAIL, which has also been used in related works for the purpose of behavior modelling, AIRL enables the interpretation of the discriminator as a true reward function, which remains meaningful in different environments from the training environment.

As a consequence, AIRL enables the *training in additional fictional environments*. By setting up the fictional situations on a different map with critical initial states, the policy is demonstrated to generalize better to unseen situations and to cause fewer collisions and driving off track.

Combined with the semantically meaningful feature vector, known from previous chapters, as the input to the discriminator, the shape of the reward terms becomes interpretable, as demonstrated in Figure 6.11. This is a requirement for *understanding the goal that the policy tries to achieve* during training.

The use of the interpretable feature vector enables two further applications that are demonstrated in the experiments: First, it enables the *targeted restriction of information that the discriminator uses*, thereby allowing the developer to decide which features should not play a role in the reconstructed reward function. For example, some environment information in the feature vector might be of relevance for the policy, such as the current road curvature, but should not contribute to the decision of the discriminator whether it is observing real or simulated behavior.

Secondly, the interpretable feature vector enables to evaluate the similarity between the learned policy and the training data not only by directly comparing the trajectories, but also by *comparing the distribution of the features*, as demonstrated in Figure 6.9. This brings an additional insight into potential limitations of the learned policy, e.g., that it behaves more homogeneously than real world drivers, as it always tries to select one specific speed when driving inside the roundabout.

Results To answer the research questions from page 146, this chapter evaluates seven independent training runs of eight different configurations of each, GAIL and AIRL. Except from two outliers, all training runs successfully learn a policy that deviates from the test trajectories with a $10\,\mathrm{s}$-RMSE below $20\,\mathrm{m}$ and a failure rate below 5%. The training results are obtained within 200 epochs, taking $45\,\mathrm{min}$ to $1\,\mathrm{h}$, which is a negligible overhead compared to standard RL training with a known reward function.

The best GAIL models, trained with a restricted discriminator feature vector, achieve a prediction RMSE around $14\,\mathrm{m}$ with a failure rate around 1%. The failure rate is significantly lower than the best BC model with around 3% collisions. A more extensive comparison follows in the next chapter.

Figure 6.11 demonstrates that AIRL reproducibly reconstructs an interpretable and meaningful reward function. The reward is used for training in additional fictional environments, producing policies that exhibit a slightly larger RMSE around $16\,\mathrm{m}$, but that are significantly more robust with failure rates around 0.3%. In a direct comparison with AIRL policies that are not trained in additional fictional situations, the failure rate is approximately halved. However, the benefit of training GAIL with restricted discriminator inputs could not be reproduced for AIRL, where the best models are trained with the full discriminator feature vector.

An extensive comparison of the performance of the BC and IRL methods presented throughout this thesis is provided in the next chapter.

7 Comparison of All Models

After introducing different approaches to obtaining a driving policy in Chapters 4 and 6, this chapter evaluates the methods by comparing their prediction accuracy, plausibility and the ability to generalize to untrained situations. To assess the accuracy, the RMSE is compared. To evaluate the plausibility, the failure rate across different situations is measured. The ability to generalize is examined by contrasting these factors in situations that resemble the training data with situations that are different. To ensure a fair comparison, the same training and test data is used for all models. Also, the policy neural network architecture is identical.

This chapter is split into two parts. First, for a visual inspection, all models are executed in the same traffic situation with the ground truth evolution of the situation for reference. In the second part, the models are compared numerically. The performance is measured in one location where the model has been trained, and in three other locations from which no training data was used.

Multiple variations of the different methods were described in the respective chapters. Here, only the best configurations of all models are compared. To display the final variation in performance and to ensure reliability of the results, all evaluations are performed on seven independent training runs. The models compared are:

- Single-step training,
- Multi-step training, trained with $8\,\mathrm{s}$-trajectories,
- Multi-step training, trained with $16\,\mathrm{s}$-trajectories,
- GAIL, trained with the restricted discriminator feature vector, no additional steering input and additional noise during discriminator training, and
- AIRL, trained with the full discriminator input vector, without additional noise but with additional training in fictional situations.

7.1 Visual Comparison of the Model Performance

First, one of the seven final results of each model is executed in a traffic situation for a visual inspection of their prediction performance. The results are shown in Figures 7.1 and 7.2, including the ground truth evolution of the situation in Figure 7.1a. Overall, all methods provide a plausible prediction of the situation with all vehicles respecting right-of-way, waiting in queues at the roundabout entries, driving through the roundabout at adequate velocities, slightly cutting curves when entering and leaving the roundabout, and leaving the map at similar velocities. However, in the simulations with the single-step model and with the

16 s-multi-step model, one vehicle drives off the track and its simulation is thus terminated (vehicle #5 and #16).

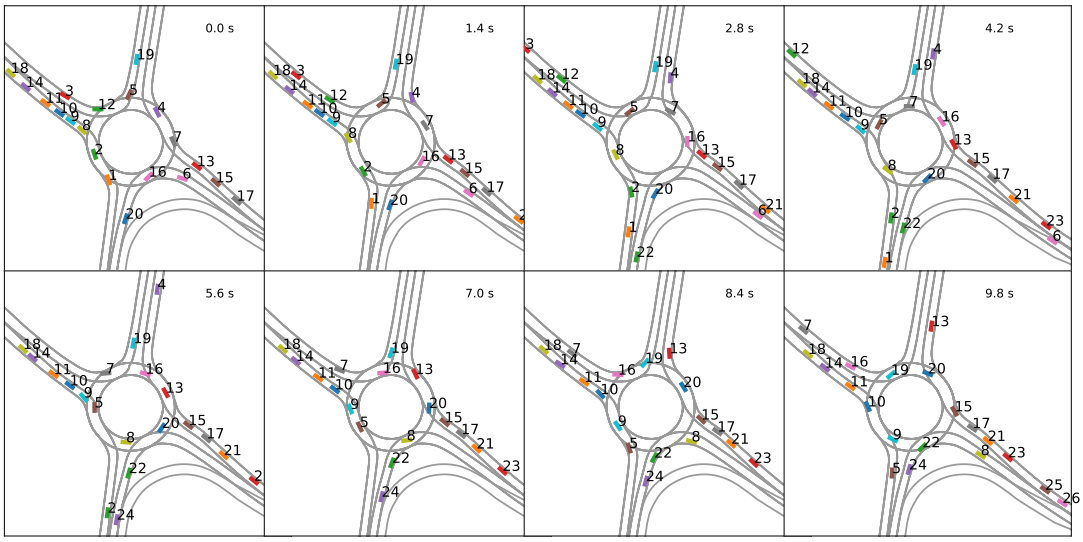
Considering individual vehicles, all predictions remain remarkably close to the ground truth evolution of the situation. For example, vehicle #1 and #12 leave the depicted frame in all predictions around 5 s. Vehicle #1 is closely followed by #2 after about 7 s. After 10 s, #7 is predicted close to the boundary of the field of view in all cases and #11 is still waiting to enter the roundabout. Coming from south, #20 is predicted to have made approximately a half turn in all predictions.

However, these predictions are all influenced relatively little by the predictions of other vehicles, as they do not need to negotiate right-of-way at the roundabout entrance. One vehicle that is subject to strong interaction is #9. After the vehicle in front of it has entered the roundabout, #9 could either merge in front of #5 around 2 s, or after #5. With hindsight, a large gap exists, but it remains unclear whether #9 could enter the roundabout, because the intention of the potentially conflicting vehicles #7 and #16 of leaving the roundabout becomes evident only relatively late. While the ground truth driver decides to enter the roundabout directly after #5, the BC driver enters the roundabout very slowly, moving only a few meters during the 10 s prediction. The MS-8 s and MS-16 s policy is more courageous and #9 also follows #5 closely. The GAIL and AIRL policies are more careful and wait until a larger gap for #9 emerges around 8 s, when clearly no conflicting vehicle is close. This is likely due to the secondary goal of avoiding collisions.

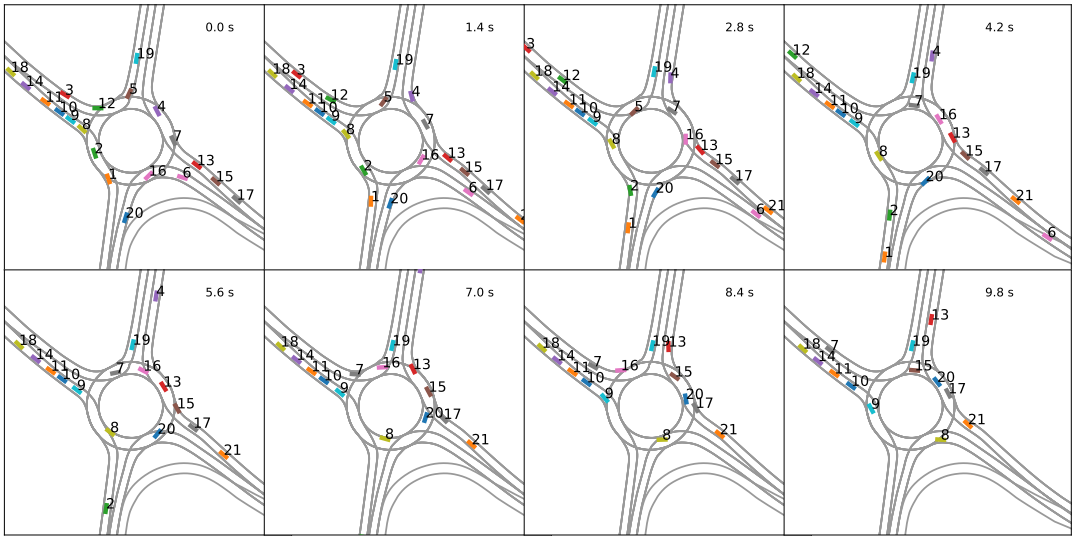
As discussed previously, an incorrect prediction of one vehicle can have knock-on effects on the prediction of other vehicles. For example, #20 is predicted to drive slightly faster by the AIRL model than by all other models, thereby disallowing #13 to enter the roundabout around 3 s, causing the final prediction of #13 to lag behind the ground truth position. Similar knock-on effects can be observed for other models, e.g., the interaction between #15 and #20 in the single-step model. Nevertheless, the prediction often stays impressively close to the ground truth, e.g., the fact that #19 merges into the gap between #16 and #20 after about 9 s is predicted correctly by three models.

7.2 Quantitative Evaluation

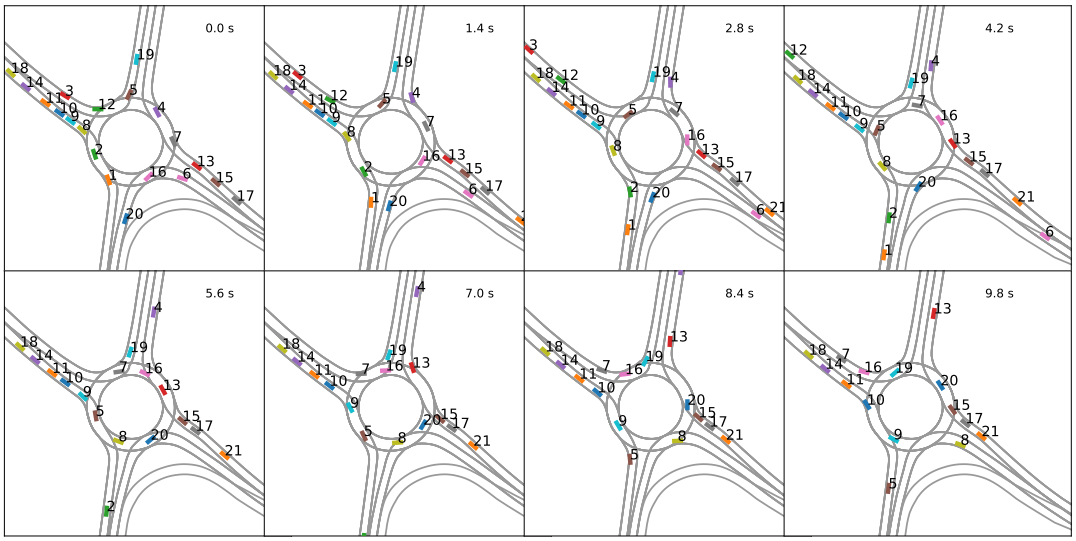
While the comparison in the previous section gives a visual insight on the plausibility of the policies, this section aims to make reliable statements on the overall prediction performance of the models. The central properties that the policies should fulfill are plausibility, accuracy, and generality. The plausibility is measured by evaluating the number of vehicles that collide or leave the track when following the policy, as these events rarely occur with real drivers. The accuracy is measured by comparing the prediction of the model to the true evolution of the same traffic situation and evaluating the deviation between both. As the dataset used in



(a) Ground truth evolution of the traffic situation

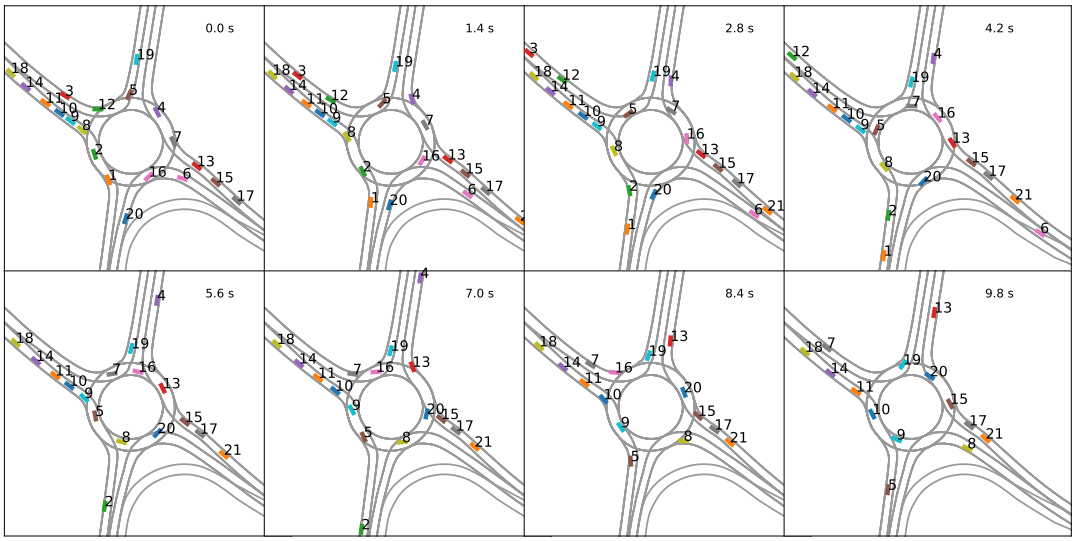


(b) Single-step behavior cloning model prediction

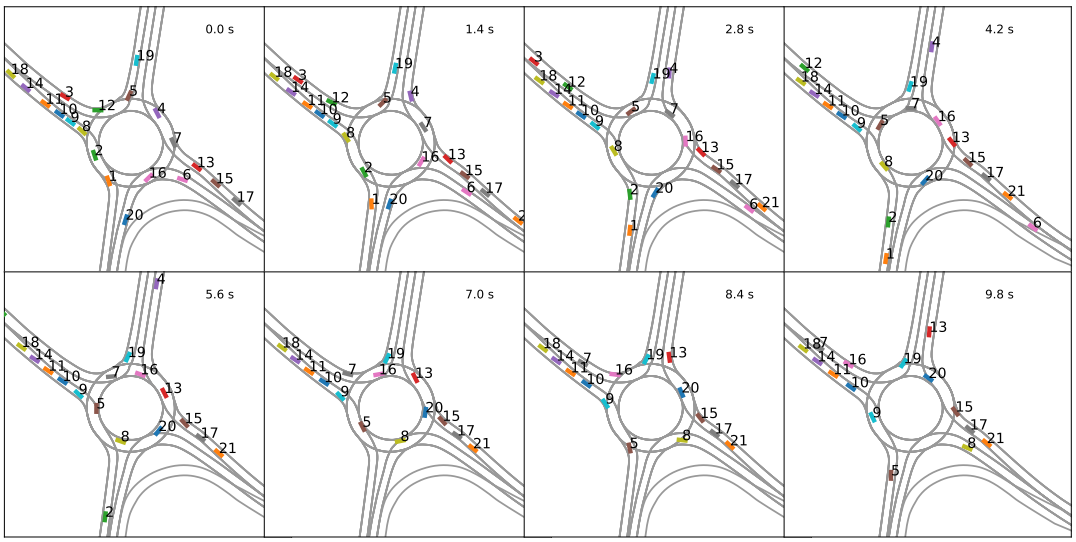


(c) Multi-step model prediction, trained with 8 s trajectories

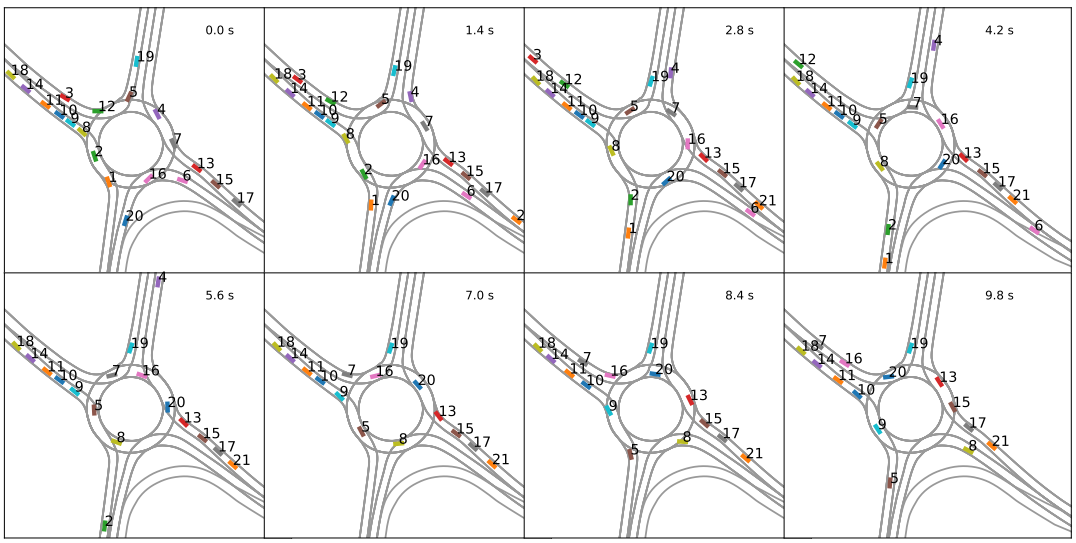
Figure 7.1: Prediction of the evolution of the same original situation using different models



(a) Multi-step model prediction, trained with 16 s trajectories



(b) GAIL model prediction



(c) AIRL model prediction

Figure 7.2: Prediction of the evolution of the same original situation using different models, continuation of Figure 7.1

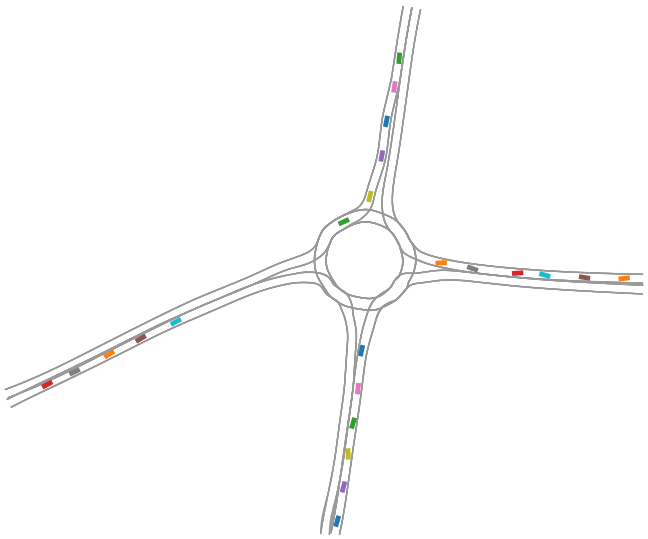


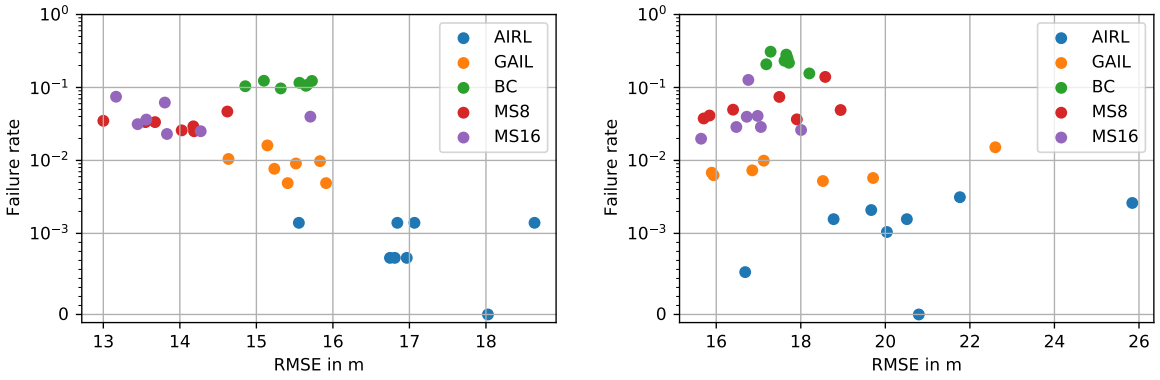
Figure 7.3: This map, based on a real untrained roundabout, is used exclusively for evaluating the models. The vehicles are placed according to a random initialization scheme. The depicted initialization has the maximum number of vehicles, other situations are initialized with fewer vehicles.

this thesis is limited to two roundabouts, it is difficult to make statements on the generality of the models. However, an attempt of estimating the ability to generalize to new situations is made by evaluating the performance in situations at one roundabout that was not used for training, and in additional fictional situations with randomly initialized vehicle states.

To this end, all seven training runs of each of the five models introduced in the beginning of this chapter are executed on four sets of $10\,\mathrm{s}$-situations:

- The test dataset, i.e., 118 unseen situations with 1432 trajectories in the training roundabout,
- the unseen roundabout dataset, comprising 402 situations with 1918 trajectories,
- 200 fictional situations with a total of 4955 vehicles on the map that was used during AIRL training,
- 200 fictional situations with 2501 vehicles at another untrained roundabout.

The two real-world roundabouts are shown in Figure 3.4. The map from AIRL training is shown in Figure 6.12. To be successful in it, the policies are required to have learned a behavior that is not specific to roundabouts: They need to handle regular right-of-way situations, drive on a winding course, drive on a slightly curved road and on a clockwise circular track. While the specific situations are initialized randomly and are hence unknown to the AIRL policy, it has been trained with similarly initialized situations on this map. For a fair comparison on a map that was not used in the training of any policy, the map of one additional real roundabout is populated with randomly initialized situations. The map is shown in Figure 7.3.



(a) Unseen situations in training roundabout

(b) Situations in untrained roundabout

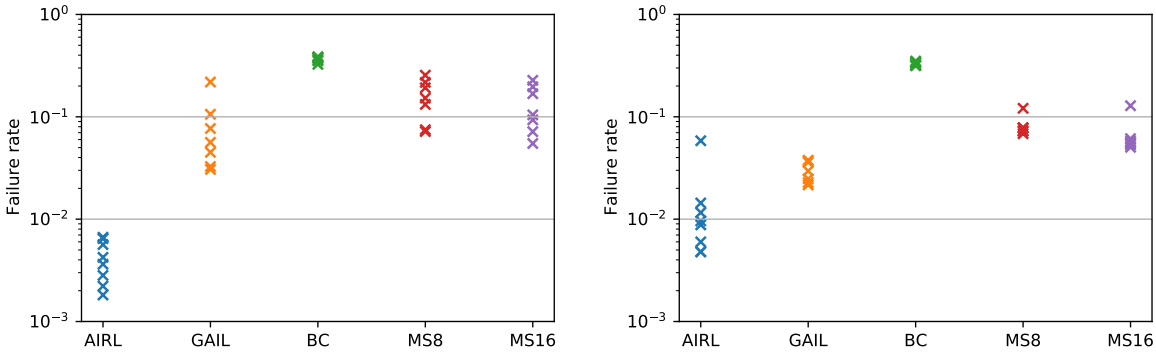
Figure 7.4: Prediction RMSE and failure rate of 10 s-predictions by the best variant of each model presented in this thesis. The performance is evaluated in situations that were not used for training. Observe that the failure rate is displayed on a combined linear-logarithmic scale, as the values vary by several orders of magnitude, and certain policies attain a failure rate of 0.

