Fakultät für Informatik
Technische Universität München

Learning, Evaluating and Optimizing Behavior Policies for Autonomous Vehicles

Patrick Christopher Hart

Vollständiger Abdruck der von der Fakultät für Informatik der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktor der Naturwissenschaften (Dr. rer. nat.)

genehmigten Dissertation.

Vorsitzender:
Prof. Dr. Helmut Seidl

Prüfende der Dissertation:
1. Prof. Dr.-Ing. habil. Alois Knoll
2. Prof. Mykel Kochenderfer, Ph.D.

Die Dissertation wurde am 29.03.2021 bei der Technischen Universität München eingereicht und durch die Fakultät für Informatik am 11.08.2021 angenommen.

Abstract

With autonomous vehicles operating in mixed traffic, they need to learn and adapt their driving behavior over time to integrate themselves seamlessly. Therefore, data-driven approaches are required that can provide safe and comfortable driving behaviors. This thesis' structure is three-fold: First, it focuses on how to learn behavior policies and discusses related topics. The second part focuses on evaluating and gaining insights into learned behavior policies at runtime. Finally, the third part introduces a post-optimization that generates smooth behaviors.

The first chapter discusses and introduces the Markov decision process (MDP) problem formulation, discusses input representations, reward signals, and solution methods for learning behavior policies. As the order and number of vehicles in traffic can change quickly, the methodologies should be able to handle a varying number and order of vehicles. This work proposes a novel graph neural network (GNN) architecture for actor-critic reinforcement learning that is invariant to the number and order of vehicles. Further, due to the GNN having structured edges, the information flow between vehicles in the behavior policy can be evaluated – providing additional insights into the learned behavior policies. Various input representations and reward signals are proposed and benchmarked in terms of their performance. Finally, variational studies are performed where the behavior of the other vehicles is changed to evaluate the generalization of the used methodologies. The novel GNN actor-critic architecture is shown to outperform conventional approaches in their performance and generalization capabilities.

Behavior policies using deep neural networks (DNNs) cannot guarantee safe behaviors, e.g., due to the lack of full exploration or due to the highly nonlinear nature of DNNs. Therefore, a runtime counterfactual behavior policy evaluation (CBPE) is proposed that uses non-factual worlds to ask and answer counterfactuals, such as "Would the learned behavior policy have been collision-free if the other vehicle had changed lanes?". By deriving counterfactual worlds at every time step and forward-simulating these, evidence on the policies' performance can be obtained for the current scenario. The CBPE has been shown to provide additional insights into the performance and relations of the learned behavior policies. Restricting the learned behavior policies' usage based on their performance over the counterfactual worlds has significantly increased the success rate.

Finally, a post-optimization is introduced to obtain smooth behavior trajectories that utilizes learned behavior policies to generate initial estimates. Using the initial estimate also avoids optimizing multiple combinatorial options, such as merging behind or in front of another vehicle, as the combinatorial option is implicitly chosen by the learned behavior policy. Further, by enforcing proximity of the optimized

to the initial trajectory, interactions of the learned behavior policy with other traffic participants remain valid – e.g., nudging slightly in front of the other vehicle to slow it down. The post-optimization has been shown to produce significantly smoother trajectories whilst upholding the same safety constraints. All experiments and evaluations have been performed using the simulation framework Behavior benchmARK (BARK).

Zusammenfassung

Autonome Fahrzeuge im gemischten Verkehr müssen ihr Fahrverhalten im Laufe der Zeit lernen und anpassen, um sich nahtlos zu integrieren. Daher sind datengesteuerte Ansätze erforderlich, die ein sicheres und komfortables Fahrverhalten ermöglichen. Der Aufbau dieser Arbeit ist dreigeteilt: Erstens konzentriert sie sich auf das Erlernen von Verhaltensstrategien und diskutiert verwandte Themen. Der zweite Teil konzentriert sich auf die Bewertung und Gewinnung von Einblicken in gelernte Verhaltensstrategien zur Laufzeit. Im dritten Teil wird eine Nachoptimierung eingeführt, welche ein glattes Verhalten erzeugt.

Das erste Kapitel diskutiert und führt die MDP-Problemformulierung ein, diskutiert Eingabedarstellungen, Belohnungssignale und Lösungsmethoden für gelernte Verhaltensstrategien. Da sich die Reihenfolge und die Anzahl der Fahrzeuge im Verkehr schnell ändern können, sollten Methoden in der Lage sein, eine unterschiedliche Anzahl und Reihenfolge von Fahrzeugen zu handhaben. Diese Arbeit schlägt eine neuartige GNN actor-critic Architektur vor, die invariant gegenüber der Anzahl und Reihenfolge der Fahrzeuge ist. Da GNNs strukturierte Kanten aufweisen, kann außerdem der Informationsfluss zwischen Fahrzeugen in der Verhaltensstrategie bewertet werden – was zusätzliche Einblicke in die gelernten Verhaltensstrategien liefert. Es werden verschiedene Eingabedarstellungen und Belohnungssignale vorgeschlagen und hinsichtlich ihrer Leistung bewertet. Schließlich werden Variationsstudien durchgeführt, bei denen das Verhalten der anderen Fahrzeuge geändert wird, um die Generalisierung der verwendeten Methoden zu bewerten. Es wird gezeigt, dass die neuartige GNN actor-critic Architektur konventionelle Ansätze in ihrer Leistungsfähigkeit und Generalisierung übertrifft.

Verhaltensstrategien, die DNNs als Approximationsfunktion verwenden, können kein sicheres Verhalten gewährleisten, z.B. aufgrund unvollständiger Exploration oder aufgrund der stark nichtlinearen Natur von DNNs. Daher wird eine Laufzeit CBPE vorgeschlagen, die nicht-faktische Welten verwendet, um kontrafaktische Dinge zu fragen und zu beantworten, wie zum Beispiel “Wäre die gelernte Verhaltensstrategie kollisionsfrei gewesen, wenn das andere Fahrzeug die Spur gewechselt hätte?”. Durch die Ableitung kontrafaktischer Welten zu jedem Zeitschritt und deren Vorwärtssimulation können Nachweise für die Leistung der Verhaltensstrategie für das aktuelle Szenario gewonnen werden. Es hat sich gezeigt, dass CBPE zusätzliche Einblicke in die Leistung und Beziehungen der gelernten Verhaltensstrategien liefert. Das Einschränken der Verwendung der gelernten Verhaltensstrategien basierend auf ihrer Leistung in den kontrafaktischen Welten erhöht die Erfolgsquote erheblich.

Schließlich wird eine Nachoptimierung eingeführt, um glatte Verhaltenstrajektorien zu erhalten, welche gelernte Verhaltensstrategien verwendet, um initiale Schätzungen

zu generieren. Die Verwendung der initialen Schätzung vermeidet die Optimierung mehrerer kombinatorischer Optionen, wie beispielsweise hinter oder vor einem anderen Fahrzeug die Spur zu wechseln, da die kombinatorische Option implizit von der gelernten Verhaltensstrategie gewählt wird. Durch Erzwingen von Nähe der optimierten zur initialen Trajektorie bleiben außerdem Interaktionen der gelernten Verhaltensstrategie mit anderen Verkehrsteilnehmern gültig – z.B. leichtes Reindrängeln vor einem anderen Fahrzeug, um es zu verlangsamen. Es hat sich gezeigt, dass die Nachoptimierung deutlich glattere Trajektorien generiert, während die gleichen Sicherheitsbeschränkungen eingehalten werden. Alle Experimente und Auswertungen wurden mit dem Simulationsframework BARK durchgeführt.

Contents

List of Figures

List of Tables

Acronyms

AC	actor-critic.
BARK	Behavior benchmARK.
BARK-ML	Behavior benchmARK - Machine Learning.
BFGS	Broyden-Fletcher-Goldfarb-Shanno.
CBPE	counterfactual behavior policy evaluation.
CNN	convolutional neural network.
ConvGNN	convolutional graph neural network.
DDPG	deep deterministic policy gradient.
DDQN	double deep Q-learning.
DNN	deep neural network.
DP	dynamic programming.
DPG	deep policy gradient.
DQN	deep Q-network.
GAE	graph autoencoder.
GAN	generative adversarial network.
GN	graph network.
GN block	graph network block.
GNN	graph neural network.
GPU	graphics processing unit.
IDM	intelligent driver model.
IL	imitation learning.
IN	interaction network.
InfoGAIL	info generative adversarial imitation learning.
IP	interior point.
IVP	initial value problem.
KKT	Karush-Kuhn-Tucker.
KL	Kullback-Leiber.
LUT	lookup table.