Performance in Real-World Situations Figure 7.4 shows the performance of the models in the two sets of unseen real-world situations, at the training roundabout and the unknown roundabout. First, the performance at the training roundabout is examined. The failure rate of the models varies by two orders of magnitude. The AIRL models exhibit the lowest failure rate, with around 0.1% of the trajectories colliding or leaving the track. The GAIL models have the second-lowest failure rate, with an average value around 1%. Then, the multi-step models follow with failure rates around 2-3% and some outliers with higher failure rates. Finally, single-step BC consistently displays failure rates around 10%. At the untrained roundabout, the values are similar, with a notable increase of the failure rate to 20-30% by the BC model.

Concerning the RMSE, the lowest values in the training roundabout are obtained by the multi-step models, between 13 to 15 m. Then, the GAIL and BC models follow with 15 to 16 m. Finally, the AIRL models exhibit the highest RMSE with values around 17 m and outliers to both sides.

Examining the RMSE on the untrained roundabout shows an interesting effect: The RMSE of all models increases significantly, compared to the known map. This indicates that the models have learned some nuances of driving behavior that are specific to the map, which boosts their performance in the known map. This effect is the most pronounced for the multi-step models, which now have similar RMSE values to GAIL and BC, around 16 to 18 m. Due to their goal of directly imitating the driver behavior, the multi-step models are likely the most prone to overfitting to map characteristics.

Especially concerning the ability to generalize, this comparison shows that the multi-step and the BC models are dominated by the indirect GAIL and AIRL methods: GAIL achieves



(a) Fictional situations from Figure 6.12 **(b)** Untrained fictional situation from Figure 7.3

Figure 7.5: Failure rate of the policies in entirely different situations with critical initialization (up to $14\,\mathrm{m/s}$ speed difference with $10\,\mathrm{m}$ distance to the preceding vehicle), for $10\,\mathrm{s}$-simulations

comparable RMSE values to MS and BC, but simultaneously has a significantly lower failure rate. AIRL shows an even lower failure rate, which comes at the cost of a slightly increased RMSE.

A possible explanation of this tradeoff between failure rate and RMSE is the following: As the policy drives more carefully, it has a bias to give way in unclear right-of-way situations, which slows down the overall progress in traffic situations, leading to the increased prediction RMSE. One example of this effect is visible for vehicle #9 in Figures 7.1 and 7.2: Both GAIL and AIRL wait for a large gap before entering the roundabout, whereas the ground-truth driver and the multi-step models enter the roundabout less hesitantly. Empirically, adding the angle between the heading of a conflicting vehicle and the merge point as an observation feature of the policy significantly improves the traffic flow, as it enables agents to detect that other vehicles are planning to leave the roundabout earlier. Real-world drivers can additionally rely on turn signals to communicate and observe intentions of other vehicles earlier than the kinematic state would allow. Thus, introducing additional means of communications between the agents, such as turn signals, would be an interesting extension of the simulation.

Performance in Fictional Situations The fictional situations are initialized randomly with different traffic densities, initial velocities ranging from $0\,\mathrm{m/s}$ to $14\,\mathrm{m/s}$, and minimum bumper-to-bumper distances between successive vehicles of $10\,\mathrm{m}$. Similar to the initialization during RL training, depicted on Figure C.2, some situations require an immediate strong braking, whereas others feature smooth driving conditions on unobstructed roads. To ensure comparability, the same 200 randomly initialized situations are used when comparing the models. Similar to Section 7.1, a visualization of the predictions of the models on the two fictional maps is shown in Appendix D.

Figure 7.5 shows the failure rate of the models on the two fictional maps. Overall, the failure rate ranges from 0.2% to around 40%, with all models performing significantly worse than

in the previously evaluated real-world situations. Two causes explain the decline in model performance: First, the policies are executed in situations which they have not been trained in. Second, the situations are typically more crowded and initialized more critically than the real world situations.

The first fictional situation in Figure 7.5a, which comprises multiple new traffic situations besides a roundabout, is especially challenging. Except from AIRL, which has been confronted with different situations on this map during training, all models show failure rates above 3%, up to 40%. Due to their training goal of directly imitating driver behavior, single- and multi-step training perform the worst. This shows that GAIL, BC, and MS-training clearly have difficulties to generalize to untrained road structures.

In the second fictional situation in Figure 7.5b, the GAIL and MS models have a lower collision rate, as the simulation takes place in a roundabout scenario. Still, with failure rates around 6-8%, single- and multi-step training have difficulties in plausibly predicting the situation. Due to the indirect training with the additional goal of avoiding collisions or leaving the track, GAIL achieves lower failure rates around 3%. AIRL, which is additionally confronted with fictional critical situations during training, further lowers the failure rate to an average of 1%.

7.3 Training- and Runtime

A comparison of the training duration until a good policy is found is interesting to shed light on the potential of the methods to scale to larger datasets. As the number of training epochs has different meanings for the different methods, it is not stated here. All runtimes are stated for a training on the same training dataset used throughout this work.

Given a pre-processed dataset of observations and reconstructed actions, the single-step model is the fastest model to train. As the dataset is comparatively small and the learning task is a standard supervised learning task that can be parallelized over the whole dataset, the training takes approximately 5 minutes on an NVIDIA RTX 2060.

Multi-step training takes significantly longer, as it needs to perform the backpropagation not only through a highly efficiently implemented off-the-shelf neural network, but also through the manually implemented simulation, as laid out in Section 4.2.1. Training each multi-step model takes approximately $1\,\mathrm{h}$ on the dataset. As training the $16\,\mathrm{s}$-model starts with a trained $8\,\mathrm{s}$-model, which is based on the final $4\,\mathrm{s}$-model, and so on, training of the $8\,\mathrm{s}$-model effectively takes $4\,\mathrm{h}$ and training the $16\,\mathrm{s}$-model takes $5\,\mathrm{h}$. In contrast to single-step training, the ability to parallelize the training is reduced, meaning that there is no benefit of executing the training on a GPU. Instead, the training is executed on a single core of an i7-9700 CPU @$3\,\mathrm{GHz}$.

IRL training does not require a differentiable simulation environment, which entails a faster forward simulation than multi-step training. On the other hand, there is no direct gradient information on how to improve the model. Rather, the policy gradient needs to be approximated stochastically by executing randomly sampled actions and assessing their consequences in terms of the obtained reward. Despite the notion that an analytic gradient computation, as performed during multi-step training, should overall be faster than a stochastic approximation thereof, the IRL methods require less training time: Training exclusively on the training dataset takes approximately $45\,\mathrm{min}$, and the additional training in fictional situations increases the training time to $1\,\mathrm{h}$.

Besides the methodological differences between the approaches, one possible explanation on why the stochastic gradient approximation by policy gradient methods is more successful than the exact analytic gradient computation is proposed in [Met+22]. The authors investigate failure modes of gradient based optimization in chaotic systems. One characteristic of chaotic systems is extreme sensitivity of the final outcome to initial conditions. While the experiments in this chapter show that traffic is generally well predictable for a few seconds, it sometimes exhibits strong sensitivity on certain states. For example, in an unclear right-of-way situation, a slightly larger velocity of one vehicle can be decisive on the evolution of the situation. Further, the evolution of the situation is non-deterministic and depends on the individual drivers. While an analytic gradient computation is possible in such a situation, it provides only the appearance of precision, as the gradient could be completely different when the initial situation is slightly changed. Metz et al. [Met+22] argue that a stochastic gradient approximation can be more robust and hence provide a better training signal in these cases. This also explains why multi-step training cannot be directly trained, but requires tweaks such as using pre-trained models and the Huber loss to achieve stable training performance.

When executing the learned models to simulate or predict a traffic situation, all approaches have an identical runtime. This is because the only difference is the weights of the policy neural network, regardless of the method that was used for its training. As described in Section 3.4, many optimizations are applied to maximize the simulation speed. Most importantly, the simulation of the traffic situation is implemented in a vectorized fashion, such that multiple situations can be simulated efficiently in parallel. This is important for fast training times for the multi-step and IRL models. For example, simulating 100 $10\,\mathrm{s}$-situations in parallel with a total of 1250 vehicles takes $4.5\,\mathrm{s}$ on a single core of an i7-9700 CPU. However, the focus of the implementation lies on fast *parallel* execution to quickly acquire experiences from multiple independent simulations. The implementation is not designed for simulating a single situation efficiently, such that it takes approximately $800\,\mathrm{ms}$ to predict a single $10\,\mathrm{s}$-situation with 25 vehicles. This is clearly too much for predicting the environment of an automated vehicle in real time. On the other hand, only $10\,\mathrm{ms}$ of this are used for the execution of the policy neural network, whereas the remainder of the time is mainly spent for computing transformations between Cartesian and Frenet coordinates, determining the relation between vehicles, and

computing the components of the observation vector. A more efficient implementation of the simulation in a compiled language such as C++ instead of Python could conceivably significantly reduce the computation time.

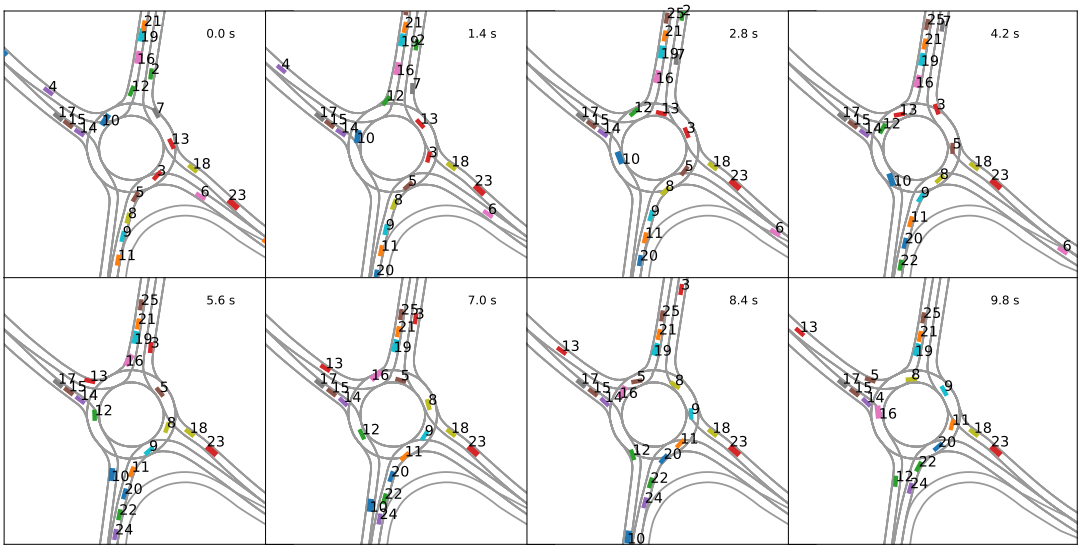
7.4 Conditional Prediction

One important motivation behind this work is the ability to make conditional predictions. In the planning system of an automated vehicle, such predictions are useful for cooperative behavior planning: By conditioning the prediction of the situation on a planned ego trajectory, the automated vehicle could assess the effect of its plan not only on itself, but also on the predicted trajectories of the surrounding vehicles.

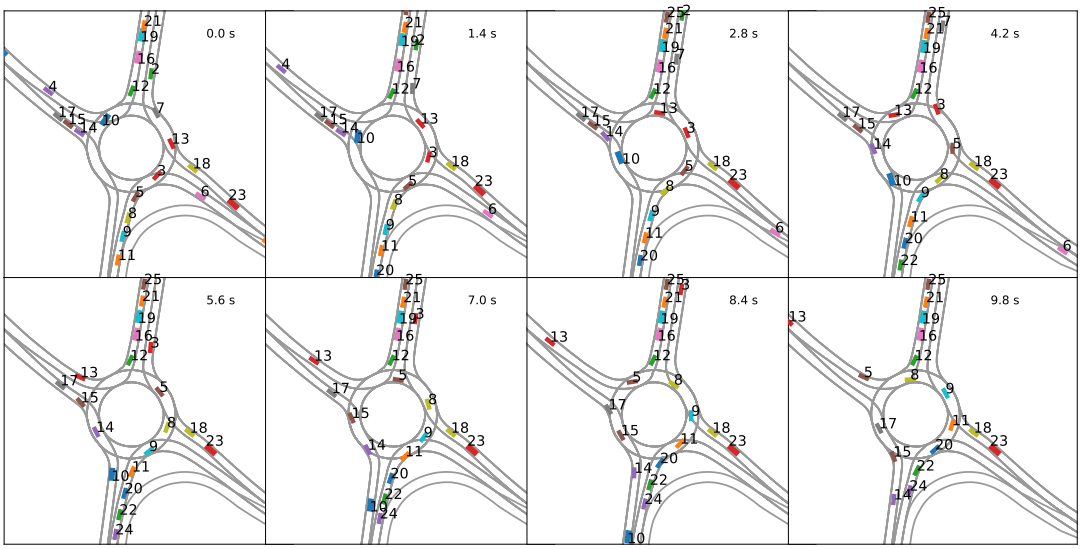
As an investigation of cooperative behavior planning is beyond the scope of this work, this section only shows a proof-of-concept of the idea of conditional prediction: In a situation where a vehicle could enter a roundabout, one prediction of the situation is issued for the case that it regularly enters the roundabout, and another prediction is issued for the case that the vehicle stops at the stop line. Similarly, the behavior planner of an automated vehicle could receive multiple future predictions by querying the prediction model with different planned trajectories for itself.

The resulting predictions are depicted in Figure 7.6. The original prediction in Figure 7.6a is generated by executing one AIRL policy for all vehicles. Any other policy could have been used for this purpose. To simulate the conditioning on an alternative future trajectory of vehicle #12, where it does not enter the roundabout, the same simulation is repeated with all vehicles controlled by the AIRL policy, except for #12, whose acceleration actions are simply replaced with a constant strong braking. The resulting conditional prediction is shown in Figure 7.6b. Except for #12 and its succeeding vehicles coming from north, the prediction of all other vehicles remains similar for about $3\,\mathrm{s}$. Then, the predictions start to diverge: If #12 does not enter the roundabout, a gap for the vehicles coming from west emerges, such that they can enter significantly earlier. Also, #13 is predicted to drive slightly slower when #12 enters the roundabout before it. Many other vehicles remain largely unaffected by the actions of #12: The final position of #8, #9, #11, #18, #20, #22, #23 #24 after $10\,\mathrm{s}$ is almost identical.

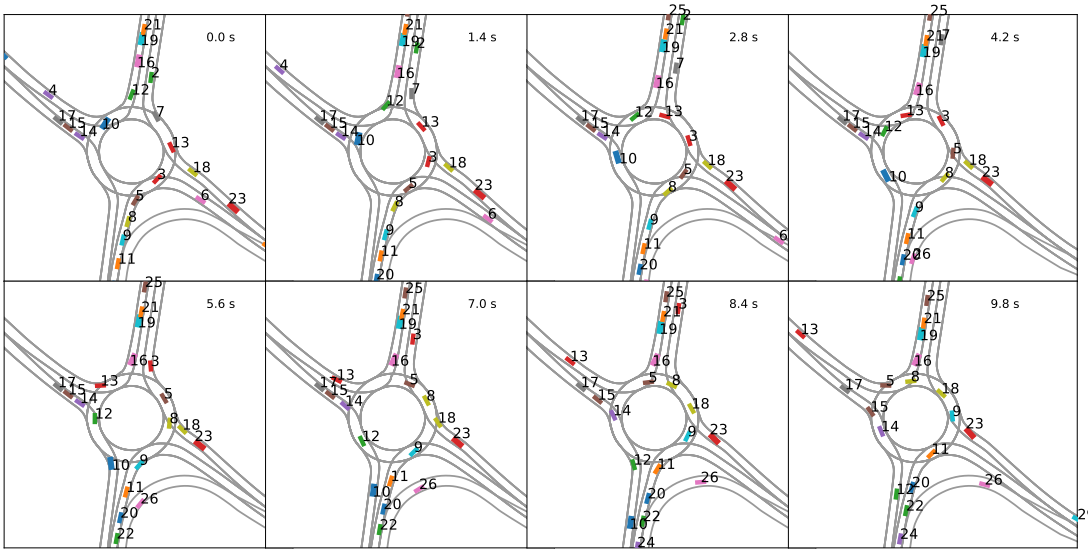
For reference, the ground truth evolution of the situation is shown in Figure 7.6c. There, #12 indeed enters the roundabout, and the situation evolves similar to the prediction for approximately $5\,\mathrm{s}$.



(a) What if #12 enters the roundabout at the next opportunity?



(b) What if #12 does not enter the roundabout?



(c) Ground truth evolution of the traffic situation

Figure 7.6: Demonstration of a conditional prediction

7.5 Conclusion

This chapter compares the prediction performance of the best variants of the proposed approaches to learning a policy. Between the approaches, only the training differs, whereas the execution is always the same: For the prediction of a traffic situation, all policies are executed in the same simulation structure, described in Chapter 3. Hence, the runtime of all approaches is identical. Also, due to the simulation-based architecture, every approach is capable of issuing conditional predictions, as outlined in Section 7.4.

All methods successfully handle untrained situations in the training roundabout, but single- and multi-step training struggle to plausibly control vehicles in the fictional situations. In the training roundabout, single-step training exhibits the highest failure rate around 10%. As expected, multi-step training improves the failure rate and the prediction RMSE, which can be explained by the model being more robust to causal confusion and distributional shift, as discussed in Chapter 4.

The IRL based methods GAIL and AIRL do not directly try to imitate the ground truth trajectories, but rather follow an indirect approach—by reconstructing the reward function and finding a policy that maximizes this reward. In the training roundabout, this indirect approach leads to an increased prediction RMSE, but it allows reducing the failure rate by a manually specified secondary goal of avoiding collisions or driving off the track. In the untrained roundabout, the better accuracy of the multi-step models compared to the IRL based models vanishes, which indicates that the direct multi-step models have learned a policy that is over-adapted to the training roundabout. Moreover, in the two fictional situations, the IRL based methods demonstrate the lowest failure rates.

Overall, it can be concluded that the indirect IRL methods are superior to direct single- or multi-step training. In situations that are different from the training situation, GAIL exhibits lower failure rates at equal accuracy, and AIRL shows even lower failure rates at a slightly decreased accuracy. Moreover, the training time of the IRL methods is significantly lower and AIRL provides means to extend the training to fictional situations for which no training data exists.

8 Conclusion

An automated vehicle should drive safely and comfortably. For this, it needs to make plans on how to act in the next seconds. To make these plans, a model of how the environment around the automated vehicle evolves is required. Specifically, it is of interest how the surrounding vehicles will act in the near future. For this reason, this work is concerned with predicting their trajectories.

8.1 Summary

In contrast to many other recent publications in trajectory prediction surveyed in Section 2.4, this work generates trajectory predictions by executing a driver behavior model for each vehicle in a microscopic simulation of the current traffic situation, described in Chapter 3. This approach has multiple advantages, the central ones are:

- It naturally enables the predictions to interact with each other: The simulated vehicles observe each other in the simulation, such that complex traffic patterns can be predicted. For example, as illustrated in Section 7.2, vehicles are predicted to wait at the roundabout entry until a sufficiently large gap emerges between two other predicted vehicles.
- It enables the prediction model to issue conditional predictions, as demonstrated in Section 7.4. This can be used to answer questions such as "How would the situation evolve if the automated vehicle enters the roundabout? How would it evolve if the automated vehicle stops at the roundabout entry?"
- At its core, the learned model controls a kinematic vehicle model. Hence, the predictions are guaranteed to be physically plausible. Also, the prediction is not merely a sequence of future positions, but rather the full vehicle state, including the velocity, acceleration and heading of other vehicles. Combined with the conditional prediction, this can be used to assess the impact that the plan of an automated vehicle has on others, e.g., "How strong does another vehicle need to brake if the automated vehicle enters the roundabout ahead of it?"

As lined out in Section 2.1.1, these properties of the prediction pave the way for implementing a cooperative behavior planner for an automated vehicle: The behavior planner could compare the influence of different future ego trajectories on other vehicles and select the trajectory that is optimal not only for itself, but also for others. Moreover, as lined out in Section 2.1.2, the driver behavior models cannot only be employed for predicting traffic situations, but also for a

realistic traffic simulation to test the performance of the cognition component of an automated vehicle.

Despite these advantages, one major drawback has hindered progress in the class of simulation-based prediction models: There is no straightforward way to learning a driving policy that generates accurate and plausible long-term predictions when executed by all vehicles in the simulation. This is the central motivation of this thesis, which investigates different approaches to learning a driving policy. In the simulation framework described in Chapter 3, the policy is the component that controls the behavior of a vehicle: It determines which action (acceleration, steering) a vehicle selects, based on a simulated observation of its local environment. The key contributions that this thesis makes towards learning a driving policy are emphasized with bold letters in the following.

The baseline approach, single-step Behavioral Cloning, works by generating a dataset of simulated observations and reconstructed actions of vehicles from an original dataset of recorded trajectories. Then, a neural network is employed to establish the relation between observations and actions. Details are described in Section 4.1. While this is seemingly a straightforward solution to learning a policy, the central problem of single-step training is that it does not learn a causal model of the relation between observations and actions, and that it suffers from distributional shift: When the learned policy is executed in the simulation, it is prone to slowly drifting out of the domain of observations in the training dataset. As the learned policy is only capable of selecting appropriate actions in situations that are similar to the training situation, this leads to an inherent instability of the model, as discussed and supported by multiple sources in Section 4.1.3.

This motivates the proposal of **multi-step training** in Section 4.2. Instead of only predicting the next action, the idea is to execute the policy in the simulation during training, and to use the deviation between the predicted and the real trajectory as the training loss that shall be minimized. The loss is minimized by following its gradient with respect to the policy parameters. To compute this gradient, a **differentiable simulation environment** is required, where the output of each component—observation, policy and kinematic model—can be differentiated with respect to its inputs. During multi-step training, the policy experiences the consequences of its actions on subsequent observations and learns to compensate for them. The experiments in Section 4.3.3 and Chapter 7 show that this leads to a decreased prediction RMSE and to an improved ability of the policy to remain on the track and to avoid collisions, compared to single-step training.

However, even the best multi-step models exhibit failure rates around 2-3%. Moreover, as their training goal is the direct imitation of the trajectories from the training dataset, all Behavioral Cloning approaches can only be trained in situations for which training data exist. This idea leads to the exploration of an entirely different class of algorithms to obtain a driving policy, namely Reinforcement Learning. Hereby, the same simulation environment as before

is used. However, the goal is now to find a policy that maximizes a manually specified reward function. This happens entirely in simulated traffic situations and requires no real world training data.

At its core, the employed PPO algorithm to find an optimal policy operates by randomly selecting different sequences of actions, evaluating the obtained rewards, and increasing the chance of selecting actions that lead to high rewards, compared to those that lead to low rewards. Iteratively, this leads to a policy that obtains increasingly larger rewards. As most RL algorithms target single-agent scenarios, Chapter 5 describes the necessary adaptions to **apply RL to multi-agent roundabout traffic situations**. Also, a **new reward function** is derived under which a vehicle needs to maintain a specified lateral acceleration in curves to maximize its reward. It is demonstrated that the learned policy indeed acts close to this theoretical reward-maximizing optimum, confirming the effectiveness of the learning algorithm. For the multi-agent case, the reward function is extended to reflect cooperative driving behavior and the observance of right-of-way rules.

Furthermore, a method for **learning a heterogeneous policy** that can represent different behaviors is proposed. Depending on three special preference inputs, the policy can be changed on the fly, for example to drive faster or slower through curves, or to maintain larger or smaller safety distances to other vehicles. The motivation behind this is to capture the heterogeneity of real world driver behavior, which a single fixed policy cannot represent.

However, when RL is employed to learn a policy that resembles the behavior of human drivers, the reward function must accurately model the goals of human drivers, as well as their relative weights. This reward function is generally unknown and hard to model manually. As an alternative, this work investigates two algorithms, GAIL and AIRL, to effectively automate the search for the reward function. The algorithms operate in an adversarial learning framework, described in Chapter 6, where a second neural network denoted as discriminator has the task of distinguishing between real trajectories and trajectories generated by the policy. Concurrently to training the discriminator, the policy is trained using the PPO algorithm with the goal of maximizing the "realness" score assigned by the discriminator. This can be interpreted as a game between discriminator and policy: As the discriminator learns to identify the real trajectories more confidently, it simultaneously provides a stronger training signal to the policy, which in turn learns to generate more realistic appearing trajectories.

This thesis proposes **adaptions of GAIL and AIRL for the multi-agent setting**, because the original algorithms are designed for the single-agent setting. For this, the parameter-sharing idea from the RL chapter is revisited. **AIRL is used to reconstruct the reward function**, and a visualization of the learned function is provided, demonstrating that it has an interpretable meaning: Progress along the track is rewarded, whereas lateral and longitudinal accelerations are penalized approximately quadratically.

The indirect IRL approach of simultaneously reconstructing a reward function and training the policy to maximize it has multiple advantages over trying to directly imitate the driver behavior, as multi-step training does: Additional rewards can be manually specified, such as the goal to avoid collisions and to remain on the track. As a consequence, the learned GAIL policies achieves the same accuracy as the best multi-step policies while exhibiting a lower collision rate. Moreover, the reconstructed reward can be used to train in arbitrary simulated situations, for which no training data exists. This is exploited during the AIRL training to **additionally train in fictional critical situations**, where the policy is forced to immediately brake strongly to avoid a collision. With this, AIRL further reduces the collision rate at the cost of a slightly decreased accuracy, compared to GAIL.

Ultimately, Chapter 7 contains an **extensive comparison of all models**, trained and evaluated on the same datasets. The evaluation shows that the policies obtained with GAIL and AIRL exhibit the best prediction performance in terms of accuracy, plausibility, and ability to generalize to new situations.

8.2 Limitations and Future Work

Intention Prediction This work is exclusively concerned with behavior modeling via a driving policy. All presented approaches assume that the route intention of the predicted vehicles is known in advance, e.g., whether a vehicle will leave a roundabout at the next exit. To truly predict the future trajectories of other vehicles, their intention needs to be predicted as well. This could be realized by using a separate intention detection mechanism, as proposed in [**Sac+20a**; **Vog+20**], which is not covered in this work.

In unclear cases, multiple trajectory predictions would need to be issued to cover all options. As the models proposed in this work are interacting in a consistent world, each branching of options of a vehicle would create additional predicted situations, e.g., one where the vehicle leaves the roundabout and one where it does not. This leads to a combinatorial growth of the number of situations that need to be predicted to cover all options of all vehicles. Pruning these options to predict only the relevant ones from the perspective of an automated vehicle is an important step for making the models applicable. Many promising ideas concerning this problem are presented in [Sch21].

Heterogeneity of Agents In Section 5.5, a RL-based method to learn a policy that can exhibit different driving characteristics is proposed. By letting the policy observe some features of its reward function, e.g., the minimum allowed time gap to the preceding vehicle before a penalty is assigned, the policy learns to adhere to these preferences. An interesting direction for future research is the combination of this idea with IRL-based approaches to learn a flexible policy that exhibits different realistic behaviors, depending on specific preference

inputs. These preferences could also be estimated on-line to issue driver-specific predictions after observing the past driver behavior. Another interesting future research direction is to learn policies for the interaction between completely heterogeneous agent classes, such as cars, trucks, pedestrians and cyclists.

Handling Uncertainty Behavior planning can benefit from a representation of the uncertainty of the prediction. As discussed in Section 2.5, no surveyed approach is capable of emitting the closed form density of all vehicle states at all future time steps. The reason is that no simple density exists that captures the complex interaction between successive and interacting states of different vehicles. Instead, it is more straightforward to issue multiple representative predictions of the potential evolution of the traffic situation. To generate a diverse set of predictions, ideas from the previous two paragraphs can be leveraged: The predictions need to cover all potential future routes. Along a route, multiple realistic trajectories could be predicted by sampling from a low-dimensional preference space to represent the potential heterogeneity of driving behavior. Effectively, this transforms random samples from a relatively low-dimensional space (preferences, route intentions of all agents) to samples in the high-dimensional space of the interacting future states of all vehicles in the traffic situation.

Graph-based Environment Representation The policies proposed in this work operate on a hand-defined observation vector described in Section 3.2. While it has been demonstrated that this enables the different algorithms to learn policies that successfully handle roundabout situations and gives insight into their functioning, it has the disadvantage that it is restricted to situations that can sensibly be described with this representation. For example, in a subsequent work for predictions on a highway [**Rad+23**], a different observation vector is required, which also means that a new policy needs to be trained. A more flexible input representation could be realized using graph neural networks that can process arbitrary map layouts and a varying number of surrounding vehicles. In recent years, GNNs have been successfully employed for supervised learning style vehicle trajectory prediction, for example in [Gao+20; Lia+20]. A combination with policy-based prediction approaches is an interesting direction for future research. A first step in this direction is made in the associated publication [**Kon+23**], where a RL policy is trained with a graph-based observation of the environment.

Planning by Prediction Another promising future research direction is to use the learned policy to boost sampling-based behavior planning algorithms such as on-line POMDP solvers, for example employed in [Hub+18; **Bey+21b**]. As demonstrated in Chapter 5, RL is capable of learning a policy that exhibits behavior that is close to the optimal behavior. However, while it maximizes the expected mean return, is does not rule out rare worst-case scenarios, such as

collisions. These occur throughout all examined RL and IRL approaches for at least 0.1% of all trajectories. Guiding the search of a sampling-based behavior planner with the actions predicted by the policy would perform at least as good as directly following the policy, but can search for better trajectories on-line, for example to avoid collisions or to further increase the obtained reward. Tree-search based behavior planning, on the other hand, would benefit from this combination, because the policy could guide its search towards selecting promising actions, thereby reducing the number of trajectories that are evaluated, or increasing the search depth of the planner. A combination of these methods has been demonstrated in [Sil+17] for the board game Go, and could likely be transferred to the domain of automated driving with methods that handle continuous observations and actions, similar to those presented in this thesis.

Scaling Finally, as for all data-based approaches, it would be interesting to investigate how the proposed methods scale to larger datasets that encompass a wider variety of situations. Also, for a large-scale application of the methods, it is important to investigate on-line learning systems that improve their predictions over time, as new data is continuously collected by a fleet of vehicles.

A Evaluation of the Dataset Accuracy

To evaluate the accuracy of the DFS dataset, a test vehicle equipped with a highly precise iTraceRT-F400[1] localization unit that estimates the vehicle state via a deeply coupled Inertial Navigation System (INS) and Global Navigation Satellite System (GNSS) is used. The test vehicle stores the iTrace state estimates and is simultaneously recorded in the drone video. The video is subsequently processed by the DFS pipeline. As the errors of the iTrace are expected to be an order of magnitude smaller than the errors of the DFS pipeline, the iTrace is assumed to be the ground truth.

To compare the two measurements, the original iTrace measurement is downsampled to $29.97\,\mathrm{Hz}$, the frequency of the video recording, using linear interpolation. Then, a temporal synchronization

$$i^* = \underset{i}{\operatorname{argmin}} \sum_k (v_{\text{gt}}[k] - v_{\text{dfs}}[k+i])^2 \tag{A.1}$$

of the two measurements is performed via the time-discrete speed signals from the iTrace $v_{\text{gt}}[k]$ and from the DFS data $v_{\text{dfs}}[k]$. For all further evaluations, all DFS signals are shifted by i^*, where the RMSE between the two speed signals is minimal. The aligned signals are depicted in Figure A.1 and show that the temporal alignment was successful. The vehicle is not permanently visible from the drone, which explains the gaps in the DFS signal.

Speed The DFS speed fits well with the iTrace measurement, except for approximately $1.5\,\mathrm{s}$ after the vehicle enters or leaves the field of view of the drone. This can probably be attributed to an initialization error of the DFS tracking pipeline. Therefore, the first and last $1.5\,\mathrm{s}$ of each trajectory are excluded from the dataset.

For the remaining parts of the trajectory, the average speed provided by DFS is $0.027\,\mathrm{m/s}$ higher than the iTrace speed with a standard deviation of $0.114\,\mathrm{m/s}$. The error histogram is shown in Figure A.2a. The relative errors $(v_{\text{gt}} - v_{\text{dfs}})/v_{\text{gt}}$ are typically below 2.5%, excluding velocities below $1\,\mathrm{m/s}$. These results are in line with [Bar+19], who also evaluate the speed estimate of the DFS pipeline and state that the speed error does not exceed $1.2\,\mathrm{km/h} = 0.33\,\mathrm{m/s}$ when the drone footage is properly calibrated and stabilized. Interestingly, the velocity error is competitive with state-of-the-art object tracking methods of surrounding vehicles. For a LiDAR based system, [Krä21, p. 129] reports a standard deviation of $0.43\,\mathrm{km/h} = 0.119\,\mathrm{m/s}$ for the velocity error of the best tracking model and sensor. For a

[1] Datasheet available at `https://www.imar-navigation.de/downloads/TraceRT-F400-E_en.pdf`; accessed July 4, 2022

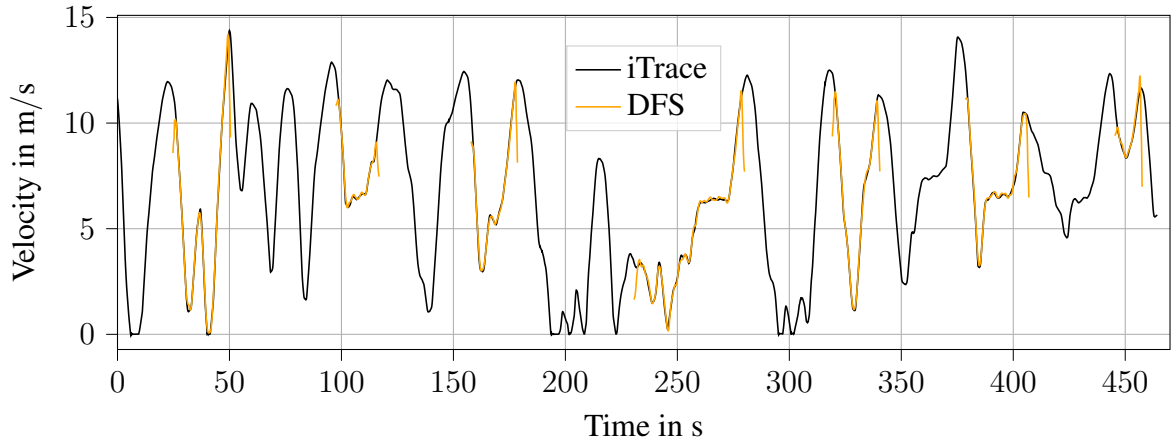


Figure A.1: Time-aligned speed signals of the test vehicle from iTrace and DataFromSky pipeline

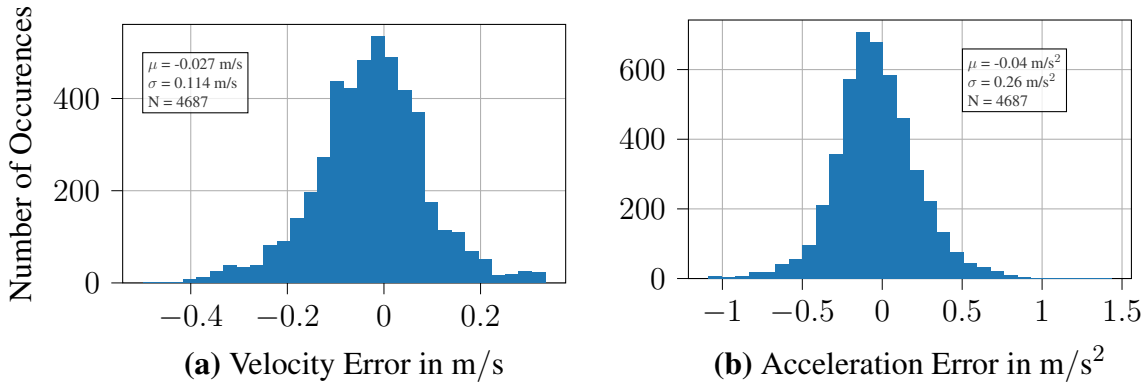


Figure A.2: Error Histograms of $v_{\text{gt}} - v_{\text{dfs}}$ and $a_{\text{gt}} - a_{\text{dfs}}$

radar based system, [Sch19, p. 111] reports a RMSE[2] of $0.17\,\mathrm{m/s}$ for the best model and scenario.

Acceleration As the DFS pipeline needs to reconstruct the acceleration from positional observations in the video as a second order derivative, errors are to be expected. On average, the acceleration provided by DFS is $0.04\,\mathrm{m/s^2}$ larger than the ground truth, with a standard deviation of $\sigma = 0.26\,\mathrm{m/s^2}$. The error histogram is provided in Figure A.2b. The relative errors are not informative in this case, because the longitudinal acceleration is often close to $0\,\mathrm{m/s^2}$ during regular driving.

[2]While the RMSE cannot directly be compared to the standard deviation, the RMSE of the DFS velocity is approximately $0.117\,\mathrm{m/s}$. See Appendix B.1 for details.

Positional Error Both, the DFS and iTrace positions, are provided in a world-fixed cartesian coordinate frame. However, as the coordinate frame of the drone videos is not earth-fixed, the frames are not aligned. Thus, the translation and rotation between the frames needs to be determined first. The temporal alignment (A.1) via the velocities is maintained unchanged.

To facilitate the ensuing alignment procedure, the positions of both, the iTrace and DFS trajectory, are shifted

$$(\tilde{x}_{\text{gt}}, \tilde{y}_{\text{gt}}) = (x_{\text{gt}} - \bar{x}_{\text{gt}}, y_{\text{gt}} - \bar{y}_{\text{gt}}) \tag{A.2}$$

$$(\tilde{x}_{\text{dfs}}, \tilde{y}_{\text{dfs}}) = (x_{\text{dfs}} - \bar{x}_{\text{dfs}}, y_{\text{dfs}} - \bar{y}_{\text{dfs}}) \tag{A.3}$$

such that the respective mean trajectory position $(\bar{x}, \bar{y})$ is the origin of a new coordinate frame. Both, the iTrace trajectory mean and the DFS trajectory mean are calculated for the synchronized time steps where the vehicle is visible in the drone video.

Next, the DFS coordinate system is translated and rotated according to

$$(\phi^*, x^*_{\text{off}}, y^*_{\text{off}}) = \underset{\phi, x_{\text{off}}, y_{\text{off}}}{\operatorname{argmin}} \sum_k \left\| \begin{pmatrix} \tilde{x}_{\text{gt}}[k] \\ \tilde{y}_{\text{gt}}[k] \end{pmatrix} - R_\phi \begin{pmatrix} \tilde{x}_{\text{dfs}}[k] + x_{\text{off}} \\ \tilde{y}_{\text{dfs}}[k] + y_{\text{off}} \end{pmatrix} \right\|^2 \tag{A.4}$$

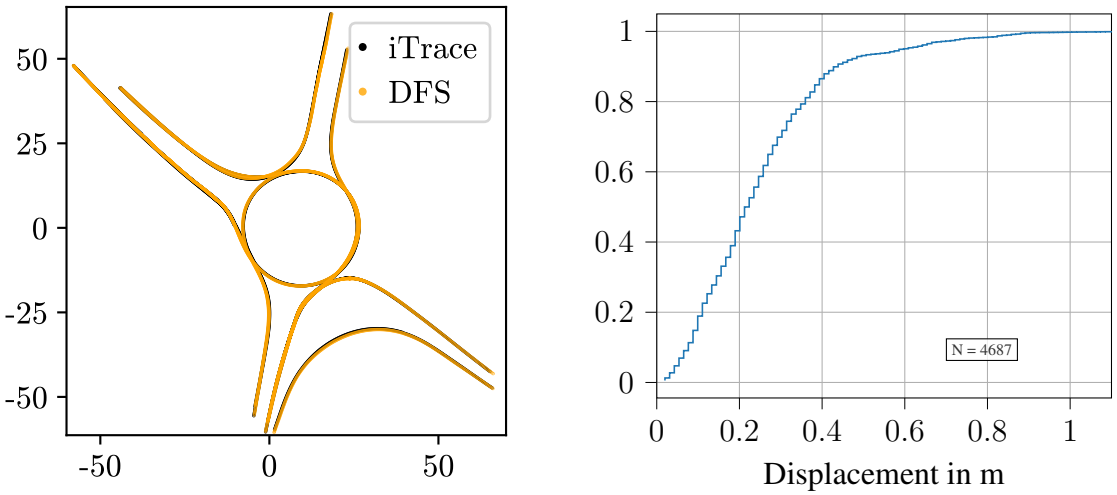
with the rotation matrix

$$R_\phi = \begin{pmatrix} \cos\phi & -\sin\phi \\ \sin\phi & \cos\phi \end{pmatrix} \tag{A.5}$$

such that the RMSE between the two trajectories is minimal with respect to the rotation angle ϕ^* and translation $(x^*_{\text{off}}, y^*_{\text{off}})$. As this optimization needs to be performed only once, it is implemented using a simple grid search. The aligned trajectories are shown in Figure A.3a.

Now, the statistics of the displacement error can be computed, which characterizes the remaining mismatch between the aligned trajectories. The mismatch is the result of localization errors of the vehicle detection mechanism in the DFS pipeline, and of intrinsic and extrinsic camera calibration errors. The intrinsic calibration errors occur for example due to incorrect lens distortion correction, and the extrinsic calibration errors are a result of the drone hovering not entirely stable in the air, for example due to wind. The DFS pipeline addresses each of these phenomena [Ape+15], but cannot entirely compensate them.

The displacement between the positions obtained via iTrace and the complete DFS pipeline is shown in the cumulative displacement histogram in Figure A.3b. It shows that the displacement error is below $0.25\,\text{m}$ in 50% of all measurements, and below $0.6\,\text{m}$ in 95% of all cases. The average displacement error is $0.25\,\text{m}$.



(a) Aligned trajectories

(b) Cumulative displacement histogram

Figure A.3: Trajectories after alignment

A.1 Dataset Statistics

Figures A.4 to A.7 show the distribution of the observation features listed in Table 3.1. Figure A.8 shows the distribution of the acceleration and steering action. Before processing the features with a neural network, these statistics are used to standardize the features such that each feature has a mean of 0 and a standard deviation of 1. To avoid leaking any information from the test dataset, the mean and standard deviation value of each feature is exclusively computed on the training and validation dataset. For this reason, the histograms show the distribution of the data in these two datasets.

Not all features are available at all timesteps. For example, not all vehicles have a preceding vehicle or the preceding vehicle is out of range. The value is then replaced with the default or maximum value from Table 3.1. These cases are excluded from the histograms for better visibility. The mean and standard deviation are computed on the full dataset, including the imputed values. For this reason, the approximate mean of the histograms and the value listed in Table 3.1 might differ. Some features can assume slightly negative values: The distance to the preceding vehicle sometimes is negative due to misdetections in the dataset. The distances to the yield or merge point become negative, because they are measured from the vehicle front and computed until the vehicle center has passed the respective point.

Some features require more explanation. The distance to the preceding vehicle in Figure A.6 occasionally assumes values below 0 due to displaced detections or wrong vehicle shapes estimates in the drone recording. The distance to the yield point, also in Figure A.6, is clipped when the vehicle center is behind the yield line. As the distance is calculated from the vehicle front, it can become slightly negative. The same applies to the distance of the closest conflicting vehicle to the yield point. The angle of the closest conflicting vehicle to the merge

point shows two clear clusters that can be used to differentiate whether a vehicle will leave the roundabout at the next exit.