MCM	Monte Carlo method.
MCTS	Monte-Carlo tree search.
MDP	Markov decision process.
MILP	mixed integer linear programming.
MIQP	mixed integer quadratic programming.
MOBIL	minimizing overall braking induced by lane changes.
MPC	model predictive control.
ODD	operational design domain.
PPO	proximal policy optimization.
PRM	probabilistic road-maps.
PRM*	probabilistic road-maps*.
QP	quadratic program.
RecGNN	recurrent graph neural network.
RL	reinforcement learning.
RQ1	research question 1.
RQ2	research question 2.
RQ3	research question 3.
RRT	rapidly-exploring random tree.
RRT*	rapidly-exploring random tree*.
RSS	responsibility-sensitive safety.
RTSA	runtime safety assurance.
SAC	soft actor critic.
SARSA	state action reward state action.
SL	supervised learning.
SOTIF	safety of the intended functionality.
SQP	sequential quadratic program.
SRL	safe reinforcement learning.
STGNN	spatial-temporal graph neural network.
TD	temporal difference.
TD3	twin-delayed DDPG.
TRPO	trust region policy optimization.
USL	unsupervised learning.
V&V	verification and validation.

1 Introduction

1.1 Motivation

Autonomous driving is one of the key technologies to stem the increased need for mobility in the future. Not only does it have the potential to lower the total number of vehicles on roads and to decrease the downtime of vehicles, but it also has the potential to make traffic safer overall.

Developing autonomous vehicles poses a challenging task as these need to adhere to societal norms and – to be accepted by the broader public – need to operate safer than humans do. Using the German traffic statistics of severely injured people as shown in Figure 1.1 and taking into account that all vehicles drove a total of 738 billion kilometers in Germany in 2019, autonomous vehicles are required to operate having a collision probability lower than 8.83×10^{-8} per driven kilometer [107, 62].

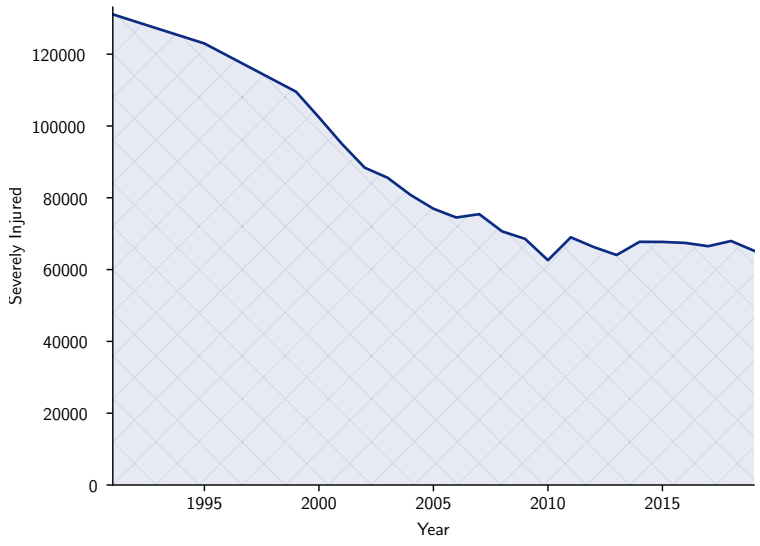


Figure 1.1: Severely injured traffic participants in Germany from 1991 until 2019 [107].

Not only do autonomous vehicles need to cater to the above-stated collision probabilities, but they also have to blend seamlessly into mixed traffic — be it with other autonomous vehicles or human drivers. At first glance, this might not seem as important as the other stated attributes but blending into mixed traffic

plays a vital role in the overall acceptance of autonomous vehicles. For example, an autonomous vehicle standing on a lane for a few minutes trying to merge would lower their acceptance. "Blending in seamlessly into traffic" requires the behavior generation of autonomous vehicles to interact with other traffic participants and, further, stipulates the need to learn from these interactions (experiences).

Conventional methods for behavior generation, such as optimization-based approaches, can guarantee collision-free and comfortable behaviors and incorporate traffic rules [30]. However, e.g., optimization-based methods often fall short when it comes to "seamlessly blending into mixed traffic" as these often require exhaustive parameter-tuning for specific scenarios.

The lack of scalability of these approaches motivates the need for learning-based methods that excel at learning from experience and adapting their driving behavior over time. These methods can learn from experiences and actively explore the configuration space by interacting with the environment. For example, in a merging scenario, the other vehicle might give way if the ego vehicle slightly nudges in front. However, as many learning-based approaches use deep neural networks as function approximators, these cannot give the same safety and comfort guarantees as, e.g., optimization-based approaches.

This work generates synergies by combining learning- and optimization-based methodologies. Learning-based approaches are used to generate interactive behaviors, and optimization-based ones are used to smoothen the learned trajectory whilst adhering to extracted safety constraints.

This work formulates learning behavior policies as an MDP and solves it using actor-critic reinforcement learning. Input representations are proposed and discussed in terms of their efficiency, invariance, and other criteria. Further, the choice of reward signals is discussed and novel potential-based reward shaping functions for learning behavior policies are proposed. As the number and order of vehicles in traffic can change rapidly, a novel GNN actor-critic architecture is proposed that is invariant to the number and order of vehicles. Graphs as an input representation are highlighted, and several other advantages are discussed in Section 2.2. Further, using GNNs, structured elements are introduced where the information flow between nodes (vehicles) can be evaluated – providing additional insights into the learned behavior policy. It is shown that the novel GNN actor-critic architecture outperforms conventional DNNs in performance and also generalizes better. An additional benefit is the reduced number of parameters of the novel GNN actor-critic architecture lowering the memory requirement.

As DNNs might not always generalize well or the state- and action-space have not been fully explored during training, a CBPE is introduced to gain insights on the performance of the learned behavior policy at runtime. The performance of learned behavior policies is evaluated using counterfactual worlds to gain introspection on, e.g., the behavior policies' generalization or whether and how it handles distributional shifts. Further, relations of how behavior policies influence each other can be extracted. Finally, the policies' usage is restricted based on its performance over all counterfactual worlds – increasing the success rate significantly.

To obtain smooth behavior trajectories from learned behavior policies, a post-optimization is introduced that utilizes an initial estimate generated by the learned behavior policy and smoothens the behavior trajectory. This is achieved by minimizing the jerk whilst adhering to extracted constraints from the learning-based initial estimate. The optimized solution is constrained to be close to the initial estimate for the interactions with other vehicles to remain valid. Further, using learning-based initial estimates mitigates optimizing all combinatorial options to obtain well-performing behavior trajectories. The optimization is further warm-started using the input control sequence obtained by the learning-based behavior policy, which speeds up the time-to-convergence. By combining learning- and optimization-based methods, the resulting behavior generation is interactive, safe, and produces comfortable trajectories. The proposed method can adapt its behavior over time to blend into mixed traffic without requiring extensive parameter-tuning.

In summary, this work provides a thorough investigation and benchmarking of input representations and reward signals for learning behavior policies for autonomous vehicles. State-of-the-art actor-critic reinforcement learning is applied to learn interactive behavior policies and a novel GNN actor-critic architecture is proposed that is invariant to the number and order of vehicles. A CBPE is introduced to gain introspection into learned behavior policies at runtime and the encoded driving behavior within these. Finally, a post-optimization is introduced that uses learned behavior policies to generate initial estimates and constraints, includes interactions with other traffic participants, and provides safe and smooth trajectories.

In the subsequent sections of this chapter, an overview of behavior generation methods used in autonomous driving is given to understand the implications, advantages, and disadvantages of each behavior generation class. The works' research questions and contributions are presented, and, finally, a more detailed outline of the structure and the content of this thesis is laid out.

1.2 Behavior Generation Methods in Autonomous Driving

The choice of the behavior generation algorithm is essential concerning safety, comfort, and blending into mixed traffic. This section discusses the most popular solution methods for behavior generation using a taxonomy dividing these into search-, optimization-, and learning-based approaches. Before discussing these in detail, a brief problem formulation for a behavior generation problem for autonomous vehicles is given.