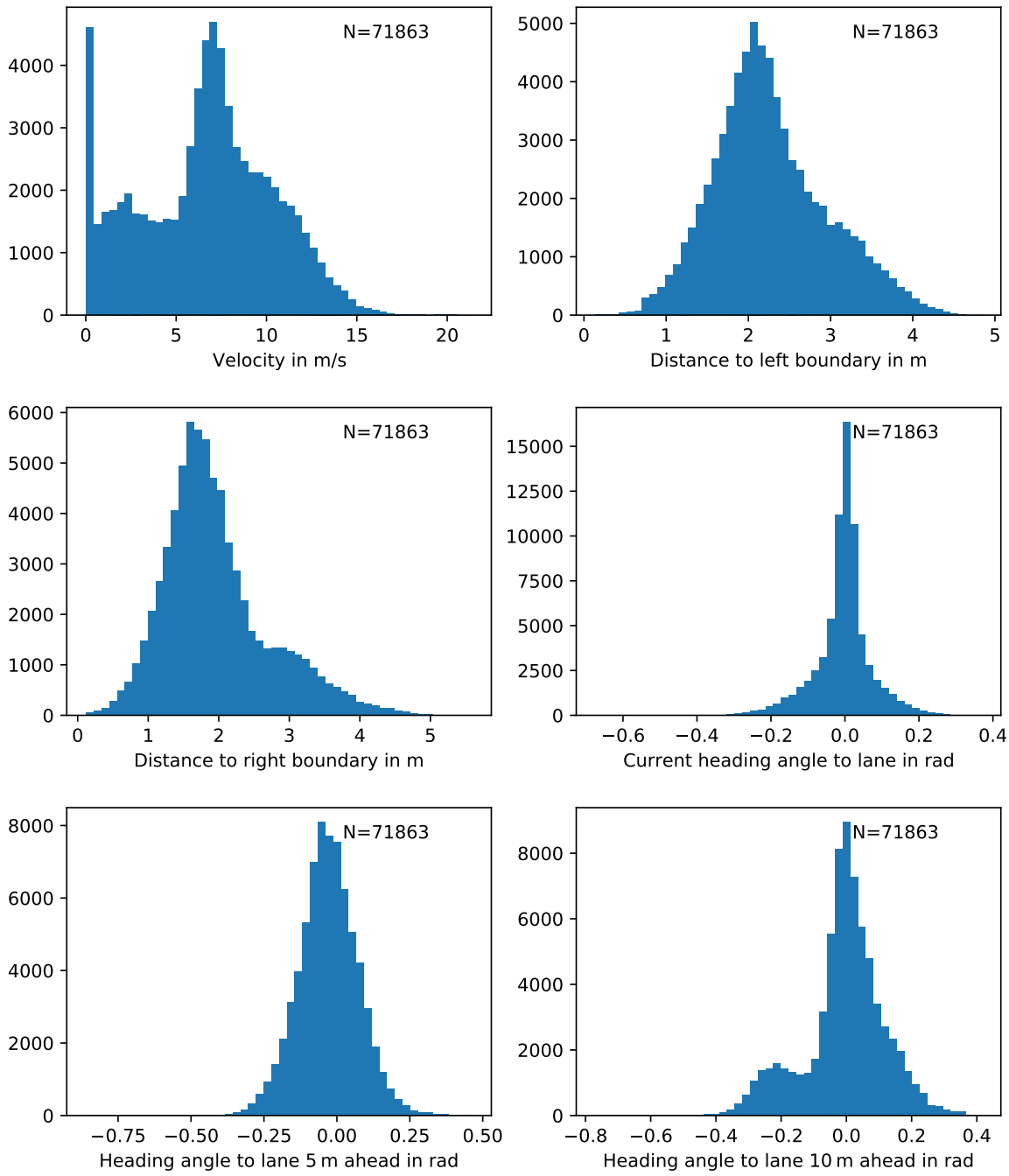


Figure A.4: Histograms of observation features in the training dataset

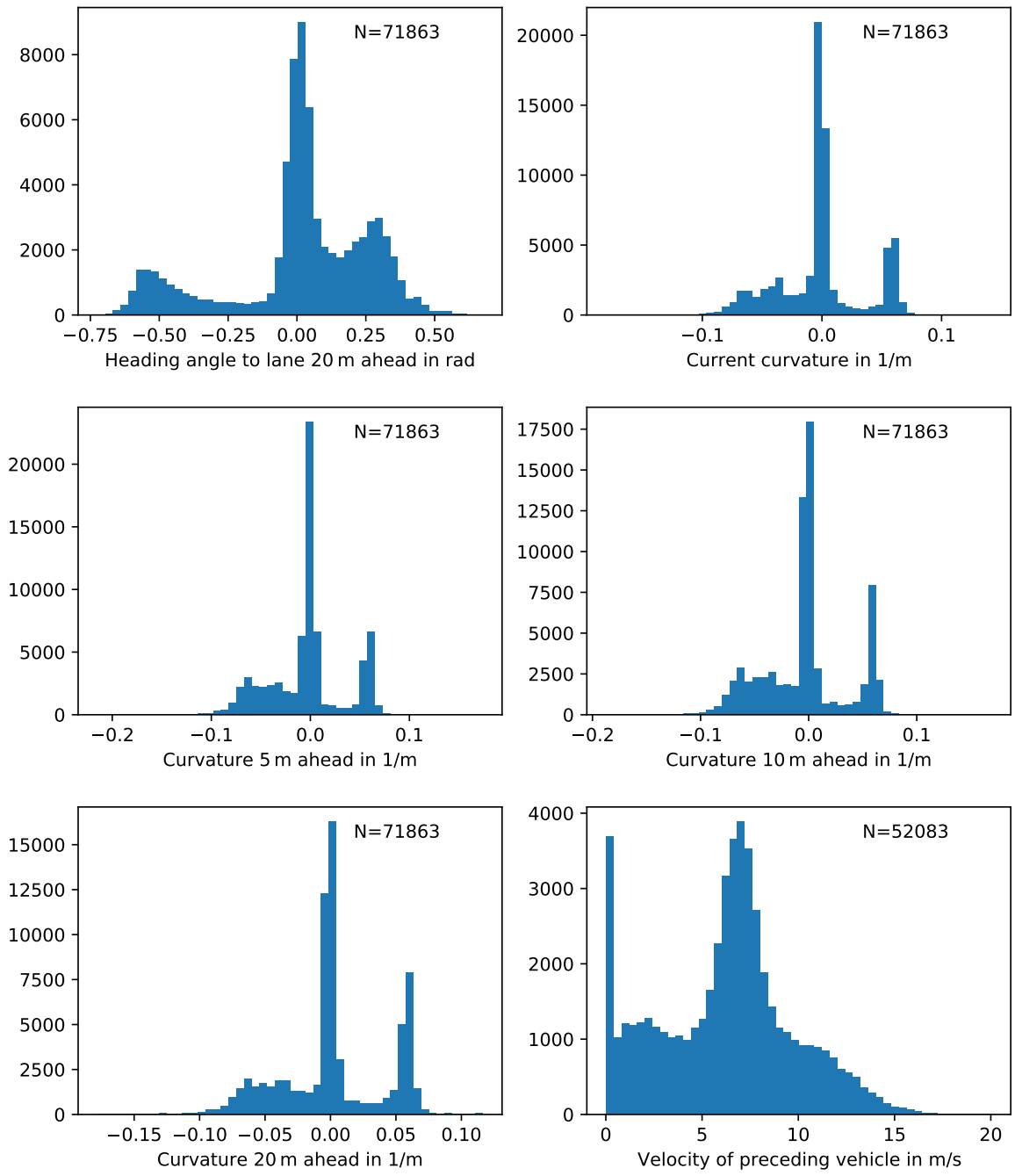


Figure A.5: Histograms of observation features in the training dataset

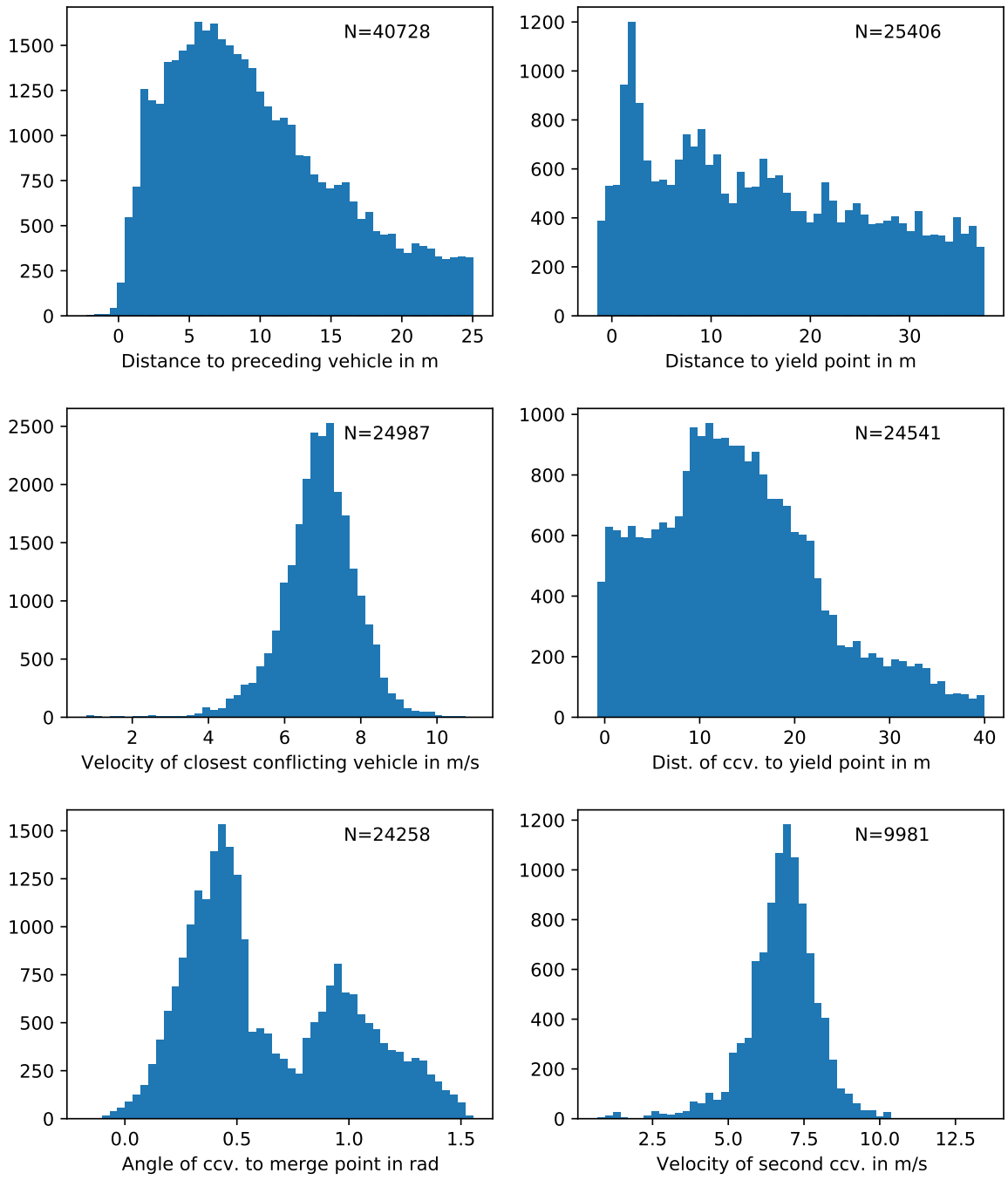


Figure A.6: Histograms of observation features in the training dataset

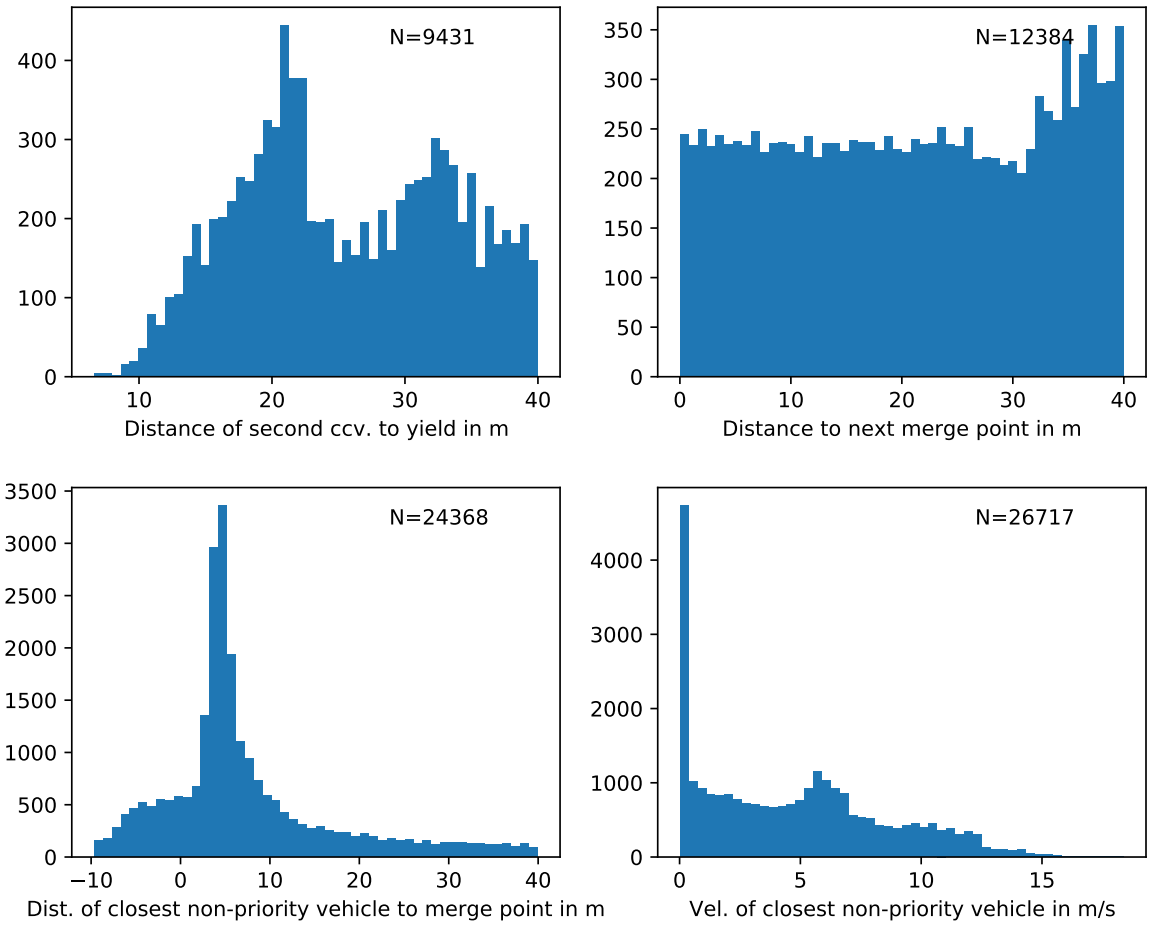


Figure A.7: Histograms of observation features in the training dataset

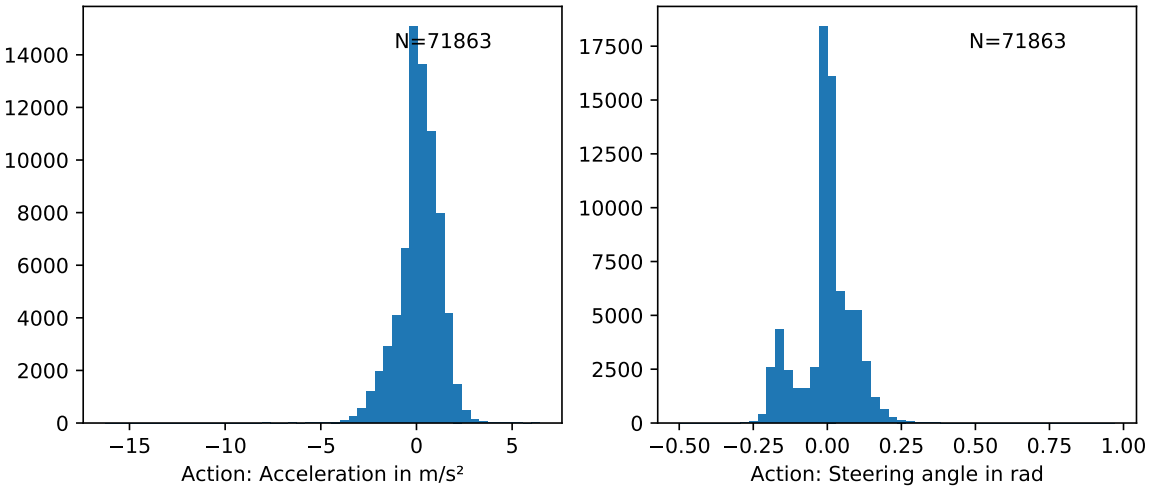


Figure A.8: Histograms of the actions in the training dataset

B Mathematical Supplements

B.1 Mean Error, Standard Deviation and RMSE

Publications in the field of prediction often specify errors as RMSE values or via their empirical mean and standard deviation. To facilitate the comparison, the relation between both representations is laid out here. Let

$$e = x - \hat{x} \tag{B.1}$$

be the error, i.e., the difference between a predicted value $\hat{x}$ and the true value x.

The mean error of N measurements is defined as

$$\mu_e = \frac{\sum_{k=1}^{N} e_k}{N}. \tag{B.2}$$

The empirical standard deviation can be computed according to

$$\sigma_e = \sqrt{\frac{\sum_{k=1}^{N} (e_k - \mu_e)^2}{N}}. \tag{B.3}$$

The RMSE $\mathcal{R}$ is defined as

$$\mathcal{R} = \sqrt{\frac{\sum_{k=1}^{N} e_k^2}{N}}. \tag{B.4}$$

The empirical standard deviation is a lower limit of the RMSE, because

$$\begin{aligned} \sigma_e^2 &= (\sum_{k=1}^{N} (e_k - \mu_e)^2)/N & \text{(B.5)} \\ &= (\sum_{k=1}^{N} e_k^2 - 2e_k\mu_e + \mu_e^2)/N & \text{(B.6)} \\ &= \mathcal{R}^2 - \mu_e^2. & \text{(B.7)} \end{aligned}$$

Thus, the RMSE can be calculated from the mean and standard deviation according to

$$\mathcal{R} = \sqrt{\mu_e^2 + \sigma_e^2}, \tag{B.8}$$

and it is equal to the standard deviation if the mean error μ_e is 0.

B.2 The Squashed Gaussian Distribution

A random variable $\mathbf{Y}$ that is distributed according to the squashed Gaussian distribution is obtained by sampling from a Gaussian distributed random variable $\mathbf{X}$, and then applying the $\tanh$ function.

$$\mathbf{X} \sim \mathcal{N}(\mu, \sigma)$$
$$\mathbf{Y} = \tanh(\mathbf{X})$$

In many RL implementations [Lia+18; Raf+21] with continuous action spaces, this distribution is employed to ensure that the realizations of a continuous action $\mathbf{Y}$ are bounded to $[-1, 1]$. To cover the designated action boundaries, the distribution must then be shifted and scaled accordingly.

As $\tanh$ is a strictly monotonic and the inverse $\tanh^{-1}$ is defined on $(-1, 1)$, the probability distribution function can be computed according to [BT08, p. 207]

$$f_{\mathbf{Y}}(y) = f_{\mathbf{X}}(\tanh^{-1}(y)) \left| \frac{\mathrm{d} \tanh^{-1}(y)}{\mathrm{d}y} \right|. \tag{B.9}$$

With this, the probability distribution function of $\mathbf{Y}$ is

$$f_{\mathbf{Y}}(y) = \begin{cases} \frac{1}{\sqrt{2\pi}\sigma(1-y^2)} e^{-\frac{(\tanh^{-1}(y)-\mu)^2}{2\sigma^2}} & \text{if } -1 < y < 1, \\ 0 & \text{otherwise.} \end{cases} \tag{B.10}$$

The resulting distribution function is shown in Figure B.1 for different parameter values. For $\sigma \gg 1$, most of the probability mass accumulates around $+1$ and -1. For $\sigma \approx 1$, the distribution is approximately uniform on $(-1, 1)$. For $\sigma \ll 1$, the probability mass accumulates close to $\tanh(\mu)$.

B.3 Maximum Entropy Distribution

Jaynes [Jay57] shows how the principle of maximum entropy can be used to derive the probability of a system being in a specific state, depending on the energy of the state. The derivations from [Jay57] are slightly adapted in the following for deriving the probability of being in a state, depending on the reward of being in that state. This derivation forms the backbone of maximum entropy IRL [Zie+08] and works that build upon it, such as AIRL [FLL18].

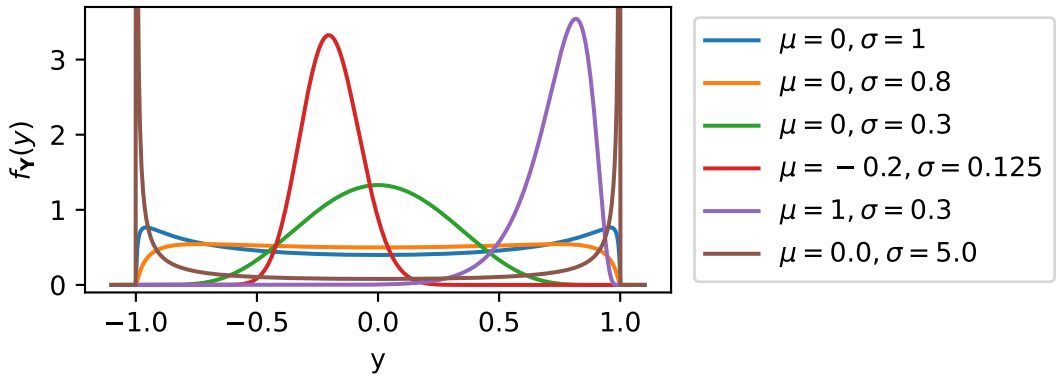


Figure B.1: The squashed Gaussian distribution for different values of μ and σ

Let a system that can assume N discrete states $(x_1, x_2, \ldots, x_N)$ with unknown probabilities $p_i = p(\mathbf{x} = x_i)$ have an expected reward of

$$\mathbf{E}\{\mathcal{R}(\mathbf{x})\} = \sum_{i=1}^{N} p_i \mathcal{R}(x_i). \tag{B.11}$$

Given no further information, what is the distribution of $p = (p_1, p_2, \ldots, p_N)$? There are infinitely many solutions to this question, as this amounts to an under-determined system of equations. For example, if there is one state x_j with $\mathcal{R}(x_j) = \mathbf{E}\{\mathcal{R}(x)\}$, a solution would be $p_j = 1$ and $p_{i \neq j} = 0$. However, this would rule out the possibility of the system being in any other state than x_j. This is a determination that cannot be inferred from the fact (B.11).

The principle of maximum entropy states that among all possible distributions of p that are compatible with the prior knowledge (B.11), the distribution which expresses the most uncertainty is the one that most accurately models the knowledge about the system. The measure of uncertainty of a distribution is the entropy

$$H(\mathbf{x}) = -\sum_{i=1}^{N} p_i \ln(p_i). \tag{B.12}$$

Problem Formulation With this, a constrained optimization problem

$$\max_{p=(p_1,p_2,\ldots)} \quad H(\mathbf{x}) \tag{B.13}$$

$$\text{subject to} \quad g(p) = \left(\sum_{i=1}^{N} p_i\right) - 1 = 0 \tag{B.14}$$

$$\text{and} \quad h(p) = \left(\sum_{i=1}^{N} p_i \mathcal{R}(x_i)\right) - \mathbf{E}\{\mathcal{R}(x)\} = 0 \tag{B.15}$$

can be formulated.

Solution To solve the problem, the Lagrange multipliers λ and μ are used.

$$L(p, \lambda, \mu) = H(p) - \lambda g(p) - \mu h(p) \tag{B.16}$$

For the maximum, it must hold that $\nabla_{(p_1,p_2,\ldots,p_N,\lambda,\mu)} L = 0$. From the partial derivative

$$\frac{\partial L}{\partial p_i} = -\ln(p_i) - 1 - \lambda - \mu \mathcal{R}(x_i) \tag{B.17}$$

it follows, that p must be distributed according to

$$p_i = e^{-\lambda - \mu \mathcal{R}(x_i) - 1}. \tag{B.18}$$

Because the sum of all probabilities adds to 1 (B.14), it is clear, that $e^{-\lambda-1}$ is a normalization constant, such that p_i can also be written as

(B.19)

$$p_i = \frac{e^{-\mu \mathcal{R}(x_i)}}{\sum_{k=1}^{N} e^{-\mu \mathcal{R}(x_k)}}. \tag{B.20}$$

The remaining constant μ can be determined by inserting (B.20) into (B.15), which is not explicitly solved here for brevity.

The result (B.20) of the maximum-entropy solution shows that the system assumes the state x_i with a probability proportional to its exponential reward. States with high rewards are exponentially more likely than states with low rewards.

Maximum Conditional Entropy Next, the same problem is solved for a conditional random variable $\mathbf{x}|\mathbf{y}$. The system again can assume N discrete states $(x_1, x_2, \ldots, x_N)$ with unknown conditional probabilities $p_y = (p(x_1|y), p(x_2|y), \ldots, p(x_N|y))$. The reward for each state depends on both, the state and the conditional variable. What is the distribution of p_y that maximizes the conditional entropy

$$H(\mathbf{x}|\mathbf{y} = y) = -\sum_{i=1}^{N} p(x_i|y) \ln p(x_i|y) \tag{B.21}$$

subject to a fixed expected value of the reward

$$\mathbf{E}_{\mathbf{x}|\mathbf{y}=y} \{\mathcal{R}(\mathbf{x}, \mathbf{y})\} = \sum_{i=1}^{N} p(x_i|y) \mathcal{R}(x, y)? \tag{B.22}$$

The problem can be solved in the same way as above using Lagrange multipliers, leading to the solution

$$p(x_i|y) = e^{-\lambda-1-\mu\mathcal{R}(x_i,y)} = e^{-\mu\mathcal{R}(x_i,y)}/Z(y). \tag{B.23}$$

In contrast to the non-conditional case, the normalizing term

$$Z(y) = e^{\lambda+1} = \sum_{i=1}^{N} e^{-\mu\mathcal{R}(x_i,y)} \tag{B.24}$$

now depends on the conditional variable.

Continuous Random Variables Applied to continuous random variables with density $p(x)$, the maximum entropy method yields similar results, as for example shown in [CT06, Ch. 12]. Then, the density assumes the form

$$p(x) = e^{-\lambda-\mu\mathcal{R}(x)-1} = e^{-\mu\mathcal{R}(x)}/Z(\mu) \tag{B.25}$$

with the normalizing factor

$$Z(\mu) = e^{\lambda+1} = \int_{x=-\infty}^{\infty} e^{-\mu\mathcal{R}(x)}dx. \tag{B.26}$$

C Training Details

C.1 Kinematic Model Parameters

To select reasonable parameters for the kinematic model (see Section 3.1), data from an 2019 Audi A6 AWD is used. The total length is $4\,951\,\mathrm{mm}$, the width is $2\,110\,\mathrm{mm}$, and the wheelbase $l_r + l_f \approx 2\,925\,\mathrm{mm}$ [AG22, p. 136]. According to NHTSA measurements [JZH20], the longitudinal center of gravity is $l_f \approx 1\,336\,\mathrm{mm}$ behind the front axle. Thus, $l_r \approx 1\,589\,\mathrm{mm}$. The turning circle is $12.4\,\mathrm{m}$, thus the minimum turning radius is $r_{\text{min}} = 6.2\,\mathrm{m}$ [ADA22]. Using the kinematic bicycle model from Figure 3.2, the turning radius is used to determine the maximum steering angle δ. From the drawing, it is clear that

$$\sin\beta = \frac{l_r}{r} \tag{C.1}$$

Moreover, according to (3.4),

$$\beta = \arctan\left(\frac{l_r}{l_f + l_r}\tan(\delta)\right). \tag{C.2}$$

With this, the steering angle

$$\delta = \arctan\left(\tan(\beta)\frac{l_f + l_r}{l_r}\right) \tag{C.3}$$

$$= \arctan\left(\frac{l_r + l_f}{\sqrt{r^2 - l_r^2}}\right) \tag{C.4}$$

corresponding to the current driving radius r can be computed.[1] Thus, $\delta_{\text{max}} \approx 26^\circ \approx \pi/7\text{rad}$.

C.2 Behavioral Cloning

The training parameters for single-step training are listed in Table C.1 and the multi-step parameters are displayed in Table C.2.

[1] The relation $\tan(\arcsin(x)) = \frac{x}{\sqrt{(1-x^2)}}$ is used [Bro+05, Ch. 2.8.3].

Table C.1: Single-step BC training parameters

Parameter	Value
Policy network shape	(22, 50, 50, 2[§])
Policy network activation	tanh
Optimizer	Adam [KB15]
Adam: Learning rate	$1 \cdot 10^{-3}$
Adam: (β_1, β_2)	$(0.9, 0.999)$
Gradient steps	100,000
Number of training samples	60,169
Training duration	$\sim 5\,\mathrm{min}$

[§]The network architecture is depicted in Figure C.1.

Table C.2: Multi-Step BC training parameters

Parameter	Value
Policy network shape	(22, 50, 50, 2[§])
Policy network activation	tanh
Optimizer	Adam [KB15]
Adam: Learning rate	$1 \cdot 10^{-3}$
Adam: (β_1, β_2)	$(0.9, 0.999)$
Gradient steps	500
Simulation Length	5, 10, 20, 40, and 80 steps
Simulation step length Δt	$0.2\,\mathrm{s}$
Number of training trajectories	11279, 6046, 3039, 1499, 695[*]
Training duration	$\sim 1\,\mathrm{h}, 2\,\mathrm{h}, 3\,\mathrm{h}, 4\,\mathrm{h}, 5\,\mathrm{h}^{\dagger}$

[§]The network architecture is depicted in Figure C.1.
[*]As the training trajectories are sliced into non-overlapping pieces, the number of training trajectories approximately halves when the number of simulation steps is doubled. See the description in Section 4.3.2.
[†]Each model is approximately trained for 1 hour. However, the trained 5-step model is the basis of the 10-step model, and that is the basis of the 20-step model, and so on. Hence, the effective training time of the models with more simulation steps is larger.

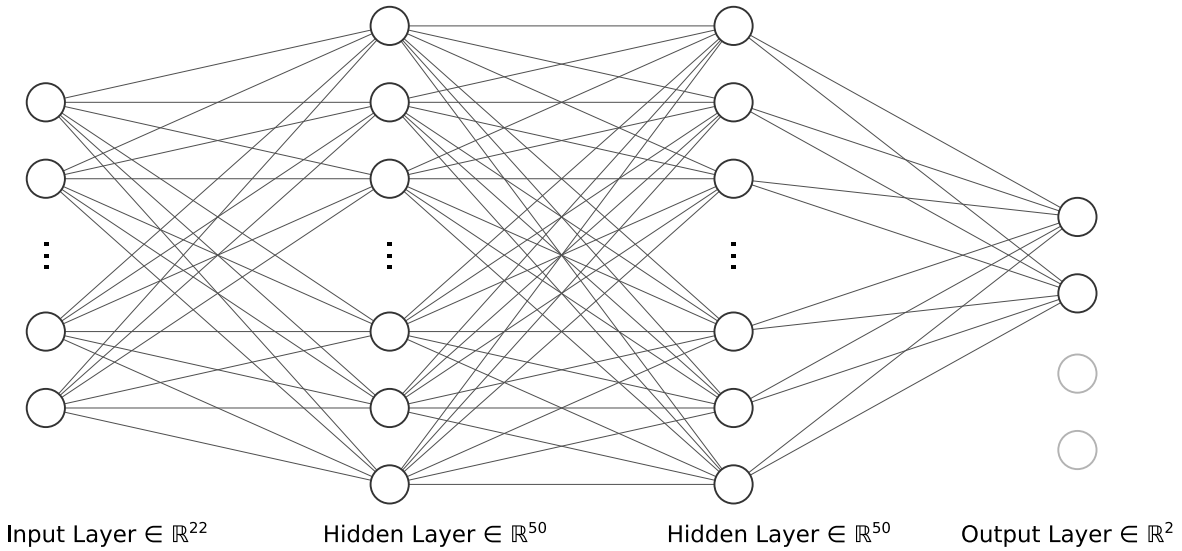


Figure C.1: Architecture of the multilayer perceptron used for learning the policy. The number of inputs is typically 22, but varies according to the number of elements of the feature vector. Two hidden layers with 50 neurons each follow. After each hidden layer, a $\tanh$ nonlinearity is used (not shown). The number of outputs is 2, representing the steering angle and the acceleration. To ensure that they are bounded, the outputs are also passed through a $\tanh$ function and then scaled and shifted before being interpreted the physical values in SI units. For RL, two additional outputs that are independent of the inputs represent the standard deviation of the two actions. These outputs are only used during the training phase. The same network architecture is also used for training the RL value estimate and the GAIL and AIRL discriminator, the only difference being that the network has just one output in these cases. Diagram generated with [LeN19].

C.3 Reinforcement Learning

The training parameters for single-agent Reinforcement Learning are listed in Table C.3, and the multi-agent training parameters are listed in Table C.4. To illustrate the situations that the MARL training is performed in, Figure C.2 displays some of the randomly initialized training situations.

Table C.3: Single-Agent RL training parameters

Parameter	**Value**
Policy network shape	(11, 50, 50, 2+2[§])
Value network shape	(11, 50, 50, 1)
Policy, value net. activation	tanh
Discount factor γ	0.99
Generalized Advantage Estimate λ [Sch+18a]	0.95
Optimizer*	Adam [KB15]
Adam: Learning rate	$3 \cdot 10^{-4}$
Adam: (β_1, β_2)	$(0.9, 0.999)$
PPO iterations over full batch	20 per epoch
PPO clip range	0.2
PPO minibatch size	1024
PPO minimum action noise σ_{min}	e^{-2}
PPO epochs	1000
Simulation length	200 steps
Step length Δt	$0.2\,\text{s}$
Simulated agents per epoch	50
Training duration	$\sim 1\,\text{h}$
Experience samples per epoch	up to 10,000[†]

[§]The two mean actions are the outputs of the fully connected policy network. The two standard deviations are independent of the input.
*Two optimizers are used, one for the policy network and one for the value network. Both use the same parameters.
[†]This is the product of simulation length and simulated agents per epoch. Fewer experiences are collected, when agents terminate early due to leaving the track, which often happens during early training.

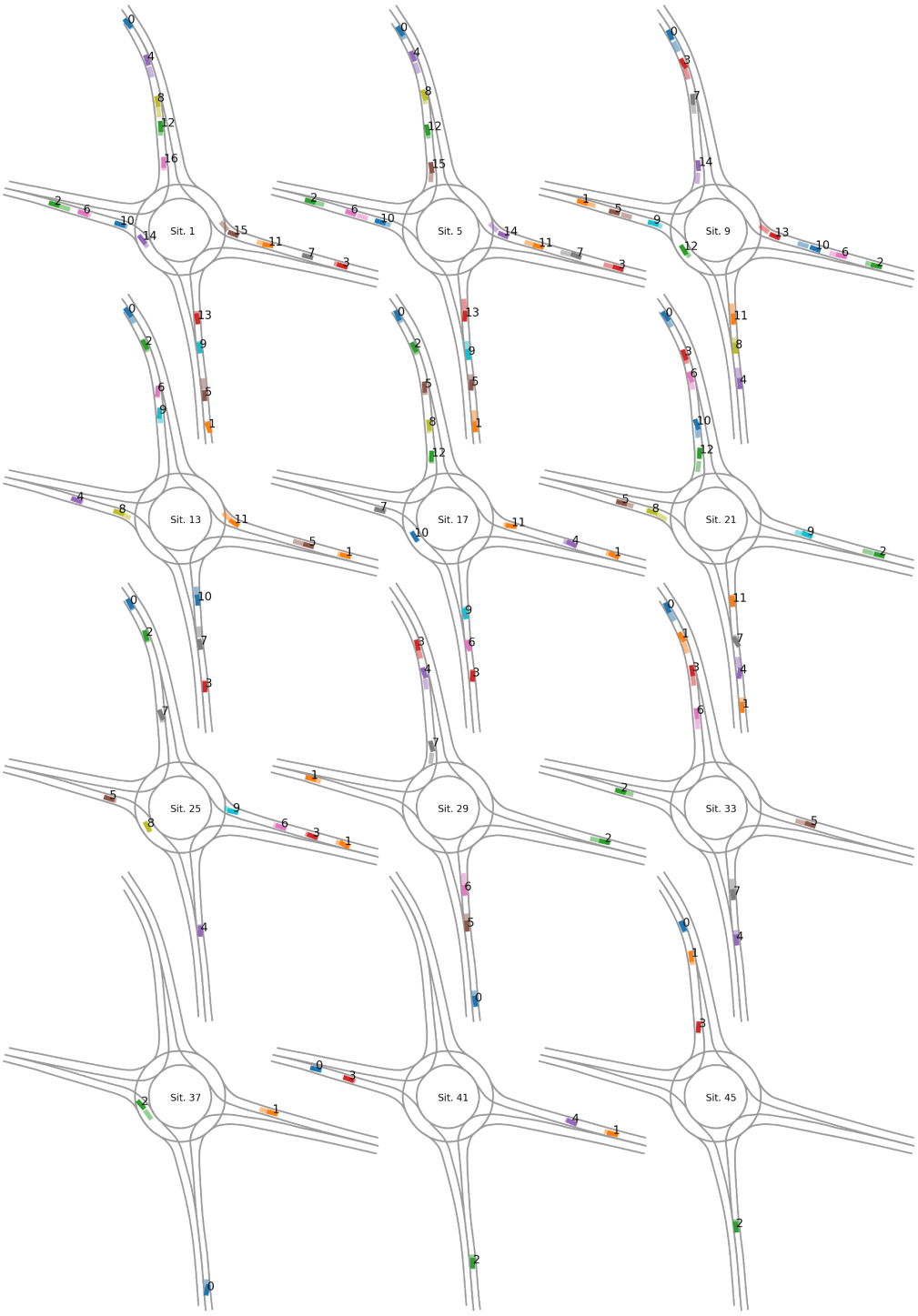


Figure C.2: 12 of the 50 situations that are initialized randomly at every training epoch. The traffic density decreases for higher situation numbers. The route intention, lane relative heading and lateral offset of each agent is initialized randomly. The speed is initialized within $\mathcal{U}(0, 20)\mathrm{m/s}$ in some situations, and within $\mathcal{U}(0, 3)\mathrm{m/s}$ in others. To visualize the different initial velocities, the position of each agent after $0.5\,\mathrm{s}$ of simulation is plotted transparent.

Table C.4: Multi-Agent RL training parameters

Parameter	Value
Policy network shape	(22[§], 50, 50, 2+2[*])
Value network shape	(22[§], 50, 50, 1)
Policy, value net. activation	tanh
Discount factor γ	0.99
Generalized Advantage Estimate λ [Sch+18a]	0.95
Optimizer	same as single-agent[*]
PPO settings	same as single-agent[*]
Simulation settings	same as single-agent[*]
Simulated agents per epoch	approximately 500
Training duration	3 to 4 h
Experience samples per epoch	up to 100,000[*]

[§]The networks with additional preference input have 25 inputs.
[*]See Table C.3.

C.4 AIRL, GAIL

The training parameters of GAIL and AIRL are listed in Table C.5.

Table C.5: GAIL and AIRL training parameters

Parameter	Value
Policy network shape	(22, 50, 50, 2+2)
Value network shape	(22, 50, 50, 1)
Discriminator network shape	(8-25[§], 50, 50, 1)
Policy, value, discriminator net. activation	tanh
Discount factor γ	0.99
Generalized Advantage Estimate λ [Sch+18a]	0.95
Optimizer*	Adam [KB15]
Adam: Policy and Value network learning rate	$1 \cdot 10^{-3}$
Adam: Discriminator learning rate	$3 \cdot 10^{-4}$
Adam: (β_1, β_2)	$(0.9, 0.999)$
PPO iterations over full batch	20 per epoch
PPO clip range	0.2
PPO minibatch size	2048
PPO minimum action noise σ_{min}	e^{-2}
Discriminator: updates per epoch over full batch	1
Discriminator: minibatch size	2048
Discriminator: number of historic experiences	20[†]
Normalized discriminator noise	0.2[¶] or 0
Training epochs	200
Simulation length	50 steps
Step length Δt	$0.2\,\text{s}$
Simulated agents per epoch	1007
Min./Mean/Max. number of vehicles per situation	4/12.1/24
Experience samples per epoch	around 50,000[‡]
Training duration	$\sim 45\,\text{min}$
AIRL only: additional agents in fictional situations per epoch	~ 1250

[§]The input size depends on the number of features used by the discriminator. This varies, depending on whether the full or restricted feature set is used and depending on whether the last steering angle is an additional input. See Table 6.1 for details.

*Three optimizers are used, one for the policy network, one for the value network, and one for the discriminator. In all cases, the Adam algorithm is used.

[†]The discriminator is not only trained on the most recent policy execution, but additionally on data from 20 random earlier policy executions to avoid overfitting to the current policy.

[¶]The discriminator inputs are standardized such that the mean is 0 and the standard deviation is 1. When discriminator noise is enabled, additional Gaussian noise with $\mu = 0, \sigma = 0.2$ is added to each feature value independently.

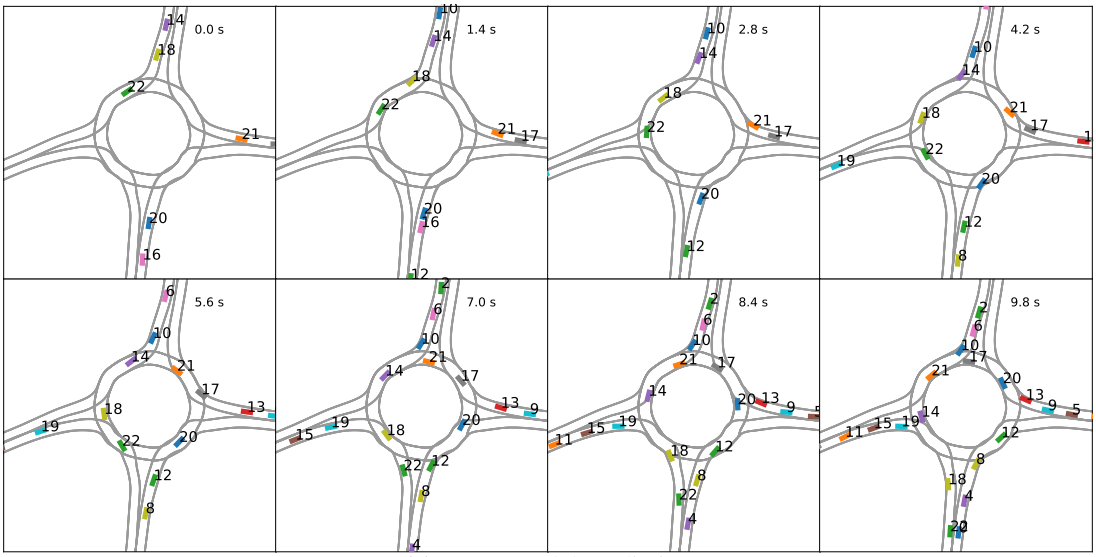
[‡]This is the product of simulation length and simulated agents per epoch. Fewer experiences are collected, when agents terminate early due to leaving the track, which often happens during early training. See also p.148.

D Additional Model Executions

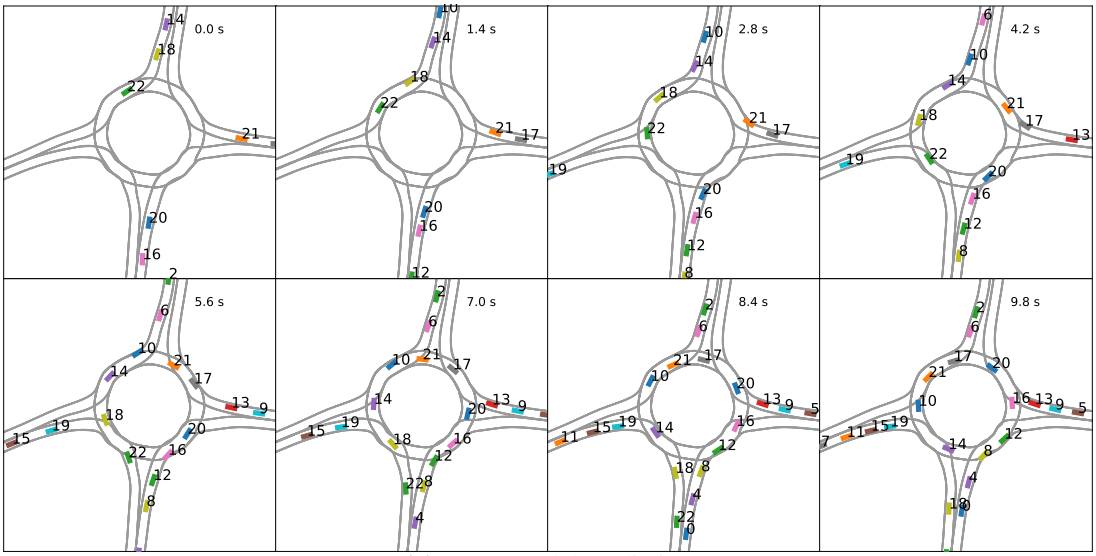
This chapter shows the final models evaluated in Chapter 7 in additional situations for a closer visual investigation of their performance.

The situation depicted in Figures D.1 and D.2 is set up at the new roundabout from Figure 7.3 that was not used for training any of the models. Hence, this experiment probes the ability of the models to generalize to an unseen situation. A crop of the full map is shown to focus on interesting interactions at the roundabout entries. Some vehicles are initialized critically. Consider for example vehicle #16, which approaches its preceding vehicle #20 with a large velocity. Except from the AIRL policy, no model has learned to brake strong enough to avoid a collision. Apart from that, GAIL and the multi-step policies do not leave the track and collide only in one additional case (#9 for the MS8-policy). The single-step policy seems incapable of controlling the situation, as many vehicles drift off the track in Figure D.2a.

Next, the performance in the fictional and more diverse situation from Figure 6.12 is investigated. As the situation spans a large area, labels of individual vehicles are omitted. The vehicles are initialized critically with some near-collisions. Figure D.3 shows that both GAIL and AIRL are capable of issuing collision-free predictions on the four new road layouts around the roundabout. In contrast, none of the direct methods from Figure D.4 is able to control all vehicles in the new auxiliary situations. The multi-step policies leave the track less often than the single-step policy, but still fail to successfully handle the unknown situation. Also, the dense traffic with many critical initial states leads to many collisions inside the roundabout for all models from Figure D.4. In contrast, only three collisions for GAIL and one collision for AIRL can be observed inside the roundabout.

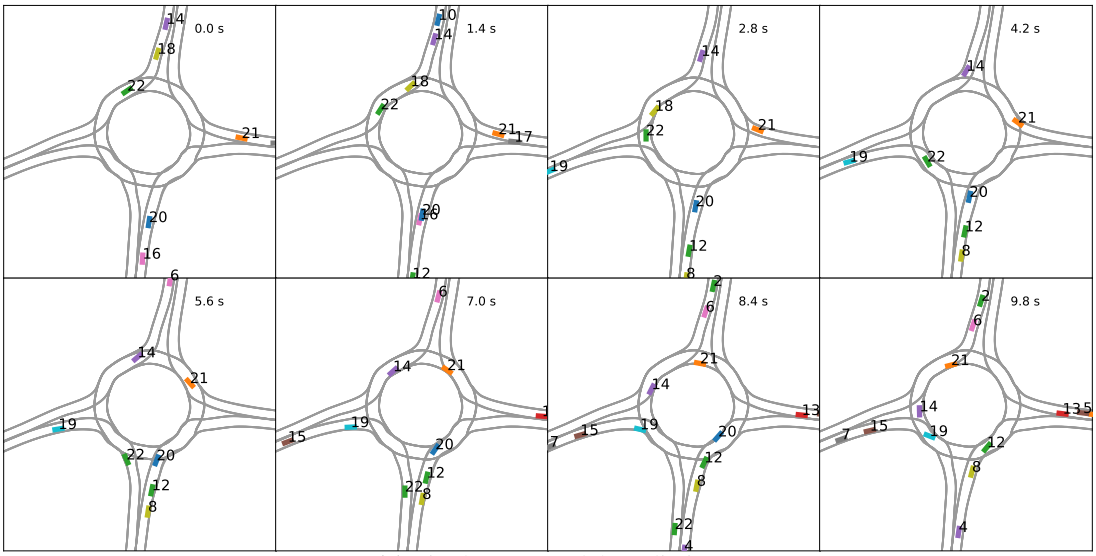


(a) GAIL model prediction

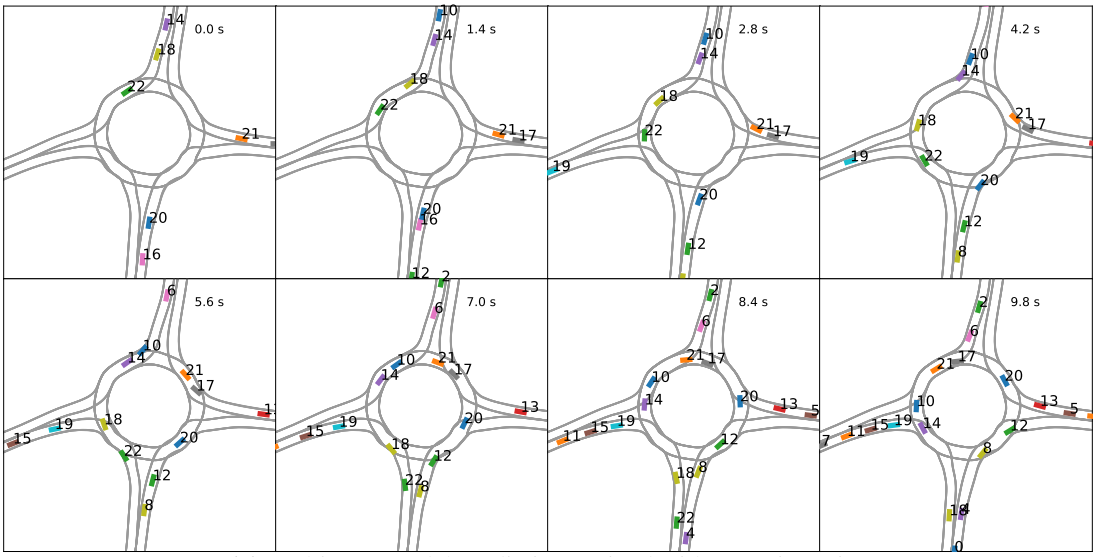


(b) AIRL model prediction

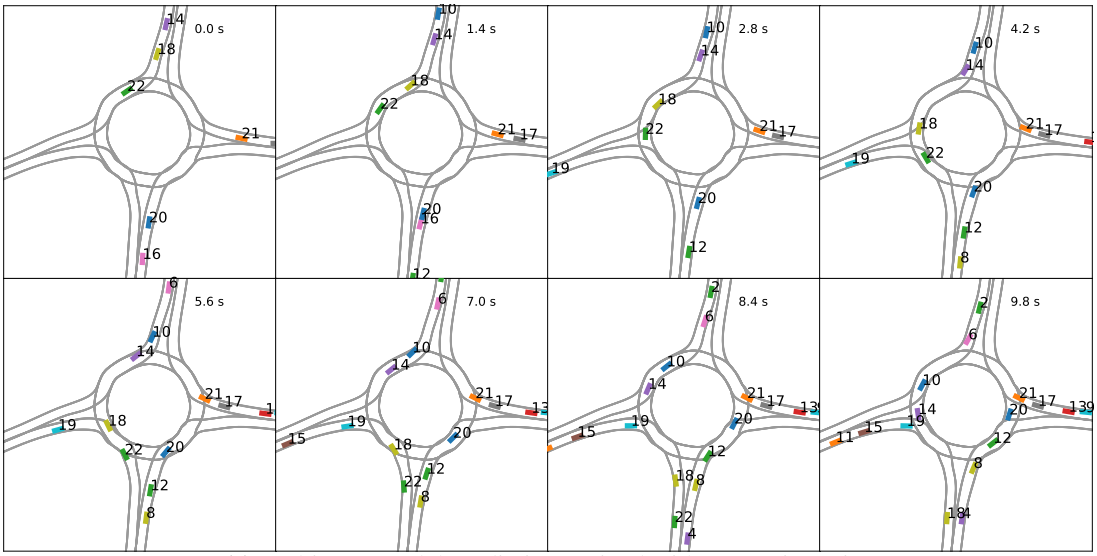
Figure D.1: Execution of the learned policies in a randomly initialized situation on the map of an untrained roundabout.