Each vehicle v_i in a scene is controlled by a behavior policy π_i that returns an action a_t for a given world state s_t at time t. The behavior policy can either be deterministic or probabilistic and is expressed as $\pi_i(a_t|s_t)$. By iteratively calling the behavior policies of all vehicles simultaneously a state-space trajectory $\tau_i(t) : [t_{start}, t_{start} + T] \rightarrow \mathcal{X}_{occupied}$ for each vehicle is obtained with T being the time-horizon of the trajectory. As the behavior policies π_i influence each other, the behavior generation is, henceforth, called *interactive behavior generation*. The

configuration space $\mathcal{X}$ is comprised of the initial configuration space $\mathcal{X}_{init}$ and of the occupied configuration space $\mathcal{X}_{occupied}$. The free configuration space is then given by $\mathcal{X}_{free} \leftarrow \mathcal{X} \notin \mathcal{X}_{occupied}$. Additionally, there exists a goal configuration space $\mathcal{X}_{goal}$ the vehicle tries to reach within the specified time horizon T. For the resulting ego vehicle's trajectory τ_{ego} to be executable, the trajectory needs to adhere to the ego vehicle's dynamic constraints. The vehicle's dynamics can either be included using a constraint function $D(\tau, \dot{\tau}, \ddot{\tau})$ that constrains the differential values of the trajectory or by directly forward integrating a dynamic vehicle model, e.g., using time-marching integration methods. Using the above-introduced terminologies the optimal behavior generation problem for the ego vehicle can then be formulated as

$$
\begin{aligned}
&\underset{\pi \in \Pi(\mathcal{X},T)}{\arg\min}\ f_0(\tau_{ego}(\pi)) && && (1.1)\\
&\text{subj. to } \tau(0) = \mathbf{x}_{\text{init}} \text{ and } \tau(T) \in \mathcal{X}_{\text{goal}} && &&\\
&\qquad \tau(t) \in \mathcal{X}_{\text{free}} && \forall t \in [0, \mathrm{T}] &\\
&\qquad D(\tau, \dot{\tau}, \ddot{\tau}) && \forall t \in [0, \mathrm{T}] &
\end{aligned}
$$

that tries to find a policy π from a set of policies $\Pi(\mathcal{X}, T)$ that minimizes the objective function $f_0(\tau_{ego}(\pi))$ and adheres to the constraints.

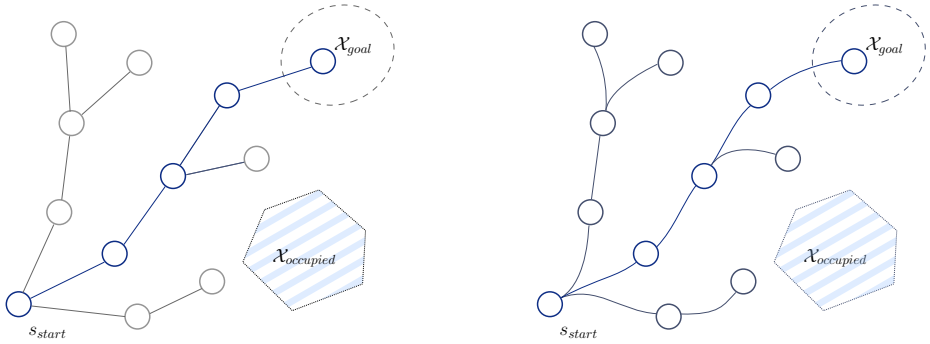
The problem formulation in Equation 1.1 is denoted as an optimization-based formulation having equality and inequality constraints. However, it is by no means restricted to optimization-based approaches and also applies to search- and learning-based methodologies. The problem in Equation 1.1 is an NP-hard problem that tends to scale poorly with an increasing number of vehicles and combinatorial options in general [83].

Low collision probabilities, the problems' complexity, the computational requirements, and other criteria restrict the usage of behavior generation methods in safety-critical applications, such as autonomous driving, significantly. In the following, each behavior generation class will be discussed in respect to the problem formulation in Equation 1.1.

1.2.1 Search-based

Search-based methods discretize the configuration space $\mathcal{X}$ and represent their solution in either graph or tree structures. These can be subdivided into probabilistic and deterministic methods. In safety-critical applications, such as autonomous driving, probabilistically exploring methods cannot guarantee low collision probabilities as stated in Section 1.1. However, in praxis, these have empirically shown to work well and have been applied for behavior generation of autonomous vehicles but might fall short in an assurance case. Table 1.1 provides an overview of the most popular search-based methodologies applied to autonomous driving.

Visibility graphs connect all points that are visible to each other. However, this results in a quadratic runtime complexity which makes these not applicable to complex problems. For simple problems, where the shortest path between two points



(a) rapidly-exploring random tree (RRT) **(b)** rapidly-exploring random tree* (RRT*)

Figure 1.2: Search-based behavior generation methods.

is required, these might pose a feasible choice. Visibility graphs cannot provide an optimal solution to the problem formulation in Equation 1.1 as no vehicle dynamics are considered and no objective function can be defined.

Lattices explore the configuration space $\mathcal{X}$ deterministically and uniformly and avoid dense cluttering around the root vertices [83]. Pivtoraiko et al. [88] use the term "state lattice" to describe a sampled configuration space $\mathcal{X}$. As the lattice states have to be connected, these need to be connectable, e.g., by formulating a two-point boundary problem. The discretization of the configuration and action space might prohibit finding a truly optimal solution for Equation 1.1, but avoids relying on stochastic sampling, which might make these methods more feasible in an assurance case for autonomous driving.

The RRT algorithm has initially been applied to solve motion planning problems in robotics [110]. Compared to many other search-based methodologies, it explores the configuration space more efficiently due to its Voronoi bias. However, a drawback in using the RRT algorithm is that it does not provide an optimal solution for the behavior generation problem in Equation 1.1.

The RRT* was then introduced that is capable of providing asymptotically optimal solutions for Equation 1.1 [55]. Two main mechanisms achieve the asymptotically optimal solutions: Finding better parent nodes and rewiring the search tree. However, when using a vehicle model, the rewiring is non-trivial as this poses a two-point boundary value problem optimization problem. Depending on the used solution method and not having an analytical steering function, the complexity might be far worse than $O(n \log n)$.

Similar to the above-stated algorithms, PRMs stochastically explore the configuration space $\mathcal{X}$. After the graph has been built, it is queried to find a solution to the motion planning problem. An extension, the probabilistic road-maps* (PRM*) exists capable of providing asymptotically optimal solutions. However, it has the

	Model Assumptions	Anytime	Complexity	Optimality
Deterministic				
Visibility Graph [23]	2D polygonal config. space	No	$O(n^2)$	No
Lattice and Dijkstra Pivtoraiko et al. [88]	Any with steering method	No	$O(n \log n)$	No
Probabilistic				
RRT [110]	Any	Yes	$O(n \log n)$	No
RRT* [55]	Any with steering method	Yes	$O(n \log n)$	Yes†
PRM [55]	Any with steering method	No	$O(n^2)$	No
PRM* [55]	Any with steering method	No	$O(n \log n)$	Yes†
MCTS [68]	Any	Yes	O(mk)	Yes‡

Table 1.1: Overview of search-based methods for behavior generation in autonomous driving. †: Solutions are asymptotically optimal. ‡: Optimal in respect to the Markov decision process. n: number of expansions during the search. m: number of children. k: simulation steps

same disadvantages as the RRT* as it needs to solve two-point boundary problems. A search tree spanned by the PRM algorithm is shown in Figure 1.3.

The Monte-Carlo tree search (MCTS) has recently gained popularity following the publications of AlphaGo and AlphaGo Zero [99, 102]. In contrast to the before stated methodologies, the underlying problem formulation of an MCTS is an MDP, of which the MCTS tries to maximize the future expected cumulative reward. However, the MCTS also relies on random sampling, which makes it challenging to guarantee low collision probabilities.

In summary, search-based methods are a powerful tool for generating behaviors for autonomous vehicles. They can include the ego vehicle's dynamics, solve multi-agent problem formulations, and provide asymptotically optimal solutions. However, due to the sample and problem complexity, they often cannot produce solutions in real-time or, in the stochastic case, guarantee to find a solution in bounded runtime. These characteristics might hinder their usage as behavior generation methods for autonomous vehicles as these require real-time operation and have to undergo a system assurance case that proves these to be safe.

1.2.2 Optimization-based

Optimization-based methods can deliver truly optimal solutions for the problem formulation in Equation 1.1. Depending on the solution method and problem formulation, the solution can either be globally or locally optimal using either global or local optimization, respectively. Table 1.2 provides an overview of optimization-

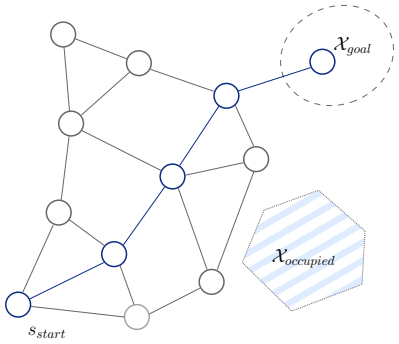


Figure 1.3: Graph spanned by the PRM algorithm.

based approaches applied to autonomous driving and in Figure 1.4 an exemplary constrained optimization problem is shown.

Global optimization methods are computationally more demanding as they find the global optimum instead of a local one and often cannot be applied in real-time.