(a) Single-step model prediction



(b) Multi-step model prediction, trained with 8 s trajectories



(c) Multi-step model prediction, trained with 16 s trajectories

Figure D.2: Execution of the learned policies in a randomly initialized situation on the map of an untrained roundabout.

(a) GAIL model prediction

(b) AIRL model prediction

Figure D.3: Execution of the learned policies in a randomly initialized situation on the map of another roundabout that was used during additional AIRL training.

(a) Single-step model prediction

(b) Multi-step model prediction, trained with 8 s trajectories

Figure D.4: Execution of the learned policies in a randomly initialized situation on the map of another roundabout that was used during additional AIRL training.

(a) Multi-step model prediction, trained with 16 s trajectories

Figure D.5: Continuation of Figure D.4

Bibliography

Own References

[Bey+21a] Henrik Bey, Moritz Sackmann, Alexander Lange, and Jörn Thielecke. "Handling Prediction Model Errors in Planning for Automated Driving Using POMDPs". In: *International Intelligent Transportation Systems Conference*. Indianapolis, USA: IEEE, 2021, pp. 439–446.

[Bey+21b] Henrik Bey, Moritz Sackmann, Alexander Lange, and Jörn Thielecke. "POMDP Planning at Roundabouts". In: *Intelligent Vehicles Symposium*. Online event: IEEE, 2021, pp. 264–271.

[Bey+20] Henrik Bey, Maximilian Tratz, Moritz Sackmann, Alexander Lange, and Jörn Thielecke. "Tutorial on Sampling-based POMDP-planning for Automated Driving". In: *International Conference on Vehicle Technology and Intelligent Transport Systems*. Online event: INSTICC, 2020, pp. 312–321.

[Hof+22] Ulrich Hofmann, Moritz Sackmann, Henrik Bey, and Tobias Leemann. "Verfahren zur Prädiktion von Fahreingriffen, Verfahren zum Training eines Algorithmus und Kraftfahrzeug". DE102020129451A1. Patent pending. May 2022.

[Kon+21] Fabian Konstantinidis, Moritz Sackmann, Oliver De Candido, Ulrich Hofmann, Jörn Thielecke, and Wolfgang Utschick. "Parameter Sharing Reinforcement Learning for Modeling Multi-Agent Driving Behavior in Roundabout Scenarios". In: *International Intelligent Transportation Systems Conference*. Indianapolis, USA: IEEE, 2021, pp. 1974–1981.

[Kon+23] Fabian Konstantinidis, Moritz Sackmann, Ulrich Hofmann, and Christoph Stiller. "Modeling Interaction-Aware Driving Behavior using Graph-Based Representations and Multi-Agent Reinforcement Learning". In: *International Conference on Intelligent Transportation Systems*. Bilbao, Spain: IEEE, 2023.

[Lee+21] Tobias Leemann, Moritz Sackmann, Jörn Thielecke, and Ulrich Hofmann. "Distribution Preserving Multiple Hypotheses Prediction for Uncertainty Modeling". In: *European Symposium on Artificial Neural Networks*. Online event (Bruges, Belgium), 2021, pp. 523–528.

[Rad+23] Henrik Radtke, Henrik Bey, Moritz Sackmann, and Torsten Schön. "Predicting Driver Behavior on the Highway With Multi-Agent Adversarial Inverse Reinforcement Learning". In: *Intelligent Vehicles Symposium*. Anchorage, Alaska: IEEE, 2023.

[Sac+20a] Moritz Sackmann, Henrik Bey, Ulrich Hofmann, and Jörn Thielecke. “Classification of Driver Intentions at Roundabouts”. In: *International Conference on Vehicle Technology and Intelligent Transport Systems*. Online event: INSTICC, 2020, pp. 301–311.

[Sac+20b] Moritz Sackmann, Henrik Bey, Ulrich Hofmann, and Jörn Thielecke. “Prediction Error Reduction of Neural Networks for Car-Following Using Multi-Step Training”. In: *International Intelligent Transportation Systems Conference*. Online event: IEEE, 2020.

[Sac+22a] Moritz Sackmann, Henrik Bey, Ulrich Hofmann, and Jörn Thielecke. “Learning a Diverse and Cooperative Policy for Predicting Roundabout Traffic Situations”. In: *14. Uni-DAS Workshop Fahrerassistenz und automatisiertes Fahren*. Berkheim, Germany, 2022, pp. 1–10.

[Sac+22b] Moritz Sackmann, Henrik Bey, Ulrich Hofmann, and Jörn Thielecke. “Modeling Driver Behavior using Adversarial Inverse Reinforcement Learning”. In: *Intelligent Vehicles Symposium*. Aachen, Germany: IEEE, 2022, pp. 1683–1690.

[Sac+21] Moritz Sackmann, Tobias Leemann, Henrik Bey, Ulrich Hofmann, and Jörn Thielecke. “Multi-Step Training for Predicting Roundabout Traffic Situations”. In: *International Intelligent Transportation Systems Conference*. Indianapolis, USA: IEEE, 2021.

[Vog+20] Carina Vogl, Moritz Sackmann, Ludwig Kürzinger, and Ulrich Hofmann. “Frenet Coordinate Based Driving Maneuver Prediction at Roundabouts Using LSTM Networks”. In: *Computer Science in Cars Symposium*. Online event: ACM, 2020.

References

[AN04] Pieter Abbeel and Andrew Y Ng. “Apprenticeship Learning via Inverse Reinforcement Learning”. In: *International Conference on Machine Learning*. Banff, Canada, 2004.

[Ala+16] Alexandre Alahi, Kratarth Goel, Vignesh Ramanathan, Alexandre Robicquet, Li Fei-Fei, and Silvio Savarese. “Social LSTM: Human Trajectory Prediction in Crowded Spaces”. In: *Conference on Computer Vision and Pattern Recognition*. Las Vegas, USA: IEEE, 2016, pp. 961–971.

[AY22] Berat Mert Albaba and Yıldıray Yıldız. “Driver Modeling Through Deep Reinforcement Learning and Behavioral Game Theory”. In: *IEEE Transactions on Control Systems Technology* 30.2 (2022), pp. 885–892.

[Alt10] Matthias Althoff. “Reachability Analysis and its Application to the Safety Assessment of Autonomous Cars”. PhD thesis. Technische Universität München, 2010.

[AM11] Matthias Althoff and Alexander Mergel. “Comparison of Markov Chain Abstraction and Monte Carlo Simulation for the Safety Assessment of Autonomous Cars”. In: *IEEE Transactions on Intelligent Transportation Systems* 12.4 (2011), pp. 1237–1247.

[AN09] Samer Ammoun and Fawzi Nashashibi. “Real time trajectory prediction for collision risk estimation between vehicles”. In: *International Conference on Intelligent Computer Communication and Processing*. IEEE, 2009, pp. 417–422.

[And+21] Marcin Andrychowicz et al. “What Matters in On-Policy Reinforcement Learning? A Large-Scale Empirical Study”. In: *International Conference on Learning Representations*. arXiv: 2006.05990. Online event (Vienna, Austria), 2021.

[Ape+15] Jiří Apeltauer, Adam Babinec, David Herman, and Tomáš Apeltauer. “Automatic Vehicle Trajectory Extraction for Traffic Analysis from Aerial Video Data”. In: *The Int. Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences* XL-3/W2 (2015), pp. 9–15.

[Arg+09] Brenna D. Argall, Sonia Chernova, Manuela Veloso, and Brett Browning. “A survey of robot learning from demonstration”. In: *Robotics and Autonomous Systems* 57.5 (2009), pp. 469–483.

[AB17] Martin Arjovsky and Léon Bottou. “Towards Principled Methods for Training Generative Adversarial Networks”. In: *International Conference on Learning Representations*. Toulon, France, 2017.

[ACB17] Martin Arjovsky, Soumith Chintala, and Léon Bottou. “Wasserstein Generative Adversarial Networks”. In: *Proceedings of Machine Learning Research* 70 (2017). arXiv: 1701.07875, pp. 214–223.

[BKS21] Moritz Bächer, Espen Knoop, and Christian Schumacher. “Design and Control of Soft Robots Using Differentiable Simulation”. In: *Current Robotics Reports* 2.2 (2021), pp. 211–221.

[Bag15] J. Andrew Bagnell. *An Invitation to Imitation*. Tech. rep. Robotics Institute, Carnegie Mellon University, 2015.

[BKO19] Mayank Bansal, Alex Krizhevsky, and Abhijit Ogale. “ChauffeurNet: Learning to Drive by Imitating the Best and Synthesizing the Worst”. In: *Robotics: Science and Systems*. Freiburg, Germany, 2019.

[Bar+19] Emmanouil N. Barmpounakis, Eleni I. Vlahogianni, John C. Golias, and Adam Babinec. “How accurate are small drones for measuring microscopic traffic parameters?” In: *Transportation Letters* 11.6 (2019), pp. 332–340.

[Ber+21] Luca Bergamini et al. “SimNet: Learning Reactive Self-driving Simulations from Real-world Observations”. In: *International Conference on Robotics and Automation*. Xi’An, China: IEEE, 2021, pp. 5119–5125.

[BT08] Dimitri P. Bertsekas and John N. Tsitsiklis. *Introduction to probability*. 2nd ed. Optimization and computation series. Belmont: Athena scientific, 2008.

[Bey+19] Henrik Bey, Frank Dierkes, Sebastian Bayerl, Alexander Lange, Dennis Fassbender, and Jörn Thielecke. “Optimization-based Tactical Behavior Planning for Autonomous Freeway Driving in Favor of the Traffic Flow”. In: *Intelligent Vehicles Symposium*. Paris, France: IEEE, 2019, pp. 1033–1040.

[Bha+23] Raunak Bhattacharyya, Blake Wulfe, Derek Phillips, Alex Kuefler, Jeremy Morton, Ransalu Senanayake, and Mykel Kochenderfer. “Modeling Human Driving Behavior through Generative Adversarial Imitation Learning”. In: *IEEE Transactions on Intelligent Transportation Systems* 24.3 (2023). arXiv: 2006.06412, pp. 2874–2887.

[Bha+19] Raunak P. Bhattacharyya, Derek J. Phillips, Changliu Liu, Jayesh K. Gupta, Katherine Driggs-Campbell, and Mykel J. Kochenderfer. “Simulating Emergent Properties of Human Driving Behavior Using Multi-Agent Reward Augmented Imitation Learning”. In: *International Conference on Robotics and Automation*. Montreal, Canada: IEEE, 2019, pp. 789–795.

[Bha+18] Raunak P. Bhattacharyya, Derek J. Phillips, Blake Wulfe, Jeremy Morton, Alex Kuefler, and Mykel J. Kochenderfer. “Multi-Agent Imitation Learning for Driving Simulation”. In: *International Conference on Intelligent Robots and Systems*. Madrid, Spain: IEEE/RSJ, 2018.

[Bin07] Ken Binmore. *Playing for real: a text on game theory*. Oxford University Press, 2007.

[Bis06] Christopher M. Bishop. *Pattern recognition and machine learning*. 1st ed. New York: Springer, 2006.

[Boj+16] Mariusz Bojarski et al. *End to End Learning for Self-Driving Cars*. Pre-print, arXiv: 1604.07316. 2016.

[Bou+20] Maxime Bouton, Alireza Nakhaei, David Isele, Kikuo Fujimura, and Mykel J. Kochenderfer. "Reinforcement Learning with Iterative Reasoning for Merging in Dense Traffic". In: *International Conference on Intelligent Transportation Systems*. Online event (Rhodes, Greece): IEEE, 2020.

[Bro+05] Ilja Nikolajewitsch Bronstein, Konstantin Adolfowitsch Semendjajew, Gerhard Musiol, and Heiner Mühlig. *Taschenbuch der Mathematik*. 6th ed. Frankfurt am Main, Germany: Harri Deutsch, 2005.

[BDK20] Kyle Brown, Katherine Driggs-Campbell, and Mykel J. Kochenderfer. *A Taxonomy and Review of Algorithms for Modeling and Predicting Human Driver Behavior*. Pre-print, arXiv:2006.08832v3. Nov. 2020.

[Bur+22] Christoph Burger, Johannes Fischer, Frank Bieder, Ömer Şahin Taş, and Christoph Stiller. "Interaction-Aware Game-Theoretic Motion Planning for Automated Vehicles using Bi-level Optimization". In: *International Conference on Intelligent Transportation Systems*. Macau, China: IEEE, 2022, pp. 3978–3985.

[BBD08] Lucian Buşoniu, Robert Babuška, and Bart De Schutter. "A Comprehensive Survey of Multiagent Reinforcement Learning". In: *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 38.2 (2008), pp. 156–172.

[Cae+20] Holger Caesar et al. "nuScenes: A multimodal dataset for autonomous driving". In: *Conference on Computer Vision and Pattern Recognition*. Online event: IEEE/CVF, 2020, pp. 11621–11631.

[Cas+20a] Sergio Casas, Cole Gulino, Renjie Liao, and Raquel Urtasun. "SpAGNN: Spatially-Aware Graph Neural Networks for Relational Behavior Forecasting from Sensor Data". In: *International Conference on Robotics and Automation*. Paris, France: IEEE, 2020, pp. 9491–9497.

[Cas+20b] Sergio Casas, Cole Gulino, Simon Suo, Katie Luo, Renjie Liao, and Raquel Urtasun. "Implicit Latent Variable Model for Scene-Consistent Motion Forecasting". In: *European Conference on Computer Vision*. arXiv: 2007.12036. Online Event (Glasgow, UK): Springer, 2020, pp. 624–641.

[CLU18] Sergio Casas, Wenjie Luo, and Raquel Urtasun. "IntentNet: Learning to Predict Intention from Raw Sensor Data". In: *Proceedings of The 2nd Conference on Robot Learning, PMLR*. Vol. 87. Zürich, Switzerland, 2018, pp. 947–956.

[Cha+19] Yuning Chai, Benjamin Sapp, Mayank Bansal, and Dragomir Anguelov. "MultiPath: Multiple Probabilistic Anchor Trajectory Hypotheses for Behavior Prediction". In: *Proceedings of the Conference on Robot Learning, PMLR*. Vol. 100. Osaka, Japan, 2019, pp. 86–99.

[CT06] Thomas M. Cover and Joy A. Thomas. *Elements of Information Theory*. 2nd ed. Hoboken, New Jersey: Wiley-Interscience, 2006.

[Cui+19] Henggang Cui, Vladan Radosavljevic, Fang-Chieh Chou, Tsung-Han Lin, Thi Nguyen, Tzu-Kuo Huang, Jeff Schneider, and Nemanja Djuric. "Multimodal Trajectory Predictions for Autonomous Driving using Deep Convolutional Networks". In: *International Conference on Robotics and Automation*. Montreal, Canada: IEEE, 2019, pp. 2090–2096.

[DBU21] Oliver De Candido, Maximilian Binder, and Wolfgang Utschick. "An Interpretable Lane Change Detector Algorithm based on Deep Autoencoder Anomaly Detection". In: *Intelligent Vehicles Symposium*. Nagoya, Japan: IEEE, 2021, pp. 516–523.

[dHJL19] Pim de Haan, Dinesh Jayaraman, and Sergey Levine. "Causal Confusion in Imitation Learning". In: *Advances in Neural Information Processing Systems*. Vol. 32. Vancouver, Canada, 2019.

[DRT18] Nachiket Deo, Akshay Rangesh, and Mohan M. Trivedi. "How Would Surround Vehicles Move? A Unified Framework for Maneuver Classification and Motion Prediction". In: *IEEE Transactions on Intelligent Vehicles* 3.2 (2018), pp. 129–140.

[DT18] Nachiket Deo and Mohan M. Trivedi. "Convolutional Social Pooling for Vehicle Trajectory Prediction". In: *Conference on Computer Vision and Pattern Recognition Workshops*. Salt Lake City, USA: IEEE, 2018, pp. 1468–1476.

[dWit+20] Christian Schroeder de Witt, Tarun Gupta, Denys Makoviichuk, Viktor Makoviychuk, Philip H. S. Torr, Mingfei Sun, and Shimon Whiteson. *Is Independent Learning All You Need in the StarCraft Multi-Agent Challenge?* Pre-print, arXiv: 2011.09533v1. Nov. 2020.

[DZ87] E.D. Dickmanns and A. Zapp. "Autonomous High Speed Road Vehicle Guidance by Computer Vision". In: *IFAC Proceedings Volumes* 20.5 (1987), pp. 221–226.

[Die+19] Frederik Diehl, Thomas Brunner, Michael Truong Le, and Alois Knoll. "Graph Neural Networks for Modelling Traffic Participant Interaction". In: *Intelligent Vehicles Symposium*. Paris, France: IEEE, 2019.

[Dju+20] Nemanja Djuric, Vladan Radosavljevic, Henggang Cui, Thi Nguyen, Fang-Chieh Chou, Tsung-Han Lin, Nitin Singh, and Jeff Schneider. “Uncertainty-aware Short-term Motion Prediction of Traffic Actors for Autonomous Driving”. In: *Winter Conference on Applications of Computer Vision*. Note: Work was originally published in 2018, but revised twice with different titles. See arXiv: 1808.05819. Snowmass Village, USA: IEEE/CVF, 2020, pp. 2095–2104.

[Dju+21] Nemanja Djuric et al. “MultiXNet: Multiclass Multistage Multimodal Motion Prediction”. In: *Intelligent Vehicles Symposium*. Nagoya, Japan: IEEE, 2021, pp. 435–442.

[Dos17] Alexey Dosovitskiy. “CARLA: An Open Urban Driving Simulator”. In: *Proceedings of the 1st Annual Conference on Robot Learning, PMLR*. Vol. 78. Mountain View, USA, 2017.

[Ett+21] Scott Ettinger et al. “Large Scale Interactive Motion Forecasting for Autonomous Driving: The Waymo Open Motion Dataset”. In: *International Conference on Computer Vision*. Online event (Montreal, Canada): IEEE/CVF, 2021, pp. 9710–9719.

[Eve+16] Niclas Evestedt, Erik Ward, John Folkesson, and Daniel Axehill. “Interaction aware trajectory planning for merge scenarios in congested traffic situations”. In: *International Intelligent Transportation Systems Conference*. Rio de Janeiro, Brazil: IEEE, 2016, pp. 465–472.

[Fin+16] Chelsea Finn, Paul Christiano, Pieter Abbeel, and Sergey Levine. *A Connection between Generative Adversarial Networks, Inverse Reinforcement Learning, and Energy-Based Models*. NIPS workshop on adversarial training, arXiv: 1611.03852. Nov. 2016.

[FLA16] Chelsea Finn, Sergey Levine, and Pieter Abbeel. “Guided Cost Learning: Deep Inverse Optimal Control via Policy Optimization”. In: *Proceedings of The 33rd International Conference on Machine Learning*. Vol. 48. New York City, USA, 2016, pp. 49–58.

[Fis+19] Jaime F. Fisac, Eli Bronstein, Elis Stefansson, Dorsa Sadigh, S. Shankar Sastry, and Anca D. Dragan. “Hierarchical Game-Theoretic Planning for Autonomous Vehicles”. In: *International Conference on Robotics and Automation*. Montreal, Canada: IEEE, 2019, pp. 9590–9596.

[FLL18] Justin Fu, Katie Luo, and Sergey Levine. “Learning Robust Rewards with Adversarial Inverse Reinforcement Learning”. In: *International Conference on Learning Representations*. Vancouver, Canada, 2018.

[Gao+20] 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: *Conference on Computer Vision and Pattern Recognition*. Online event: IEEE/CVF, 2020.

[GS19] Mario Garzón and Anne Spalanzani. “Game theoretic decision making for autonomous vehicles’ merge manoeuvre in high traffic scenarios”. In: *International Intelligent Transportation Systems Conference*. Auckland, New Zealand: IEEE, 2019, pp. 3448–3453.

[GS20] Mario Garzón and Anne Spalanzani. “Game theoretic decision making based on real sensor data for autonomous vehicles’ maneuvers in high traffic”. In: *International Conference on Robotics and Automation*. Online event (Paris, France): IEEE, 2020, pp. 5378–5384.

[Gep17] Pawel Gepner. “Using AVX2 Instruction Set to Increase Performance of High Performance Computing Code”. In: *Computing and Informatics* 36.5 (2017), pp. 1001–1018.

[Gil+21] Thomas Gilles, Stefano Sabatini, Dzmitry Tsishkou, Bogdan Stanciulescu, and Fabien Moutarde. “HOME: Heatmap Output for future Motion Estimation”. In: *International Intelligent Transportation Systems Conference*. Indianapolis, USA: IEEE, 2021, pp. 500–507.

[Gip80] Peter G. Gipps. “A Behavioural Car-Following Model for Computer Simulation”. In: *Transportation Research Part B: Methodological* 15.2 (1980), pp. 105–111.

[GBC16] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. 1st ed. MIT Press, 2016. URL: `http://www.deeplearningbook.org`.

[Goo+14] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. “Generative Adversarial Nets”. In: *Neural Information Processing Systems*. Vol. 27. Montreal, Canada, 2014.

[Gre00] Marc Green. “"How Long Does It Take to Stop?" Methodological Analysis of Driver Perception-Brake Times”. In: *Transportation Human Factors* 2.3 (2000), pp. 195–216.

[Gup+18] Agrim Gupta, Justin Johnson, Li Fei-Fei, Silvio Savarese, and Alexandre Alahi. “Social GAN: Socially Acceptable Trajectories with Generative Adversarial Networks”. In: *Conference on Computer Vision and Pattern Recognition*. Salt Lake City, USA: IEEE/CVF, 2018, pp. 2255–2264.

[GEK17] Jayesh K. Gupta, Maxim Egorov, and Mykel Kochenderfer. “Cooperative Multi-agent Control Using Deep Reinforcement Learning”. In: *Autonomous Agents and Multiagent Systems*. São Paulo, Brazil, 2017, pp. 66–83.

[GBK12] Abner Guzmán-Rivera, Dhruv Batra, and Pushmeet Kohli. “Multiple Choice Learning: Learning to Produce Multiple Structured Outputs”. In: *Advances in Neural Information Processing Systems*. Lake Tahoe, USA, 2012.

[Haa+19] Tuomas Haarnoja, Sehoon Ha, Aurick Zhou, Jie Tan, George Tucker, and Sergey Levine. “Learning to Walk via Deep Reinforcement Learning”. In: *Robotics: Science and Systems*. Freiburg, Germany, 2019.

[Haa+18] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. “Soft Actor Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor”. In: *Proceedings of the 35th International Conference on Machine Learning, PMLR*. Vol. 80. Stockholm, Sweden, 2018, pp. 1861–1870.

[HBZ04] Eric A. Hansen, Daniel S. Bernstein, and Shlomo Zilberstein. “Dynamic Programming for Partially Observable Stochastic Games”. In: *Proceedings of the Nineteenth National Conference on Artificial Intelligence*. San Jose, USA, 2004, pp. 709–715.

[He+16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. “Deep Residual Learning for Image Recognition”. In: *Conference on Computer Vision and Pattern Recognition*. Las Vegas, USA: IEEE/CVF, 2016, pp. 770–778.

[HCL19] Mikael Henaff, Alfredo Canziani, and Yann LeCun. “Model-Predictive Policy Learning with Uncertainty Regularization for Driving in Dense Traffic”. In: *International Conference on Learning Representations*. New Orleans, USA, 2019.

[HE16] Jonathan Ho and Stefano Ermon. “Generative Adversarial Imitation Learning”. In: *Advances in Neural Information Processing Systems*. Vol. 29. Barcelona, Spain, 2016.

[HWL18] Carl-Johan Hoel, Krister Wolff, and Leo Laine. “Automated Speed and Lane Change Decision Making using Deep Reinforcement Learning”. In: *International Intelligent Transportation Systems Conference*. Maui, USA: IEEE, 2018, pp. 2148–2155.

[HWL20] Carl-Johan Hoel, Krister Wolff, and Leo Laine. “Tactical Decision-Making in Autonomous Driving by Reinforcement Learning with Uncertainty Estimation”. In: *Intelligent Vehicles Symposium*. Las Vegas, USA: IEEE, 2020, pp. 1563–1569.

[HBD18] Stefan Hoermann, Martin Bach, and Klaus Dietmayer. “Dynamic Occupancy Grid Prediction for Urban Autonomous Driving: A Deep Learning Approach with Fully Automatic Labeling”. In: *International Conference on Robotics and Automation*. Brisbane, Australia: IEEE, 2018, pp. 2056–2063.

[HSD17] Stefan Hoermann, Daniel Stumper, and Klaus Dietmayer. “Probabilistic long-term prediction for autonomous vehicles”. In: *Intelligent Vehicles Symposium*. Los Angeles, USA: IEEE, 2017, pp. 237–243.

[HSP19] Joey Hong, Benjamin Sapp, and James Philbin. “Rules of the Road: Predicting Driving Behavior With a Convolutional Model of Semantic Interactions”. In: *Conference on Computer Vision and Pattern Recognition*. Long Beach, USA: IEEE/CVF, 2019, pp. 8446–8454.

[Hou+20] J. Houston, G. Zuidhof, L. Bergamini, Y. Ye, A. Jain, S. Omari, V. Iglovikov, and P. Ondruska. “One Thousand and One Hours: Self-driving Motion Prediction Dataset”. In: *Proceedings of the Conference on Robot Learning, PMLR*. Vol. 155. Online event, 2020, pp. 409–418.

[Hua+22] Yanjun Huang, Jiatong Du, Ziru Yang, Zewei Zhou, Lin Zhang, and Hong Chen. “A Survey on Trajectory-Prediction Methods for Autonomous Driving”. In: *IEEE Transactions on Intelligent Vehicles* 7.3 (2022), pp. 652–674.

[Hub64] Peter J. Huber. “Robust Estimation of a Location Parameter”. In: *Annals of Statistic* 53.1 (1964), pp. 73–101.

[Hub+17] Constantin Hubmann, Marvin Becker, Daniel Althoff, David Lenz, and Christoph Stiller. “Decision making for autonomous driving considering interaction and uncertain prediction of surrounding vehicles”. In: *Intelligent Vehicles Symposium*. Los Angeles, USA: IEEE, 2017, pp. 1671–1678.

[Hub+18] Constantin Hubmann, Jens Schulz, Marvin Becker, Daniel Althoff, and Christoph Stiller. “Automated Driving in Uncertain Environments: Planning With Interaction and Uncertain Maneuver Prediction”. In: *IEEE Transactions on Intelligent Vehicles*. Vol. 3. 2018, pp. 5–17.

[HW21] Eyke Hüllermeier and Willem Waegeman. “Aleatoric and epistemic uncertainty in machine learning: an introduction to concepts and methods”. In: *Machine Learning* 110.3 (2021), pp. 457–506.

[Iba+21] Julian Ibarz, Jie Tan, Chelsea Finn, Mrinal Kalakrishnan, Peter Pastor, and Sergey Levine. “How to train your robot with deep reinforcement learning: lessons we have learned”. In: *The International Journal of Robotics Research* 40.4-5 (2021), pp. 698–721.

[Ing+19] John Ingraham, Adam Riesselman, Chris Sander, and Debora Marks. “Learning Protein Structure with a Differentiable Simulator”. In: *International Conference on Learning Representations*. 2019.

[Ise+18] David Isele, Reza Rahimi, Akansel Cosgun, Kaushik Subramanian, and Kikuo Fujimura. “Navigating Occluded Intersections with Autonomous Vehicles Using Deep Reinforcement Learning”. In: *International Conference on Robotics and Automation*. Brisbane, Australia: IEEE, 2018, pp. 2034–2039.

[JDZ21] Faris Janjoš, Maxim Dolgov, and J. Marius Zöllner. “Self-Supervised Action-Space Prediction for Automated Driving”. In: *Intelligent Vehicles Symposium*. Aachen, Germany: IEEE, 2021, pp. 200–207.

[JDZ22] Faris Janjoš, Maxim Dolgov, and J. Marius Zöllner. “StarNet: Joint Action-Space Prediction with Star Graphs and Implicit Global-Frame Self-Attention”. In: *Intelligent Vehicles Symposium*. Aachen, Germany: IEEE, 2022, pp. 280–286.

[Jay57] E. T. Jaynes. “Information Theory and Statistical Mechanics”. In: *Physical Review* 106.4 (1957), pp. 620–630.

[KP16] Nidhi Kalra and Susan M Paddock. “Driving to Safety”. In: *Transportation Research Part A: Policy and Practice* 94 (2016), pp. 182–193.

[Kam+20] Danial Kamran, Carlos Fernandez Lopez, Martin Lauer, and Christoph Stiller. “Risk-Aware High-level Decisions for Automated Driving at Occluded Intersections with Reinforcement Learning”. In: *Intelligent Vehicles Symposium*. Las Vegas, USA: IEEE, 2020, pp. 1205–1212.

[Kar+22] Phillip Karle, Maximilian Geisslinger, Johannes Betz, and Markus Lienkamp. “Scenario Understanding and Motion Prediction for Autonomous Vehicles - Review and Comparison”. In: *IEEE Transactions on Intelligent Transportation Systems* 23.10 (2022), pp. 16962–16982.

[Kar+20] Tero Karras, Samuli Laine, Miika Aittala, Janne Hellsten, Jaakko Lehtinen, and Timo Aila. “Analyzing and Improving the Image Quality of StyleGAN”. In: *Conference on Computer Vision and Pattern Recognition*. Online event: IEEE/CVF, 2020, pp. 8110–8119.

[Ken+19] Alex Kendall et al. “Learning to Drive in a Day”. In: *International Conference on Robotics and Automation*. Montreal, Canada: IEEE, 2019.

[KTH07] Arne Kesting, Martin Treiber, and Dirk Helbing. “General Lane-Changing Model MOBIL for Car-Following Models”. In: *Transportation Research Record: Journal of the Transportation Research Board* 1999.1 (2007), pp. 86–94.

[KTH09] Arne Kesting, Martin Treiber, and Dirk Helbing. “Enhanced Intelligent Driver Model to Access the Impact of Driving Strategies on Traffic Capacity”. In: *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences* 368.1928 (2009), pp. 4585–4605.

[KKC20] Sanmin Kim, Dongsuk Kum, and Jun Won Choi. “RECUP Net: RECUrsive Prediction Network for Surrounding Vehicle Trajectory Prediction with Future Trajectory Feedback”. In: *International Intelligent Transportation Systems Conference*. Online event (Rhodes, Greece): IEEE, 2020.

[KB15] Diederik P. Kingma and Jimmy Ba. “Adam: A method for stochastic optimization”. In: *International Conference on Learning Representations*. San Diego, USA, 2015.

[KW14] Diederik P. Kingma and Max Welling. “Auto-Encoding Variational Bayes”. In: *International Conference on Learning Representations*. Banff, Canada, 2014.

[Kit+12] Kris M. Kitani, Brian D. Ziebart, James Andrew Bagnell, and Martial Hebert. “Activity Forecasting”. In: *European Conference on Computer Vision*. Florence, Italy, 2012, pp. 201–214.

[KWW22] Mykel J. Kochenderfer, Tim A. Wheeler, and Kyle H. Wray. *Algorithms for decision making*. 1st ed. Cambridge: MIT Press, 2022.

[Kon+15] Jason Kong, Mark Pfeiffer, Georg Schildbach, and Francesco Borrelli. “Kinematic and dynamic vehicle models for autonomous driving control design”. In: *Intelligent Vehicles Symposium*. Seoul, South Korea: IEEE, 2015, pp. 1094–1099.

[KY21] Cevahir Köprülü and Yıldıray Yıldız. “Act to Reason: A Dynamic Game Theoretical Driving Model for Highway Merging Applications”. In: *Conference on Control Technology and Applications*. Online event (San Diego, USA): IEEE, 2021, pp. 747–752.

[KA17] Markus Koschi and Matthias Althoff. “SPOT: A tool for set-based prediction of traffic participants”. In: *Intelligent Vehicles Symposium*. Los Angeles, USA: IEEE, 2017, pp. 1686–1693.

[Krä21] Stefan Krämer. “LiDAR-Based Object Tracking and Shape Estimation”. PhD thesis. Karlsruhe, Germany: Karlsruher Institut für Technologie (KIT), 2021.

[KWG97] S. Krauß, P. Wagner, and C. Gawron. “Metastable states in a microscopic model of traffic flow”. In: *Physical Review E* 55.5 (1997), pp. 5597–5602.

[Kre89] David M. Kreps. “Nash Equilibrium”. In: *Game Theory*. Ed. by John Eatwell, Murray Milgate, and Peter Newman. London: Macmillan, 1989, pp. 167–177.

[KGB15] Markus Kuderer, Shilpa Gulati, and Wolfram Burgard. “Learning driving styles for autonomous vehicles from demonstration”. In: *International Conference on Robotics and Automation*. Seattle, USA: IEEE, 2015, pp. 2641–2646.

[Kue+17] Alex Kuefler, Jeremy Morton, Tim Wheeler, and Mykel Kochenderfer. “Imitating Driver Behavior with Generative Adversarial Networks”. In: *Intelligent Vehicles Symposium*. Los Angeles, USA: IEEE, 2017, pp. 204–211.

[Lee+17] Namhoon Lee, Wongun Choi, Paul Vernaza, Christopher B. Choy, Philip H. S. Torr, and Manmohan Chandraker. “DESIRE: Distant Future Prediction in Dynamic Scenes with Interacting Agents”. In: *Conference on Computer Vision and Pattern Recognition*. Honolulu, HI, USA: IEEE/CVF, 2017, pp. 2165–2174.

[Lef+14] Stéphanie Lefèvre, Chao Sun, Ruzena Bajcsy, and Christian Laugier. “Comparison of parametric and non-parametric approaches for vehicle speed prediction”. In: *American Control Conference*. Portland, OR, USA: IEEE, 2014, pp. 3494–3499.

[LVL14] Stéphanie Lefèvre, Dizan Vasquez, and Christian Laugier. “A survey on motion prediction and risk assessment for intelligent vehicles”. In: *ROBOMECH Journal* 1.1 (2014).

[Len+17] David Lenz, Frederik Diehl, Michael Truong Le, and Alois Knoll. “Deep neural networks for Markovian interactive scene prediction in highway scenarios”. In: *Intelligent Vehicles Symposium*. Los Angeles, USA: IEEE, 2017, pp. 685–692.

[LKK16] David Lenz, Tobias Kessler, and Alois Knoll. “Tactical cooperative planning for autonomous highway driving using Monte-Carlo Tree Search”. In: *Intelligent Vehicles Symposium*. Gothenburg, Sweden: IEEE, 2016, pp. 447–453.

[Lev+20] Sergey Levine, Aviral Kumar, George Tucker, and Justin Fu. *Offline Reinforcement Learning: Tutorial, Review, and Perspectives on Open Problems*. Pre-print, arXiv:2005.01643. Nov. 2020.

[Li+16] Nan Li, Dave Oyler, Mengxuan Zhang, Yildiray Yildiz, Anouck Girard, and Ilya Kolmanovsky. “Hierarchical reasoning game theory based approach for evaluation and testing of autonomous vehicle control systems”. In: *Conference on Decision and Control*. Las Vegas, USA: IEEE, 2016, pp. 727–733.

[Li+18] Nan Li, Dave W. Oyler, Mengxuan Zhang, Yildiray Yildiz, Ilya Kolmanovsky, and Anouck R. Girard. “Game Theoretic Modeling of Driver and Vehicle Interactions for Verification and Validation of Autonomous Vehicle Control Systems”. In: *IEEE Transactions on Control Systems Technology* 26.5 (2018), pp. 1782–1797.

[Li+22] Nan Li, Yu Yao, Ilya Kolmanovsky, Ella Atkins, and Anouck R. Girard. “Game-Theoretic Modeling of Multi-Vehicle Interactions at Uncontrolled Intersections”. In: *IEEE Transactions on Intelligent Transportation Systems* 23.2 (2022), pp. 1428–1442.

[Lia+20] Ming Liang, Bin Yang, Rui Hu, Yun Chen, Renjie Liao, Song Feng, and Raquel Urtasun. “Learning Lane Graph Representations for Motion Forecasting”. In: *European Conference on Computer Vision*. Online event, 2020, pp. 541–556.

[Lie+12] Martin Liebner, Michael Baumann, Felix Klanner, and Christoph Stiller. “Driver intent inference at urban intersections using the intelligent driver model”. In: *Intelligent Vehicles Symposium*. Alcalá de Henares, Madrid, Spain: IEEE, 2012, pp. 1162–1167.

[Lop+18] Pablo Alvarez Lopez et al. “Microscopic Traffic Simulation using SUMO”. In: *International Intelligent Transportation Systems Conference*. Maui, USA: IEEE, 2018, pp. 2575–2582.

[Low+17] Ryan Lowe, Yi Wu, Aviv Tamar, Jean Harb, OpenAI Pieter Abbeel, and Igor Mordatch. “Multi-Agent Actor-Critic for Mixed Cooperative-Competitive Environments”. In: *Advances in Neural Information Processing Systems*. Vol. 30. Long Beach, USA, 2017.

[LYU18] Wenjie Luo, Bin Yang, and Raquel Urtasun. “Fast and Furious: Real Time End-to-End 3D Detection, Tracking and Motion Forecasting with a Single Convolutional Net”. In: *Conference on Computer Vision and Pattern Recognition*. Salt Lake City, USA: IEEE/CVF, 2018, pp. 3569–3577.

[Ma+19] Hengbo Ma, Jiachen Li, Wei Zhan, and Masayoshi Tomizuka. “Wasserstein Generative Learning with Kinematic Constraints for Probabilistic Interactive Driving Behavior Prediction”. In: *Intelligent Vehicles Symposium*. Paris, France: IEEE, 2019, pp. 2477–2483.

[MKA20] Vishal Mahajan, Christos Katrakazas, and Constantinos Antoniou. “Prediction of Lane-Changing Maneuvers with Automatic Labeling and Deep Learning”. In: *Transportation Research Record: Journal of the Transportation Research Board* 2674.7 (2020), pp. 336–347.

[Mak+19] Osama Makansi, Eddy Ilg, Ozgun Cicek, and Thomas Brox. “Overcoming Limitations of Mixture Density Networks: A Sampling and Fitting Framework for Multimodal Future Prediction”. In: *Conference on Computer Vision and Pattern Recognition*. Long Beach, USA: IEEE/CVF, 2019, pp. 7137–7146.

[Met+22] Luke Metz, C. Daniel Freeman, Samuel S. Schoenholz, and Tal Kachman. *Gradients are Not All You Need*. Pre-print, arXiv:2111.05803v2. Jan. 2022.

[Mni+15] Volodymyr Mnih et al. “Human-level control through deep reinforcement learning”. In: *Nature* 518.7540 (2015), pp. 529–533.

[MWK17] Jeremy Morton, Tim A. Wheeler, and Mykel J. Kochenderfer. “Analysis of Recurrent Neural Networks for Probabilistic Modeling of Driver Behavior”. In: *IEEE Transactions on Intelligent Transportation Systems* 18.5 (2017), pp. 1289–1298.

[Moz+22] Sajjad Mozaffari, Omar Y. Al-Jarrah, Mehrdad Dianati, Paul Jennings, and Alexandros Mouzakitis. “Deep Learning-Based Vehicle Behavior Prediction for Autonomous Driving Applications: A Review”. In: *IEEE Transactions on Intelligent Transportation Systems* 23.1 (2022), pp. 33–47.

[Nas51] John Nash. “Non-Cooperative Games”. In: *The Annals of Mathematics* 54.2 (1951), pp. 286–295.

[Nau+20] Maximilian Naumann, Liting Sun, Wei Zhan, and Masayoshi Tomizuka. “Analyzing the Suitability of Cost Functions for Explaining and Imitating Human Driving Behavior based on Inverse Reinforcement Learning”. In: *International Conference on Robotics and Automation*. Online event: IEEE, 2020, pp. 5481–5487.

[NR00] Andrew Y. Ng and Stuart J. Russell. “Algorithms for Inverse Reinforcement Learning”. In: *International Conference on Machine Learning*. Stanford, USA, 2000, pp. 663–670.

[NW89] Nguyen and Widrow. “The truck backer-upper: an example of self-learning in neural networks”. In: *IEEE International Joint Conference on Neural Networks*. Washington, D.C., USA, 1989, 357–363 vol.2.

[OA16] Frans A. Oliehoek and Christopher Amato. *A Concise Introduction to Decentralized POMDPs*. Cham: Springer, 2016.

[Ond+16] Peter Ondrúška, Julie Dequaire, Dominic Zeng Wang, and Ingmar Posner. *End-to-End Tracking and Semantic Segmentation Using Recurrent Neural Networks*. Workshop contribution at Robotics: Science and Systems, Ann Arbor, USA, arXiv:1604.05091. 2016.

[OP16] Peter Ondrúška and Ingmar Posner. “Deep Tracking: Seeing Beyond Seeing Using Recurrent Neural Networks”. In: *Conference on Artificial Intelligence*. Vol. 30. Phoenix, USA: AAAI, 2016.

[Oyl+16] Dave W. Oyler, Yildiray Yildiz, Anouck R. Girard, Nan I. Li, and Ilya V. Kolmanovsky. “A game theoretical model of traffic with multiple interacting drivers for use in autonomous vehicle development”. In: *American Control Conference*. Boston, USA: IEEE, 2016, pp. 1705–1710.

[Par+18a] Fabio Pardo, Arash Tavakoli, Vitaly Levdik, and Petar Kormushev. “Time Limits in Reinforcement Learning”. In: *Proceedings of the 35th International Conference on Machine Learning, PMLR*. Vol. 80. Stockholm, Sweden, 2018, pp. 4045–4054.

[Par+18b] Florian Particke, Markus Hiller, Christian Feist, and Jörn Thielecke. “Improvements in pedestrian movement prediction by considering multiple intentions in a Multi-Hypotheses filter”. In: *Position, Location and Navigation Symposium*. Monterey, USA: IEEE/ION, 2018, pp. 209–212.

[Pea09] Judea Pearl. “Causal inference in statistics: An overview”. In: *Statistics Surveys* 3 (2009), pp. 96–146.

[Pek+20] Christian Pek, Stefanie Manzinger, Markus Koschi, and Matthias Althoff. “Using online verification to prevent autonomous vehicles from causing accidents”. In: *Nature Machine Intelligence* 2.9 (2020), pp. 518–528.

[Pom89] Dean A. Pomerleau. “ALVINN: An Autonomous Land Vehicle in a Neural Network”. In: *Advances in Neural Information Processing Systems*. Denver, USA, 1989, pp. 305–313.

[Pru+20] Sasinee Pruekprasert, Jérémy Dubut, Xiaoyi Zhang, Chao Huang, and Masako Kishida. *A Game-Theoretic Approach to Decision Making for Multiple Vehicles at Roundabout*. Pre-print, arXiv: 1904.06224. May 2020.

[Pru+19] Sasinee Pruekprasert, Xiaoyi Zhang, Jérémy Dubut, Chao Huang, and Masako Kishida. “Decision Making for Autonomous Vehicles at Unsignalized Intersection in Presence of Malicious Vehicles”. In: *International Intelligent Transportation Systems Conference*. Auckland, New Zealand: IEEE, 2019, pp. 2299–2304.

[PS05] Vincenzo Punzo and Fulvio Simonelli. “Analysis and Comparison of Microscopic Traffic Flow Models with Real Traffic Microscopic Data”. In: *Transportation Research Record* 1934.1 (2005), pp. 53–63.

[RK15] Eike Rehder and Horst Kloeden. “Goal-Directed Pedestrian Prediction”. In: *International Conference on Computer Vision Workshop*. Santiago, Chile: IEEE, 2015, pp. 139–147.

[Reh+18] Eike Rehder, Florian Wirth, Martin Lauer, and Christoph Stiller. “Pedestrian Prediction by Planning Using Deep Neural Networks”. In: *International Conference on Robotics and Automation*. Brisbane, Australia: IEEE, 2018, pp. 5903–5908.

[Rei22] Jörg Reichardt. “Trajectories as Markov-States for Long Term Traffic Scene Prediction”. In: *14. Uni-DAS Workshop Fahrerassistenz und automatisiertes Fahren*. Berkheim, Germany, 2022, pp. 21–34.

[Rhi+19] Nicholas Rhinehart, Rowan McAllister, Kris Kitani, and Sergey Levine. “PRECOG: PREdiction Conditioned On Goals in Visual Multi-Agent Settings”. In: *International Conference on Computer Vision*. Seoul, Korea: IEEE/CVF, 2019, pp. 2821–2830.

[Rid+20] Daniela Ridel, Nachiket Deo, Denis Wolf, and Mohan Trivedi. “Scene Compliant Trajectory Forecast With Agent-Centric Spatio-Temporal Grids”. In: *IEEE Robotics and Automation Letters* 5.2 (2020), pp. 2816–2823.