Esterle et al. [31] propose a mixed integer quadratic programming (MIQP) approach that finds globally optimal solutions using a linearized dynamic model of the ego vehicle. However, as with many MIQP approaches, the runtime complexity increases exponentially with the number of objects, making these approaches in general not applicable to complex scenarios in real-time.

Frese et al. [32] propose a mixed integer linear programming (MILP) formulation using a double integrator as vehicle model. They assure the non-holonomy by bounding the lateral acceleration using an approximation that is valid for small yaw angles. Their approach can produce globally optimal solutions but is not valid for real-world scenarios having high curvatures, such as roundabouts.

On the contrary, local optimization-methods trade in finding the global optimum for performance. Thus, these tend to be relatively performant and can be applied in real-time applications. Under certain circumstances (the objective function being convex and the constraints being affine), the global optimum collapses to a local one, in which case local optimization methods also provide the global optimum.

Ziegler et al. [132] propose a local and continuous optimization method for planning trajectories. They use differential values of the trajectory to calculate the jerk and account for non-holonomy by bounding the trajectories' curvature. The resulting optimization problem is non-linear and is then solved using a sequential quadratic program (SQP) to find a locally optimal solution. A good initialization is required where a well-performing maneuver variant is chosen as the initial estimate for the optimization to avoid poor performance [11]. However, geometric partitioning and optimization do not scale well with an increasing number of vehicles. The approach has been shown in the real-world driving the original Bertha-Benz route fully autonomously [133].

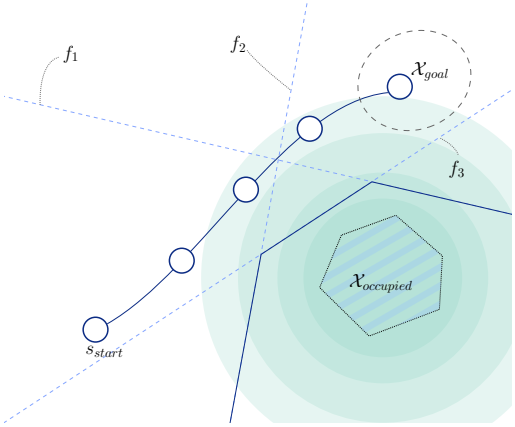


Figure 1.4: Trajectory optimization problem formulation showing the contours of the obstacle and having inequality constraints depicted by f_1, f_2, f_3 .

Gutjahr et al. [42] formulate two quadratic programs (QPs) that are solved sequentially – one for the longitudinal and one for the lateral Frenet coordinates. They additionally use a Frenet coordinate bicycle vehicle model to account for the non-holonomy of the vehicle. All obstacles have to be transformed into the Frenet-Coordinate space, which is a costly operation. Their approach has similar drawbacks as [132] as the initial estimate has a significant impact on the resulting solution.

Nilsson et al. [82] utilize two QPs for the longitudinal and lateral optimization using a linear double integrator model. They constrain the optimization using a coupling between lateral and longitudinal velocity to account for the non-holonomy of the vehicle that is valid for small yaw angles. These constraints, however, are not valid in scenarios having high curvatures, such as roundabouts.

In summary, optimization-based approaches are the only class of solution methods that can provide truly optimal solutions for Equation 1.1. At convergence, well-formulated optimization problems provide safe and comfortable behaviors. However, global optimization can be computationally expensive, and local optimization requires a "good" initialization. This work introduces a post-optimization that smoothens learned behavior trajectories whilst adhering to extracted constraints, which is built upon [46]. Combining learning- and optimization-based approaches generates synergies, as the learned behavior policy provides an initial solution and, thus, implicitly chooses a maneuver variant. In Section 4.3, a more theoretical perspective on trajectory optimization is provided.

1.2.3 Learning-based

Fueled by recent breakthroughs, such as AlphaGo [99] and AlphaGo Zero [102], as well as due to open-sourcing machine learning frameworks, such as TensorFlow[1] and

[1] `https://github.com/tensorflow/tensorflow`

	Model Assumptions	Problem Formulation	Optimality
Global Optimization			
Esterle et al. [31]	linearized bicycle model	MIQP	Yes
Frese et al. [32]	double integrator	MILP	Yes
Local Optimization			
Ziegler et al. [132]	triple integrator	SQP	Yes
Gutjahr et al. [42]	Frenet bicycle model	QP	Yes
Nilsson et al. [82]	double integrator	QP	Yes

Table 1.2: Overview of optimization-based methods for behavior generation in autonomous driving.

PyTorch[2], machine learning is gaining popularity and usage in behavior generation for autonomous vehicles. As stated in Section 1.1, these methods are capable of learning from data and experiences and can adapt their driving behavior to blend into traffic. Furthermore, many machine learning approaches do not require explicit formulations of the environment's dynamics. Additionally, these methods often outperform conventional algorithms in online performance as these can harness offline computational power. A taxonomy dividing machine learning methodologies into four classes is shown in Table 1.3. These are discussed in terms of their benefits and usage in behavior generation for autonomous vehicles.

Unsupervised learning (USL) learns only by using the input space and does not require any form of supervision. USL methods are not directly utilized for behavior generation but are very common in pre-processing steps, such as autoencoders to generate more efficient input representations [50].

Supervised learning (SL) maps an input to an output space that is directly defined by labeled data. A loss function is minimized, such as the cross-entropy or squared loss between the network's predicted output and target labels. Ross et al. [92] introduced a supervised learning approach called the DAgger framework that uses an expert's policy to gather data and train a policy that mimics the expert.

Imitation learning (IL) mimics expert behavior without access to an explicit reward signal [70]. In its simplest form, IL collapses to a SL problem, but usually more sophisticated approaches are used, such as generative adversarial networks (GANs). Li et al. [70] introduce an imitation learning approach called info generative adversarial imitation learnings (InfoGAILs) where a generator creates driving behaviors and the discriminator tries to distinguish between actual- and generated-driving behaviors. A significant drawback of GAN approaches is that these can suffer from mode collapse, in which case the performance deteriorates as the generator and the discriminator become stuck in a local minimum [106].

RL maximizes the future cumulative reward of an MDP merely using a reward signal. RL methodologies can be divided into model-based and model-free reinforce-

[2] `https://github.com/pytorch/pytorch`

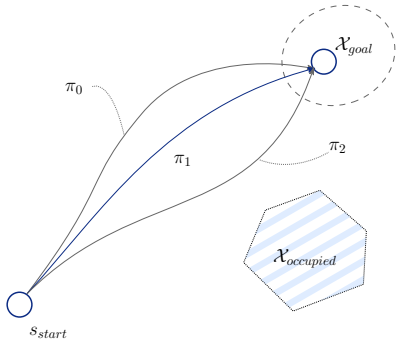


Figure 1.5: Multiple policies that reach the goal configuration space. The likelihood of "good" policies is iteratively increased using RL.

ment learning. Model-free reinforcement learning is ideal for problems in which the environment's dynamics are not known, making these approaches compelling for applications, such as autonomous driving. They learn their behavior merely requiring a reward signal and an input representation of the environment. These characteristics make them ideal in applications where the ego vehicle, e.g., has to nudge slightly onto the other vehicle's lane to slow it down so that it can merge. A detailed discussion of RL algorithms, their benefits, and drawbacks is given in Chapter 2.

In summary, learning-based approaches pose a powerful methodology that enable harnessing offline computational power, is data-driven, and that can learn game-theoretic and interactive behavior policies.

Methodology	Description
Unsupervised learning	Does not require labeling or external reward signals. Thus, does not require any form of supervision.
Supervised learning	Requires a target or label to map the input space to these.
Imitation learning	Learns from data to imitate certain styles or driving behaviors.
Reinforcement learning	Maximizes the future cumulative reward merely by using an external reward signal.

Table 1.3: Taxonomy dividing machine learning methodologies into four categories.

1.3 Research Questions and Contributions

The research questions tackled in this thesis are discussed and the individual contributions of this thesis are highlighted.

Research question 1 (RQ1): *How to learn continuous and invariant behavior policies in semantic environments?*

This thesis proposes, investigates, and benchmarks several input representations and reward signals for learning behavior policies in semantic environments for MDP settings. Solution methods for learning behavior policies using the MDP formulation are outlined and discussed. As the number and order of vehicles in traffic can change rapidly, used methodologies should be invariant to these. A novel GNN architecture for actor-critic reinforcement learning is introduced that is invariant towards the number and order of vehicles, that outperforms conventional networks, generalizes better, and has a smaller number of total network parameters as shown in [45]. Further, using GNNs the information flow in the behavior policies can be evaluated to gain insights into the learned behavior policy.

Research question 2 (RQ2): *How can the performance of learned behavior policies be evaluated at runtime?*

Neural networks often do not generalize or handle distributional shifts well. Even small changes in the input space can lead to profound changes in the output space that could potentially cause malicious behavior. For these to be deployed in safety-critical applications, runtime safety assurance (RTSA) can be used to estimate the behavior policies' performance. This thesis proposes a CBPE that is based on [44]. It uses counterfactual worlds that are derived from the actual world to evaluate the behavior policy at runtime. In these counterfactual worlds, non-factual behaviors are assigned to the other vehicles, and counterfactuals can be asked and answered, such as "Would the behavior policy be collision-free if the other vehicle had changed lanes?". It is shown that restricting the behavior policies' usage based on their performances over the counterfactual worlds significantly increases the success rate.

Research question 3 (RQ3): *How to obtain smooth behavior trajectories for learned behavior policies that utilize neural networks?*

DNNs can potentially produce non-smooth behaviors due to generalization issues or nonlinearities within these. Further, due to not fully exploring the configuration space the network might not be capable of handling all scenarios. This thesis proposes a post-optimization that smoothens the learned behavior trajectory whilst adhering to extracted constraints that is based on the work presented in [46]. By enforcing the optimized trajectory to be close to the initial, learning-based trajectory, the interactions with other vehicles remain valid. The post-optimized behavior trajectories offer the same safety guarantees and increased comfort while the interactions remain valid.

1.4 Outline

This section provides an outline and structure of the thesis.

In **Chapter 2**, learning interactive behavior policies in semantic environments is outlined, ranging from the MDP problem formulation, over the underlying theory, to the solution classes. The input representation for deep learning in semantic

environments is discussed in terms of its efficiency, transferability, and other criteria. Next, the reward signal and shaping for MDPs is discussed, and a novel potential-based reward shaping formulation for dynamic environments is presented. A particular focus is set on actor-critic (AC) RL methodologies and learning behavior policies for autonomous driving. Further, a novel GNN architecture is proposed that has been shown to outperform and generalize better than conventional DNNs. The end of the chapter provides a summary and addresses open challenges.

In **Chapter 3**, evaluating learned behavior policies at runtime is outlined. An overview of verification and validation methods is given, and the state of the art is outlined. Next, the usage of learned behavior policies in safety-critical applications, such as autonomous driving, is discussed, and several methodologies are being outlined – ranging from safe reinforcement learning (SRL), over RTSA, to methods combing conventional with learning-based methods. Finally, the CBPE is introduced that uses counterfactual worlds in which other vehicles behave non-factual. The CBPE enables evaluating the generalization capabilities of learned behavior policies at runtime and how these can, e.g., cope with distributional shifts.

In **Chapter 4**, an overview of optimization theory and its solution methods is given. Constrained and trajectory optimization theory is outlined. A theoretical view on trajectory optimization is given – the used vehicle models, numerical methods, problem formulation, and solution methods. A post-optimization built upon the presented theory is introduced that utilizes learned behavior policies to generate initial estimates and derive state constraints. The post-optimization allows for utilizing local optimization as the learned behavior policy provides a maneuver variant and warm-starts the optimization.

In **Chapter 5**, the simulation and benchmarking frameworks BARK and Behavior benchmARK - Machine Learning (BARK-ML) are introduced and the scenarios used for learning and evaluation are described. An architecture and hyperparameter search is performed for the conventional DNN and novel GNN architectures to find well-performing policies. The proposed potential-based reward shaping functions are evaluated in terms of their effectiveness. The information flow in the GNN architecture is visualized using the magnitude of the edge values of the GNN. Variational studies are performed for the learned behavior policies to conduct studies on their generalization capabilities. The CBPE is evaluated in terms of the insights it provides and how it could be used as a runtime monitor in, e.g., Simplex architectures. Influences are extracted, and how the behavior policy copes with model deviations at runtime is evaluated. Results of the post-optimization are presented that offer the same safety guarantee but additionally increase comfort.

Finally, in **Chapter 6**, a conclusion, discussion, and future work are given with the advantages and shortcomings of the proposed methodologies.

2 Learning Behavior Policies in Semantic Environments

This chapter focuses on learning behavior policies for autonomous vehicles in semantic and uncertain environments. A *semantic environment* is defined as one in which all traffic participants are represented in an object list, and information, such as maps, is represented as vectorial information. In uncertain environments, the formulation of, e.g., an objective function is non-trivial due to the unknown dynamics and transition probabilities of objects in the environment. Thus, conventional methods usually require extensive parameter-tuning to blend into these. Learning-based methods can explore the configuration space and learn behavior policies from experience or using a data-set. This chapter formulates the trajectory planning problem provided in Equation 1.1 as an Markov decision process (MDP) to find a sequential decision sequence that maximizes the cumulative future reward.

Section 2.1 outlines the underlying theory of MDPs and solution methods ranging from dynamic programming, over Monte Carlo methods, up to actor-critic reinforcement learning for learning behavior policies.

Section 2.2 discusses input representations for MDPs in semantic environments and how these can be represented for the usage with deep neural networks (DNNs). The efficiency of these input representations, their transferability, and other criteria are discussed and elaborated.

The reward signal defined in the MDP plays a crucial role in the solution methods' performance during learning and in the resulting behavior policy. Section 2.3 discusses reward signals for learning behavior policies for autonomous vehicles. Several potential-based reward shaping functions for learning behavior policies are proposed to avoid the *credit assignment problem*.

Finally, to overcome the shortcomings of, e.g., being dependent on the number and order of vehicles, a novel graph neural network (GNN) actor-critic architecture is proposed. An overview of GNNs is given and the unified framework proposed by Battaglia et al. [9] is outlined. The novel GNN actor-critic architecture based on the unified framework for learning behavior policies is then introduced in detail.

2.1 Underlying Theory: Markov Decision Process

Markov decision processes (MDPs) are a formalization of sequential decision-making problems, where actions influence future reward [111]. They provide a mathematical framework to study methods, such as dynamic programming (DP) and reinforcement

learning (RL). All solution methods have in common that they try to maximize the cumulative future expected reward.

The Markov assumption states that the history of all states can be neglected as it is represented in the current state s_t. A state s_t is Markovian if and only if

$$\mathbb{P}(s_{t+1}|s_t) = \mathbb{P}(s_{t+1}|s_1, \ldots, s_t) \tag{2.1}$$

with $\mathbb{P}$ being the state-transition probability transitioning from a Markov state s_t to a successor state s_{t+1}. Thus, it is assumed that the past information $s_1, \ldots, s_{t-1}$ is encoded in the present state s_t and does not depend on the history of states $s_{t-1}, \ldots, s_0$. For convenience of notation a short form of the transition probability is used:

$$\mathcal{P}_{s_t s_{t+1}} = \mathbb{P}(s_t|s_{t+1}) \tag{2.2}$$

MDPs are a straightforward framing of the problem of learning from interaction to achieve a specific goal [111]. In terms of decision-making, an MDP can be viewed as a state machine with transition probabilities and rewards. By optimizing actions and influencing the states' transitions, the future expected cumulative reward $v(s_t)$ can be maximized. An MDP is defined by a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$ where

- $\mathcal{S}$ is a (finite) set of states,
- $\mathcal{A}$ is a (finite) set of actions,
- $\mathcal{P}$ is a state transition probability matrix, $\mathcal{P}^a_{s_t s_{t+1}} = \mathbb{P}[s_{t+1}|s_t, a_t]$,
- $\mathcal{R}$ is a reward function, $\mathcal{R}^a_s = \mathbb{E}[R_{t+1}|s_t, a_t]$, and
- $\gamma \in [0, 1]$ is a discount factor.

A policy π that outputs an action in an MDP is defined as a distribution over actions conditioned on the state and is denoted as

$$\pi(a_t|s_t) = \mathbb{P}[a_t|s_t]. \tag{2.3}$$

Due to the Markov property, the policy $\pi(a_t|s_t)$ only depends on the current state s_t and, thus, is a stationary policy (time-independent) [60]. Having an MDP problem formulation and a policy π, the policies' actions can be learned to increase the future expected cumulative reward.

The state-value function v_π is defined as the expected return starting from state s_t and then following the policy π – in episodic environments until termination. The state-value function is given by

$$v_\pi(s_t) = \mathbb{E}_\pi[G_t|s_t]. \tag{2.4}$$

Another option is to define state-action-value function $q_\pi(s_t, a_t)$. It represents the expected return starting from state s_t, choosing an action a, and then following the policy π. Thus, the action-value function $q_\pi(s_t, a_t)$ is given by

$$q_\pi(a_t|s_t) = \mathbb{E}_\pi[G_t|s_t, a_t]. \tag{2.5}$$

Equation 2.4 and 2.5 can be decomposed into an immediate reward and the discounted value of the successor state. This then yields the Bellman expectation equations:

$$v_\pi(s_t) = \mathbb{E}[R_{t+1} + \gamma v_\pi(s_{t+1})|s_t, a_t \sim \pi] \tag{2.6}$$

$$q_\pi(s_t, a_t) = \mathbb{E}[R_{t+1} + \gamma q_\pi(s_{t+1}, a_{t+1})|s_t, a_t \sim \pi]. \tag{2.7}$$

In the following, the primary solution methodologies for MDPs are discussed: dynamic programming, sampled-based methods, and reinforcement learning.