[Rid+18] Daniela Ridel, Eike Rehder, Martin Lauer, Christoph Stiller, and Denis Wolf. “A Literature Review on the Prediction of Pedestrian Behavior in Urban Scenarios”. In: *International Intelligent Transportation Systems Conference*. Maui, USA: IEEE, 2018, pp. 3105–3112.

[Roc+22] Teresa Rock, Mohammad Bahram, Chantal Himmels, and Stefanie Marker. “Quantifying Realistic Behaviour of Traffic Agents in Urban Driving Simulation Based on Questionnaires”. In: *Intelligent Vehicles Symposium*. Aachen, Germany: IEEE, 2022, pp. 1675–1682.

[Ros+19] Sascha Rosbach, Vinit James, Simon Großjohann, Silviu Homoceanu, and Stefan Roth. “Driving with Style: Inverse Reinforcement Learning in General-Purpose Planning for Automated Driving”. In: *International Conference on Intelligent Robots and Systems*. Macau, China: IEEE/RSJ, 2019, pp. 2658–2665.

[RB10] Stephane Ross and Drew Bagnell. “Efficient Reductions for Imitation Learning”. In: *International Conference on Artificial Intelligence and Statistics*. Vol. 9. Sardinia, Italy, 2010, pp. 661–668.

[RGB11] Stephane Ross, Geoffrey J. Gordon, and J. Andrew Bagnell. “A Reduction of Imitation Learning and Structured Prediction to No-Regret Online Learning”. In: *International Conference on Artificial Intelligence and Statistics*. Fort Lauderdale, USA, 2011.

[RK08] Reuven Y. Rubinstein and Dirk P. Kroese. *Simulation and the Monte Carlo Method*. 2nd ed. Wiley-Interscience, 2008.

[RHW86] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. “Learning representations by back-propagating errors”. In: *Nature* 323 (1986), pp. 533–536.

[Sad+18] Dorsa Sadigh, Nick Landolfi, Shankar S. Sastry, Sanjit A. Seshia, and Anca D. Dragan. “Planning for cars that coordinate with people: leveraging effects on human actions for planning and active information gathering over human internal state”. In: *Autonomous Robots* 42.7 (2018), pp. 1405–1426.

[Sal+16] Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, Xi Chen, and Xi Chen. “Improved Techniques for Training GANs”. In: *Advances in Neural Information Processing Systems*. Vol. 29. Barcelona, Spain, 2016, pp. 2234–2242.

[Sam+19] Mikayel Samvelyan et al. “The StarCraft Multi-Agent Challenge”. In: *International Conference on Autonomous Agents and Multiagent Systems*. Montreal, Canada, 2019, pp. 2186–2188.

[San+19] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. “MobileNetV2: Inverted Residuals and Linear Bottlenecks”. In: *Conference on Computer Vision and Pattern Recognition*. Salt Lake City, USA: IEEE/CVF, 2019, pp. 4510–4520.

[Sch19] Alexander Scheel. “Fully Bayesian Vehicle Tracking Using Extended Object Models”. PhD thesis. Ulm, Germany: Universität Ulm, 2019.

[Sch+22] Oliver Scheel, Luca Bergamini, Maciej Wołczyk, Błażej Osiński, and Peter Ondrúška. “Urban Driver: Learning to Drive from Real-world Demonstrations Using Policy Gradients”. In: *Conference on Robot Learning, PMLR*. Vol. 164. London, UK, 2022, pp. 718–728.

[Sch+15a] Julian Schlechtriemen, Florian Wirthmueller, Andreas Wedel, Gabi Breuel, and Klaus-Dieter Kuhnert. “When will it change the lane? A probabilistic regression approach for rarely occurring events”. In: *Intelligent Vehicles Symposium*. Seoul, South Korea: IEEE, 2015, pp. 1373–1379.

[SRW08] Robin Schubert, Eric Richter, and Gerd Wanielik. “Comparison and Evaluation of Advanced Motion Models for Vehicle Tracking”. In: *International Conference on Information Fusion*. Cologne, Germany: IEEE, 2008.

[Sch16] John Schulman. “Optimizing Expectations: From Deep Reinforcement Learning to Stochastic Computation Graphs”. PhD thesis. Berkeley, USA: University of California, 2016.

[Sch+15b] John Schulman, Sergey Levine, Philipp Moritz, Michael I. Jordan, and Pieter Abbeel. “Trust Region Policy Optimization”. In: *International Conference on Machine Learning*. Lille, France, 2015, pp. 1889–1897.

[Sch+18a] John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. “High-Dimensional Continuous Control Using Generalized Advantage Estimation”. In: *International Conference on Learning Representations*. Vancouver, Canada, 2018.

[Sch+17a] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. *Proximal Policy Optimization Algorithms*. Pre-print, arXiv: 1707.06347v2. Aug. 2017.

[Sch21] Jens Schulz. “Interaction-Aware Probabilistic Behavior Prediction of Traffic Participants in Urban Environments”. PhD thesis. Munich, Germany: Technische Universität München, 2021.

[Sch+17b] Jens Schulz, Kira Hirsenkorn, Julian Lochner, Moritz Werling, and Darius Burschka. “Estimation of collective maneuvers through cooperative multi-agent planning”. In: *Intelligent Vehicles Symposium*. Los Angeles, USA: IEEE, 2017, pp. 624–631.

[Sch+18b] Jens Schulz, Constantin Hubmann, Julian Löchner, and Darius Burschka. “Interaction-Aware Probabilistic Behavior Prediction in Urban Environments”. In: *International Conference on Intelligent Robots and Systems*. Madrid, Spain: IEEE, 2018, pp. 3999–4006.

[Sch+18c] Jens Schulz, Constantin Hubmann, Julian Löchner, and Darius Burschka. “Multiple Model Unscented Kalman Filtering in Dynamic Bayesian Networks for Intention Estimation and Trajectory Prediction”. In: *International Intelligent Transportation Systems Conference*. Maui, USA: IEEE, 2018, pp. 1467–1474.

[Sch+19] Jens Schulz, Constantin Hubmann, Nikolai Morin, Julian Löchner, and Darius Burschka. “Learning Interaction-Aware Probabilistic Driver Behavior Models from Urban Scenarios”. In: *Intelligent Vehicles Symposium*. Paris, France: IEEE, 2019, pp. 1326–1333.

[Ści+21] Adam Ścibior, Vasileios Lioutas, Daniele Reda, Peyman Bateni, and Frank Wood. “Imagining The Road Ahead: Multi-Agent Trajectory Prediction via Differentiable Simulation”. In: *International Intelligent Transportation Systems Conference*. Indianapolis, USA: IEEE, 2021, pp. 720–725.

[SW65] S. S. Shapiro and M. B. Wilk. “An Analysis of Variance Test for Normality (Complete Samples)”. In: *Biometrika* 52.3/4 (1965), p. 591.

[Sil+17] David Silver et al. “Mastering the game of Go without human knowledge”. In: *Nature* 550.7676 (2017), pp. 354–359.

[Spe+21] Jonathan Spencer, Sanjiban Choudhury, Arun Venkatraman, Brian Ziebart, and J. Andrew Bagnell. *Feedback in Imitation Learning: The Three Regimes of Covariate Shift*. Pre-print, arXiv: 2102.02872. Feb. 2021.

[Sri+17] Akash Srivastava, Lazar Valkov, Chris Russell, Michael U Gutmann, and Charles Sutton. “VEEGAN: Reducing Mode Collapse in GANs using Implicit Variational Learning”. In: *Advances in Neural Information Processing Systems*. Vol. 30. Long Beach, USA, 2017.

[Sri+14] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. “Dropout: A Simple Way to Prevent Neural Networks from Overfitting”. In: *The journal of machine learning research* 15.1 (2014), pp. 1929–1958.

[SW95] Dale O. Stahl and Paul W. Wilson. “On Players’ Models of Other Players: Theory and Experimental Evidence”. In: *Games and Economic Behavior* 10 (1995), pp. 218–254.

[SH14] Thomas Streubel and Karl Heinz Hoffmann. “Prediction of driver intended path at intersections”. In: *Intelligent Vehicles Symposium*. MI, USA: IEEE, 2014, pp. 134–139.

[Suo+21] Simon Suo, Sebastian Regalado, Sergio Casas, and Raquel Urtasun. “TrafficSim: Learning to Simulate Realistic Multi-Agent Behaviors”. In: *Conference on Computer Vision and Pattern Recognition*. Online event: IEEE/CVF, 2021.

[SB18] Richard S. Sutton and Andrew G. Barto. *Reinforcement learning: an introduction*. 2nd ed. Cambridge, Massachusetts: The MIT Press, 2018.

[Tan93] Ming Tan. “Multi-Agent Reinforcement Learning: Independent vs. Cooperative Agents”. In: *Proceedings of the 10th International Conference on Machine Learning*. 1993, pp. 330–337.

[TS19] Charlie Tang and Ruslan Salakhutdinov. “Multiple Futures Prediction”. In: *Advances in Neural Information Processing Systems 32*. Vancouver, Canada, 2019, pp. 15424–15434.

[Tol+21] Ekaterina Tolstaya, Reza Mahjourian, Carlton Downey, Balakrishnan Vadarajan, Benjamin Sapp, and Dragomir Anguelov. “Identifying Driver Interactions via Conditional Behavior Prediction”. In: *International Conference on Robotics and Automation*. Xi’an, China: IEEE, 2021, pp. 3473–3479.

[TF13] Quan Tran and Jonas Firl. “Modelling of traffic situations at urban intersections with probabilistic non-parametric regression”. In: *Intelligent Vehicles Symposium*. Gold Coast City, Australia: IEEE, 2013, pp. 334–339.

[TF14] Quan Tran and Jonas Firl. “Online maneuver recognition and multimodal trajectory prediction for intersection assistance using non-parametric regression”. In: *Intelligent Vehicles Symposium Proceedings*. Dearborn, USA: IEEE, 2014, pp. 918–923.

[TK10] Peter Trautman and Andreas Krause. “Unfreezing the robot: Navigation in dense, interacting crowds”. In: *International Conference on Intelligent Robots and Systems*. Taipei, Taiwan: IEEE/RSJ, 2010, pp. 797–803.

[THH00] Martin Treiber, Ansgar Hennecke, and Dirk Helbing. “Congested Traffic States in Empirical Observations and Microscopic Simulations”. In: *Physical Review E* 62.2 (2000), pp. 1805–1824.

[TK09] Martin Treiber and Arne Kesting. “Modeling Lane-Changing Decisions with MOBIL”. In: *Traffic and Granular Flow ’07*. Berlin, Heidelberg: Springer, 2009, pp. 211–221.

[TK13a] Martin Treiber and Arne Kesting. “Microscopic Calibration and Validation of Car-Following Models – A Systematic Approach”. In: *Procedia - Social and Behavioral Sciences* 80 (2013), pp. 922–939.

[TK13b] Martin Treiber and Arne Kesting. *Traffic Flow Dynamics*. Berlin, Heidelberg: Springer, 2013.

[Um+20] Kiwon Um, Robert Brand, Yun (Raymond) Fei, Philipp Holl, and Nils Thuerey. “Solver-in-the-Loop: Learning from Differentiable Physics to Interact with Iterative PDE-Solvers”. In: *Advances in Neural Information Processing Systems*. Vol. 33. Curran Associates, Inc., 2020, pp. 6111–6122.

[Van+18] Jessica Van Brummelen, Marie O’Brien, Dominique Gruyer, and Homayoun Najjaran. “Autonomous vehicle perception: The technology of today and tomorrow”. In: *Transportation Research Part C: Emerging Technologies* 89 (2018), pp. 384–406.

[Var+22] Balakrishnan Varadarajan et al. “MultiPath++: Efficient Information Fusion and Trajectory Aggregation for Behavior Prediction”. In: *International Conference on Robotics and Automation*. arXiv: 2111.14973. IEEE, 2022.

[WW16] Walther Wachenfeld and Hermann Winner. “The Release of Autonomous Vehicles”. In: *Autonomous Driving: Technical, Legal and Social Aspects*. Ed. by Markus Maurer, J. Christian Gerdes, Barbara Lenz, and Hermann Winner. Berlin, Heidelberg: Springer, 2016, pp. 425–449.

[WV00] E.A. Wan and R. Van Der Merwe. “The unscented Kalman filter for nonlinear estimation”. In: *Adaptive Systems for Signal Processing, Communications, and Control Symposium*. Lake Louise, Alta., Canada: IEEE, 2000, pp. 153–158.

[WQ01] Danwei Wang and Feng Qi. “Trajectory planning for a four-wheel-steering vehicle”. In: *International Conference on Robotics and Automation*. Vol. 4. IEEE, 2001, pp. 3320–3325.

[Wan+21] Pin Wang, Dapeng Liu, Jiayu Chen, Hanhan Li, and Ching-Yao Chan. “Decision Making for Autonomous Driving via Augmented Adversarial Inverse Reinforcement Learning”. In: *IEEE International Conference on Robotics and Automation*. Xi’an, China, 2021.

[WKA21] Xiao Wang, Hanna Krasowski, and Matthias Althoff. “CommonRoad-RL: A Configurable Reinforcement Learning Environment for Motion Planning of Autonomous Vehicles”. In: *International Intelligent Transportation Systems Conference*. IEEE, 2021.

[WRK16] Tim A. Wheeler, Philipp Robbel, and Mykel J. Kochenderfer. “Analysis of microscopic behavior models for probabilistic modeling of driver behavior”. In: *International Intelligent Transportation Systems Conference*. Rio de Janeiro, Brazil: IEEE, 2016, pp. 1604–1609.

[WB91] Steven D. Whitehead and Dana H. Ballard. “Learning to perceive and act by trial and error”. In: *Machine Learning* 7.1 (1991), pp. 45–83.

[Wie+12] Jürgen Wiest, Matthias Höffken, Ulrich Kreßel, and Klaus Dietmayer. “Probabilistic trajectory prediction with Gaussian mixture models”. In: *2012 IEEE Intelligent Vehicles Symposium*. Alcal de Henares , Madrid, Spain: IEEE, 2012, pp. 141–146.

[Wie+13] Jürgen Wiest, Felix Kunz, Ulrich Kreßel, and Klaus Dietmayer. “Incorporating Categorical Information for Enhanced Probabilistic Trajectory Prediction”. In: *International Conference on Machine Learning and Applications*. Vol. 1. 2013, pp. 402–407.

[Wil92] Ronald J Williams. “Simple statistical gradient-following algorithms for connectionist reinforcement learning”. In: *Machine Learning* 8 (1992), pp. 229–256.

[Wil+21] Benjamin Wilson et al. “Argoverse 2: Next Generation Datasets for Self-Driving Perception and Forecasting”. In: *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*. 2021.

[Win+16] Hermann Winner, Stephan Hakuli, Felix Lotz, and Christina Singer, eds. *Handbook of Driver Assistance Systems*. Cham: Springer International Publishing, 2016.

[Wis20] Christian Wissing. “Trajektorienprädiktion für das automatisierte Fahren”. PhD thesis. Technische Universität Dortmund, 2020.

[Woo+17] Hanwool Woo et al. “Lane-Change Detection Based on Vehicle-Trajectory Prediction”. In: *IEEE Robotics and Automation Letters* 2.2 (2017), pp. 1109–1116.

[Xie+18] Guotao Xie, Hongbo Gao, Lijun Qian, Bin Huang, Keqiang Li, and Jianqiang Wang. “Vehicle Trajectory Prediction by Integrating Physics- and Maneuver-Based Approaches Using Interactive Multiple Models”. In: *IEEE Transactions on Industrial Electronics* 65.7 (2018), pp. 5999–6008.

[YL12] Je Hong Yoo and Reza Langari. “Stackelberg Game Based Model of Highway Driving”. In: *Dynamic Systems and Control Conference*. Fort Lauderdale, Florida, USA: ASME, 2012, pp. 499–508.

[YL13] Je Hong Yoo and Reza Langari. “A Stackelberg Game Theoretic Driver Model for Merging”. In: *Dynamic Systems and Control Conference*. Palo Alto, California, USA: ASME, 2013.

[Yu+22] Chao Yu, Akash Velu, Eugene Vinitsky, Yu Wang, Alexandre Bayen, and Yi Wu. “The Surprising Effectiveness of PPO in Cooperative, Multi-Agent Games”. In: *Neural Information Processing Systems*. arXiv: 2103.01955v3. 2022.

[Zha+19] Wei Zhan et al. *INTERACTION Dataset: An INTERnational, Adversarial and Cooperative moTION Dataset in Interactive Driving Scenarios with Semantic Maps*. Pre-print, arXiv:1910.03088 [cs, eess]. Sept. 2019.

[Zha+22] Aston Zhang, Zachary C. Lipton, Mu Li, and Alexander J. Smola. *Dive into Deep Learning*. version 1.0.0-alpha0. Cambridge University Press, 2022.

[ZY20] Hongming Zhang and Tianyang Yu. “Taxonomy of Reinforcement Learning Algorithms”. In: *Deep Reinforcement Learning: Fundamentals, Research and Applications*. Singapore: Springer, 2020, pp. 125–133.

[Zha+20] Lingyao Zhang, Po-Hsun Su, Jerrick Hoang, Galen Clark Haynes, and Micol Marchetti-Bowick. “Map-Adaptive Goal-Based Trajectory Prediction”. In: *Conference on Robot Learning*. arXiv: 2009.04450. 2020.

[Zha+21] Hang Zhao et al. “TNT: Target-driveN Trajectory Prediction”. In: *Conference on Robot Learning*. 2021.

[Zha+17] M. Zhao, D. Käthner, M. Jipp, D. Söffker, and K. Lemmer. “Modeling driver behavior at roundabouts: Results from a field study”. In: *Intelligent Vehicles Symposium*. Los Angeles, CA, USA: IEEE, 2017, pp. 908–913.

[Zie+08] Brian D. Ziebart, Andrew Maas, J. Andrew Bagnell, and Anind K Dey. “Maximum Entropy Inverse Reinforcement Learning”. In: *Conference on Artificial Intelligence*. AAAI, 2008.

[Zie+09] Brian D. Ziebart et al. “Planning-based prediction for pedestrians”. In: *International Conference on Intelligent Robots and Systems*. IEEE/RSJ, 2009, pp. 3931–3936.

[ZWN20] Alex Zyner, Stewart Worrall, and Eduardo Nebot. “Naturalistic Driver Intention and Path Prediction using Recurrent Neural Networks”. In: *IEEE Transactions on Intelligent Transportation Systems* (2020). arXiv: 1807.09995.

[Zyn+17] Alex Zyner, Stewart Worrall, James Ward, and Eduardo Nebot. “Long short term memory for driver intent prediction”. In: *Intelligent Vehicles Symposium*. IEEE, 2017, pp. 1484–1489.

Online Sources

[DLR] German Aerospace Center (DLR). *SUMO Documentation: Car-Following Models*. Accessed on June 13, 2022. URL: https://sumo.dlr.de/docs/Definition_of_Vehicles%2C_Vehicle_Types%2C_and_Routes.html#car-following_models.

[Ach18] Joshua Achiam. *Spinning Up in Deep RL*. accessed August 8, 2022. 2018. URL: https://spinningup.openai.com.

[Ack21] Evan Ackerman. *What Full Autonomy Means for the Waymo Driver - IEEE Spectrum*. Accessed on April 20, 2023. 2021. URL: https://spectrum.ieee.org/full-autonomy-waymo-driver.

[ADA22] ADAC. *Audi A6 Avant: Gute Noten im Test*. Accessed on August 20, 2022. Mar. 2022. URL: http://web.archive.org/web/20220420184213/https://www.adac.de/rund-ums-fahrzeug/autokatalog/marken-modelle/audi/audi-a6-avant/.

[AG22] Audi AG. *Audi A6 Katalog*. Accessed on August 20, 2022. Mar. 2022. URL: http://web.archive.org/web/20220419011940/https://www.audi.de/dam/nemo/models/misc/pdf/my-2022/preislisten/preisliste_a6-limousine_s6-limousine_a6-avant_s6-avant_a6-allroad-quattro.pdf.

[22] *CARLA Documentation: Traffic Manager*. Accessed on June 13, 2022. 2022. URL: https://carla.readthedocs.io/en/latest/adv_traffic_manager/.

[Ing23] David Ingram. *Two companies race to deploy robotaxis in San Francisco. The city wants them to hit the brakes.* Jan. 2023. URL: https://www.nbcnews.com/tech/tech-news/san-francisco-looks-hit-brakes-self-driving-cars-rcna66204 (visited on 04/30/2023).

[JZH20] Hank Jebode, Scott Zagorski, and Gary Heydinger. *Rollover Stability Measurements For 2020 New Car Assessment Program (NCAP)*. Oct. 2020. URL: http://web.archive.org/web/20220820140731/https://lindseyresearch.com/wp-content/uploads/2021/05/NHTSA-2001-9663-0497-2020-NCAP-Rollover-Stability-Measurements-Final-Report.pdf.

[Web14] Marc Weber. *Where to? A History of Autonomous Vehicles*. Section: Curatorial Insights. May 2014. URL: https://computerhistory.org/blog/where-to-a-history-of-autonomous-vehicles/ (visited on 04/30/2023).

Software Packages

[Har+20] Charles R. Harris et al. “Array programming with NumPy”. In: *Nature* 585.7825 (2020), pp. 357–362.

[LeN19] Alexander LeNail. “NN-SVG: Publication-Ready Neural Network Architecture Schematics”. In: *Journal of Open Source Software* 4.33 (2019), p. 747.

[Lia+18] Eric Liang et al. “RLlib: Abstractions for Distributed Reinforcement Learning”. In: *Proceedings of the 35th International Conference on Machine Learning, PMLR*. Vol. 80. Stockholm, Sweden, 2018, pp. 3053–3062.

[Pas+19] Adam Paszke et al. “PyTorch: An Imperative Style, High-Performance Deep Learning Library”. In: *Advances in Neural Information Processing Systems*. Vancouver, Canada, 2019, pp. 8024–8035.

[Raf+21] Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann. “Stable-Baselines3: Reliable Reinforcement Learning Implementations”. In: *The Journal of Machine Learning Research* 22.1 (2021), pp. 12348–12355.

[Tan11] O. Tange. “GNU Parallel - The Command-Line Power Tool”. In: *;login: The USENIX Magazine* 36.1 (2011). Place: Frederiksberg, Denmark, pp. 42–47.

Supervised Theses

[Jun20] Jan Jung. “Lernen von Fahrermodellen in urbanen Situationen”. Bachelor thesis, Friedrich-Alexander-Universität Erlangen-Nürnberg. 2020.

[Kon21] Fabian Konstantinidis. “Reinforcement Learning zur Vorhersage von Fahrzeugtrajektorien in Kreisverkehrssituationen”. Master thesis, Technical University of Munich. 2021.

[Lee20] Tobias Leemann. “Probabilistische Prädiktion von Fahrzeugtrajektorien mit Methoden des maschinellen Lernens”. Master thesis, Friedrich-Alexander-Universität Erlangen-Nürnberg. 2020.

[Oer19] Moritz Oertel. “Optimierungsbasierte Generierung von Trajektorien”. Master thesis, Friedrich-Alexander-Universität Erlangen-Nürnberg. 2019.

[Pij20] Lucas Pijnacker Hordijk. “From Axis-aligned to Oriented Bounding Boxes”. Master thesis, Delft University of Technology. 2020.

[Sei19] Fabian Seidl. “Parameteridentifikation von Fahrermodellen zur Erzeugung probabilistischer Vorhersagen”. Master thesis, Friedrich-Alexander-Universität Erlangen-Nürnberg. 2019.

[Vog20] Carina Vogl. “Driving Maneuver Prediction at Roundabouts Using LSTM Networks”. Master thesis, Technical University of Munich. 2020.

[Wel19] Edgar Welte. “Modellierung des Verhaltens von Verkehrsteilnehmern als Switching Linear Dynamical System”. Bachelor Thesis, Technische Hochschule Ingolstadt. 2019.