2.1.1 Solution Method: Dynamic Programming

DP refers to a collection of algorithms that compute optimal policies given a perfect model of the environment as a MDP [111]. Methods in dynamic programming mainly use one or both of the following steps: policy evaluation and policy improvement.

Policy Evaluation: In the policy evaluation, a state-value function v_π for an arbitrary policy π is computed that evaluates the policy π and the expected cumulative discounted reward. The Bellman equations defined in the previous section are used to compute the state-value function v_π. The state-value function v_π for an arbitrary policy π is given by

$$v_\pi(s_t) = \sum_a \pi(a_t|s_t) \sum_{s_{t+1} \in \mathcal{S}_{t+1}} \mathbb{P}^a_{s_t s_{t+1}}[\mathcal{R}_{s_t} + \gamma v_\pi(s_{t+1})]. \tag{2.8}$$

As there is no closed-form solution, iterative solutions have to be utilized [111]. In the iterative policy evaluation, Equation 2.8 is iteratively used as an update rule as follows:

$$v_\pi(s) \leftarrow \sum_a \pi(a_t|s_t) \sum_{s'} \mathbb{P}^a_{ss'}[\mathcal{R}_s + \gamma v_\pi(s')]. \tag{2.9}$$

There are two main ways that these updates can be achieved: there can be two value functions – one for the old values and one for the new values or a single one can be used that is updated in a sweeping manner. As discussed in [111], both of these variants will eventually converge to the optimal solution. This step is often referred to as the E-step as it evaluates the current policy.

If the state-action-value $q_\pi(s_t, a_t)$ of Equation 2.7 is greater than the (baseline) state-value function $v_\pi(s_t)$, choosing the action a_t is more benefitial than following the policy π.

Policy improvement theorem: Given two determinisic policies π and π' such that, for all $s \in \mathcal{S}$,

$$q_\pi(s, \pi'(s_t)) \geq v_\pi(s_t). \tag{2.10}$$

Algorithm 1 Policy Iteration [111]

1. Initialization
$V(s) \in \mathbb{R}$ and $\pi(s) \in \mathcal{A}(s)$ for all $s \in \mathcal{S}$

2. Policy Evaluation
repeat
  $\Delta \leftarrow 0$
  for all $s \in \mathcal{S}$ **do**
    $v \leftarrow V(s)$
    $V(s) \leftarrow \sum_{s' \in \mathcal{S}_{t+1}} \mathbb{P}_{ss'}^{a \sim \pi(s)} [\mathcal{R}_s + \gamma V(s')]$
    $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
  end for
until $\Delta < \epsilon$

3. Policy Improvement
policy-stable $\leftarrow$ *true*
for all $s \in \mathcal{S}$ **do**
  old-action $\leftarrow \pi(s)$
  $\pi(s) \leftarrow \text{argmax}_a \sum_{s' \in \mathcal{S}_{t+1}} \mathbb{P}_{ss'}^{a \sim \pi(s)} [\mathcal{R}_s + \gamma V(s')]$
  if *old-action* $\neq \pi(s)$ **then**
    policy-stable $\leftarrow false$
  end if
end for
if *policy-stable* **then**
  return $V(s), \pi$
else
  go to 2. Policy Evaluation
end if

then the policy π' must be as good as, or better than π. The detailed proof of Equation 2.10 is provided in [111]. To find the greedy policy π' in all states all actions have to be evaluated, such that

$$\pi'(s_t) \leftarrow \underset{a_t}{\text{argmax}}\ q_\pi(s_t, a_t) \tag{2.11}$$

$$\leftarrow \underset{a_t}{\text{argmax}}\ \mathbb{E}[\mathcal{R}_{t+1} + \gamma v_\pi(s_{t+1}) | s_t, a_t] \tag{2.12}$$

$$\leftarrow \underset{a_t}{\text{argmax}} \sum_{s_{t+1} \in S_{t+1}} \mathbb{P}_{s_t s_{t+1}}^{a_t} (\mathcal{R}_{s_t} + \gamma v_\pi(s_{t+1})). \tag{2.13}$$

The greedy policy in Equation 2.13 meets the conditions of the policy improvement theorem defined in Equation 2.10. The process of improving the original policy by making it greedy to the original value function is called *policy improvement* [111].

This step is also referred to as the M-step as it maximizes the expected cumulative future reward.

Policy Iteration: The policy iteration alternates between the two above-introduced E- and M-steps to find an optimal policy π^* for an MDP. The policy iteration is guaranteed to provide monotonically improving policies and value functions [111]. The schemata of the policy iteration can be depicted as follows:

$$v_\pi(s) \rightarrow \pi(s) \rightarrow v_\pi(s) \rightarrow \dots \tag{2.14}$$

The policy iteration method is outlined in Algorithm 1. One of the drawbacks of Algorithm 1 is that the transition probability $\mathcal{P}_{ss'}^{a\sim\pi(s)}$ has to be known. Additionally, when using a continuous action space, it is non-trivial to compute the argmax as there are infinite many actions. Many state-of-the-art reinforcement learning algorithms are based on these two steps (E & M).

In summary, DP are a powerful solution method when the transition probabilities are fully known in the environment. In traffic, however, the transition probabilities of the environment are often not known. Thus, sample-based methods are a more promising candidate for learning behavior policies for autonomous vehicles. These are discussed in the following section.

2.1.2 Solution Method: Sample-based Methods

Sample-based methods solve MDPs only by using samples (experiences). Contrary to DP, sample-based methods do not require full knowledge about the environment's dynamics and its transition probabilities $\mathcal{P}$. This section outlines two sample-based MDP solution methods: Monte Carlo methods (MCMs) and temporal differences (TDs).

Monte Carlo Methods: The here discussed MCMs are purely episodic – the state-value function $v(s_t)$, action-state-value function $q(s_t, a_t)$, and the behavior policy are only updated after one or multiple episodes. As in dynamic programming, there are two main steps in MCMs – a policy evaluation and policy improvement step.

In the policy evaluation step, two main methods are utilized – the first-visit and the every-visit Monte Carlo prediction methods [111]. The policy evaluation differs mainly from the one introduced in the DP methods as there are no max operator and transition probabilities used. Instead, Monte Carlo simulation is utilized by simulating many episodes and computing the state-value function $v(s_t)$ based on these.

The policy improvement step is similar to the one presented in the DP methods. The actions are still greedily choosen in respect to the esimated state-value function $v(s_t)$. As in dynamic programming, the policy improvement theorem given in Equation 2.10 applies.

One method to implement Monte Carlo policy iteration is the Monte Carlo exploring starts approach [111]. MCMs are an often recurring concept in reinforcement learning, especially in model-free actor-critic reinforcement learning.

Temporal Differences: One of the underlying key ideas central and novel to reinforcement learning is TD learning [111]. TD learning combines the ideas of MCM and DP. As TD learning is also sample-based, TD methods do not require knowledge about the environment's dynamics. The relationship between TD, DP and MCM methods is a recurring theme in the theory of reinforcement learning. Contrary to MCM, TD is an online method and does not have to be episodic. Using TD, the update of the state-value function $V(s_t)$ is given by

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

and, thus, does not require the environment to be episodic as the update just depends on the current state s_t and the next state s_{t+1}. TD methods are not restricted to only using a single time-step estimate but can utilize multiple. This is denoted as TD(N) with N being the states in the future that are being used for the update. The TD(0) equation can be formulated as

$$v_{t+1}(s_t) = v_{t+1}(s_t) + \alpha_t[\overbrace{\underbrace{R_{t+1} + \gamma v_t(s_{t+1}|s,\pi)}_{\text{TD}_{target}} - v_{t+1}(s_t)}^{\text{TD}_{error}}]. \tag{2.16}$$

Contrary to MCM, TD explicitly exploits the Markov property and is, therefore, more sample efficient in Markovian environments. The idea of TD methods forms an important basis of many state-of-the-art reinforcement learning algorithms.

In summary, sample-based methods do not require the transition probabilities of the environment, which makes them applicable to learn behaviors in traffic where the other vehicle's behaviors are unknown (e.g., mixed traffic). The next section discusses RL and in particular policy-based RL that learns the policy directly without the need of a state-value function.

2.1.3 Solution Method: Reinforcement Learning

RL is simultaneously a problem formulation, a class of solution method, and the field that studies this problem and its solution methods [111]. It is strongly based on the ideas presented in the previous sections – in particular on sample-based methods. Unlike dynamic programming, model-free RL does not require a model of the environment's dynamics and learns how to act in an environment only using a reward signal.

Value-based RL methods use state-value or state-action-value functions to find a solution for an MDP. Value-based RL is strongly interlinked with TD learning. One of the earliest and simpler on-policy value-based RL methods is the state action reward state action (SARSA) (State s, Action a, Reward r, next State s', and next Action a) algorithm [93]. It collects experiences in the form of tuples $\langle s, a, r, s', a\rangle$. These tuples are then used to directly update the state-action value function:

$$Q(s,a) \leftarrow Q(s,a) + \alpha(r + \gamma Q(s',a') - Q(s,a)). \tag{2.17}$$

For smaller, discrete state-space problems, the SARSA algorithm can be tabular-based and for larger ones deep neural networks (DNNs) can be used as function approximators. The exploration in SARSA can be guided using an ϵ-greedy exploration strategy.

Q-learning is an off-policy value-based method similar to SARSA [123]. In Q-learning a similar formulation as in Equation 2.17 is used that is defined by

$$Q(s,a) \leftarrow Q(s,a) + \alpha(r + \gamma \max_{a'} Q(s',a') - Q(s,a)). \tag{2.18}$$

The only difference to Equation 2.17 is the additional max operator. The learned state-action-value function $Q(s,a)$ directly approximates $Q^*(s,a)$ independent of the policy being followed [111]. Q-learning is also an off-policy algorithm as it does not require the policy to explore the configuration space. As in SARSA, Q-learning can explore the environment using the policy and ϵ-greedy exploration. When using a tabular-based implementation, Q-learning converges to an optimal policy $Q^*(s,a)$ [111].

Mnih et al. [78] presented a deep Q-network (DQN) algorithm that is capable of learning directly using high-dimensional inputs. They applied DQN reinforcement learning for solving Atari games and were able to outperform human players. In the presented work, they combined convolutional neural networks and Q-learning to learn on high-dimensional inputs.

Schaul et al. [94] additionally introduced a prioritized replay buffer to Q-learning. It samples transitions from the replay buffer proportional to the last encountered absolute TD error. This makes learning rare events more feasible, such as when using a binary reward signal for reaching the goal.

Another problem with conventional deep DQN methods is that these tend to overestimate the state-value function due to the max operator. Double deep Q-learning (DDQN) addresses this overestimation by decoupling the policy evaluation and improvement step [115]. DDQN was first introduced in a tabular fashion but also works with any kind of function approximator, such as DNNs. Not only did the DDQN algorithm reduce the overestimation, but it also led to an overall improvement in performance.

Bellemare et al. [10] proposed a method called distributional RL where they learn distributions of the expected return instead of the expected return. This led to a further improvement in performance over conventional DDQN approaches.

A method combining all of the above-stated methodologies and further extensions has been presented by Hessel et al. [48]. Their proposed method achieves state-of-the-art performance and is superior to any of the above-stated methods individually.

Policy-based RL methods do not rely on first approximating a state-value or state-action-value function but instead directly optimize a (stochastic) policy. Therefore, the state-value and state-action-value function are not strictly required for these methods. The policy formulation given in Equation 2.3 is extended to

$$\pi(a_t|s_t,\theta) = \mathbb{P}(a_t|s_t,\theta_t) \tag{2.19}$$

and is now additionally dependent on $\theta \in \mathbb{R}^d$ that are the DNN parameters. In policy-based methods, the utility function $U(\theta)$ defined as

$$U(\theta) = \mathbb{E}[\sum_{t=0}^{H} R(s_t, a_t); \pi_\theta] = \sum_{\tau} P(\tau; \theta) R(\tau) \tag{2.20}$$

is directly optimized. When using a neural network parameterized by θ, the gradient of the utility function $U(\theta)$ can directly be taken with respect to the network parameters θ. Thus, the likelihood ratio policy gradient can be derived as follows

$$\begin{aligned}
\nabla_\theta U(\theta) &= \nabla_\theta \sum_{\tau} P(\tau; \theta) R(\tau) && (2.21)\\
&= \sum_{\tau} \nabla_\theta P(\tau; \theta) R(\tau) && (2.22)\\
&= \sum_{\tau} \frac{P(\tau; \theta)}{P(\tau; \theta)} \nabla_\theta P(\tau; \theta) R(\tau) && (2.23)\\
&= \sum_{\tau} P(\tau; \theta) \frac{\nabla_\theta P(\tau; \theta)}{P(\tau; \theta)} R(\tau) && (2.24)\\
&= \sum_{\tau} P(\tau; \theta) \nabla_\theta \log P(\tau; \theta) R(\tau) && (2.25)
\end{aligned}$$

with $P(\tau; \theta)$ being the probability and $R(\tau)$ the reward of a trajectory τ. As there is no closed-form or analytical solution, Equation 2.25 has to be approximated by

$$U(\theta) \approx \hat{g} = \frac{1}{m} \sum_{i=1}^{m} \nabla_\theta \log P(\tau^i; \theta) R(\tau^i) \tag{2.26}$$

with m being the number of samples. This entails that the likelihood ratio changes the probabilities of experienced paths and not the paths themselves. The term $\nabla_\theta \log P(\tau^i; \theta)$ can further be extended using a dynamics model $P(s^i_{t+1}|s^i_t, a^i_t)$ and a policy $\pi_\theta(a^i_t|s^i_t)$:

$$\begin{aligned}
\nabla_\theta \log P(\tau^i; \theta) &= \nabla_\theta \log[\prod_{t=0}^{H} P(s^i_{t+1}|s^i_t, a^i_t) \pi_\theta(a^i_t|s^i_t)] && (2.27)\\
&= \nabla_\theta [\sum_{t=0}^{H} \log P(s^i_{t+1}|s^i_t, a^i_t) + \sum_{t=0}^{H} \log \pi_\theta(a^i_t|s^i_t)] && (2.28)\\
&= \sum_{t=0}^{H} \nabla_\theta \log \pi_\theta(a^i_t|s^i_t). && (2.29)
\end{aligned}$$

As the first term in Equation 2.28 does not depend on the neural network parameters θ taking the derivative yields zero. Combining Equation 2.26 and 2.29 yields the final likelihood ratio policy gradient equation:

$$\hat{g} = \frac{1}{m} \sum_{i=1}^{m} \nabla_\theta \log \pi_\theta(a^i_t|s^i_t) R(\tau^i). \tag{2.30}$$

As the dynamics model $P(s_{t+1}^i|s_t^i, a_t^i)$ is not required in Equation 2.30, policy-based methodologies derived this way are called *model-free* reinforcement learning. Model-free methods are referred to as being problem-agnostic as they do not require any knowledge about the environment's dynamics and can be applied to a wide range of problems without further modification.

In theory, Equation 2.30 can be applied to solve challenging tasks. However, using this equation, all actions would become more likely — also bad ones. To mitigate this issue, a bias in the form of a baseline can be introduced to Equation 2.30. This is valid, as introducing a baseline does not change the core properties of the optimization as long as it does not depend on the network's parameters θ directly. The modified equation using a baseline is given by

$$\hat{g} = \frac{1}{m} \sum_{i=1}^{m} \nabla_\theta \log \pi_\theta(a_t^i|s_t^i) \underbrace{(R(\tau^i) - b)}_{\text{Advantage } \hat{A}}. \tag{2.31}$$

The difference between the reward $R(\tau^i)$ and the baseline $b(s_t)$ is called the advantage $\hat{A}$. Generally speaking, actions that have a higher reward than the baseline are reinforced and other ones are made less likely. Although introducing a bias might introduce unwanted effects, this is overweight by the fact that introducing a bias also significantly reduces the variance. In general, lowering the variance is one of the core challenges in developing efficient and well-performing RL methods [43]. Another commonly used methodology to reduce variance in RL methods is to make use of the temporal structure of rewards. As the problem is assumed to be Markovian, future actions do not depend on the past states but only on the current state. Thus, Equation 2.31 can be split into two parts and modified so that it only takes future rewards into account:

$$\hat{g} = \frac{1}{m} \sum_{i=1}^{m} \left(\sum_{t=0}^{H-1} \nabla_\theta \log \pi_\theta(a_t^i|s_t^i) \right) \left(\sum_{t=0}^{H-1} R(\tau^i) - b(s_k^i) \right). \tag{2.32}$$

A basic policy gradient reinforcement learning method – the REINFORCE algorithm – is outlined in Algorithm 2. To design efficient likelihood-ratio policy gradients

Algorithm 2 REINFORCE [124]

Policy $\pi(a_t|s_t, \theta)$ and $\alpha > 0$
while Training **do**
  Generate an episode $S_0, A_0, R_1, \ldots, S_{t-1}, A_{t-1}, R_T \sim \pi(\cdot|\cdot, \theta)$
  for $t = 0, \ldots, T-1$ **do**
    $G \leftarrow \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k$
    $\theta \leftarrow \theta + \alpha G \ln \nabla \pi(A_t|S_t, \theta)$
  end for
end while

methods, the following desiderata have to be taken into account: A stable monotonic improvement and a good sample efficiency are required.

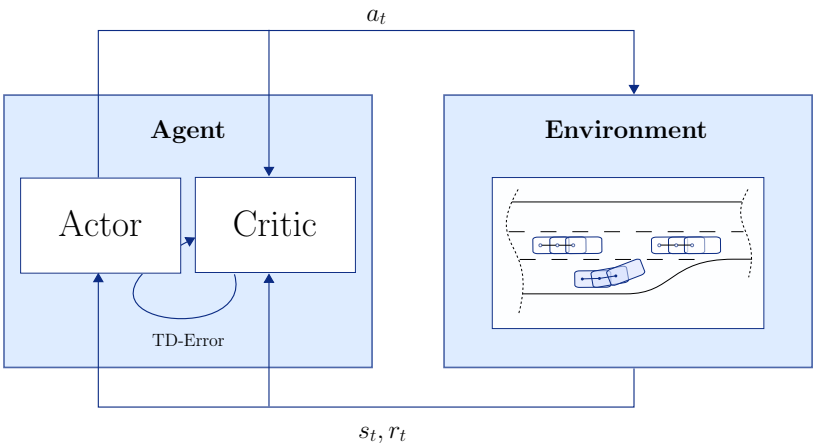


Figure 2.1: Learning cycle in actor-critic reinforcement learning.

The policy-based methods that have been discussed so far use the learned policy $\pi_\theta(a_t|s_t)$ to explore the configuration space. If a bad step is taken in the optimization, the algorithm then would explore the configuration space using the novel – possibly bad – policy. This can lead to a collapse in performance and the algorithm might not recover from such a bad step. Thus, the step size and a stable monotonic improvement are especially important. The other consideration when designing policy-based methods is their sample efficiency. This is often a major limitation for RL methods, as many of these require thousands or even millions of collected experiences.

Policy-based methodologies learn stochastic and continuous policies but often suffer from poor sample efficiency due to exploring on-policy possibly leading to degrading performance and high variances. Actor-critic reinforcement learning addresses some of these shortcomings and achieves state-of-the-art performance, which will be discussed in the next section in detail.

2.1.4 Solution Method: Actor-Critic Reinforcement Learning

Actor-critic (AC) RL is at the intersection of policy- and value-based reinforcement learning. AC RL methods directly optimize the policy, such as policy-based RL and a critic rates the actor's actions. This tends to reduce the variance and introduces a bias into the learning process. Reducing the variance and using a baseline enhances the performance significantly towards pure policy-based RL approaches. Often stochastic behavior policies are learned in AC RL that efficiently explore the configuration space using sampling. AC RL can learn stochastic and continuous behavior policies, which have shown state-of-the-art performance in various tasks [95, 96, 43, 2]. This section discusses AC RL characteristics and methodologies.

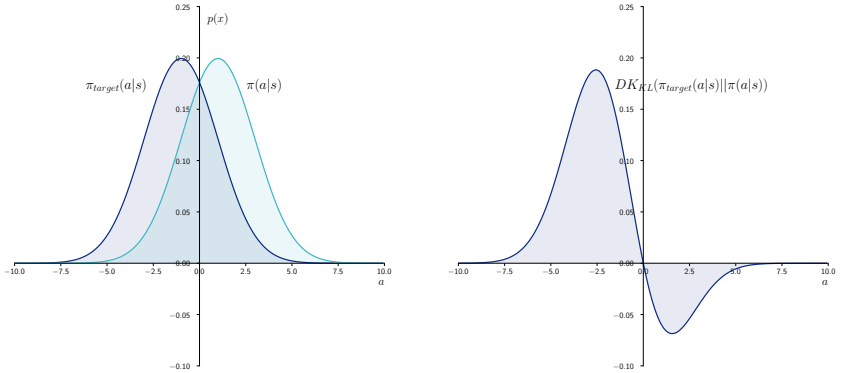


Figure 2.2: The left figure depicts a target policy π_{target} and the current policy π. The Kullback-Leiber (KL) divergence for these two distributions is plotted on the right.

Off-policy algorithms enable using a replay buffer and experiences and avoid poor policies leading to insufficient exploration. Therefore, off-policy algorithms tend to be more sample efficient as these can utilize experiences far from the past that on-policy methods cannot.

AC RL methods can use stochastic or deterministic policies. Stochastic policies offer a couple of advantages for game-theoretic environments having opponents. For example, only a stochastic policy can offer an optimal solution to the rock-paper-scissors game [65].

One of the fist AC RL methods that achieved state-of-the-art performance in continuous control tasks was the trust region policy optimization (TRPO) algorithm that then was followed by the proximal policy optimization (PPO) RL algorithm [95, 96]. Both of these methodologies make use of continuous and stochastic policies and learn on-policy. At the same time, the deep deterministic policy gradient (DDPG) and twin-delayed DDPG (TD3) algorithms were introduced that use deterministic policies inspiried by the DQN algorithm [71, 33]. These methods utilize external exploration processes as the policy cannot be sampled to explore the configuration space.

Ideally, one would combine the best characteristics of on- and off-policy AC RL algorithms. The soft actor critic (SAC) algorithm does just that by utilizing stochastic policies and still being able to learn off-policy and utilizing a replay buffer. Additionally, it replaces the exploration process by maximizing the future cumulative expected entropy in its objective function. Maximizing the entropy leads to finding the most random behavior policies that also maximize the future cumulative expected reward. In the following, *state-of-the-art* AC methods are outlined and discussed in greater detail.

The **TRPO** is an AC RL method that uses stochastic policies and explores the configuration space on-policy. The TRPO has the same monotonic improvement guarantees as the algorithm proposed by Kakade et al. [54]. Monotonic improvement is achieved by utilizing a commonly used concept in the optimization domain — a (safe) trust-region for limiting the policies' change [60]. As a measure for this trust-region, the KL divergence is used. The KL divergence is depicted in Figure 2.2 on the right and the policies using a Gaussian distribution on the left. The overall objective function of the TRPO can be formulated as a constrained optimization problem as

$$\text{maximize } L_{\theta_{old}}(\theta) \tag{2.33}$$

$$\text{subject to } D_{KL}^{max}(\theta_{old}, \theta) \leq \delta. \tag{2.34}$$

with $L_{\theta_{old}}(\theta)$ being a surrogate objective function and δ the trust-region radius. The surrogate objective function is defined as

$$L_{\theta_{old}}(\theta) = \mathbb{E}[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}\hat{A}_t] \tag{2.35}$$

that uses an importance ratio between the current policy π_θ and the old policy $\pi_{\theta_{old}}$. An implementation of the TRPO method is provided in Algorithm 3.

Algorithm 3 Trust-Region Policy Optimization [95]

for all iteration=$1, 2, \ldots$ **do**
  Run policy for T timesteps or N trajectories
  Estimate the advantages $\hat{A}$
  Compute the policy gradient g
  Compute the search direction $x_{sd} = F^{-1}g$ using the conjugate gradients
  Line-search along x_{sd} while satisfying the KL constraint
end for

To compute the search-direction x_{sd}, the inverse of the Fisher-information matrix F^{-1} multiplied by the gradient g is used. After obtaining the search direction, a line-searched in the search-direction x_{sd} is performed subject to the defined KL divergence trust region.

The **PPO** algorithm is based on the TRPO approach and restricts the policy update [96]. It has the same characteristics as the TRPO — using a stochastic policy, exploring on-policy, and having a continuous action space. Enforcing the KL constraint in the TRPO algorithm is computationally expensive. The PPO avoids this by introducing a surrogate objective function that restricts the policy update step using clipping. The main objective that is proposed in the PPO algorithm is defined as

$$L^{CLIP}(\theta) = \mathbb{E}[\min(r_t(\theta)\hat{A}, clip(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t)] \tag{2.36}$$

with ϵ being a tunable hyperparameter. To better understand the idea behind Equation 2.36, Figure 2.3 visualizes the clipping mechanism. The clipping function

restricts too large positive updates by L^{CLIP}. For the negative ones, these are only bound up to a reward value of $1 - \epsilon$ and then can decrease linearly. The full PPO algorithm is outlined in Algorithm 4.

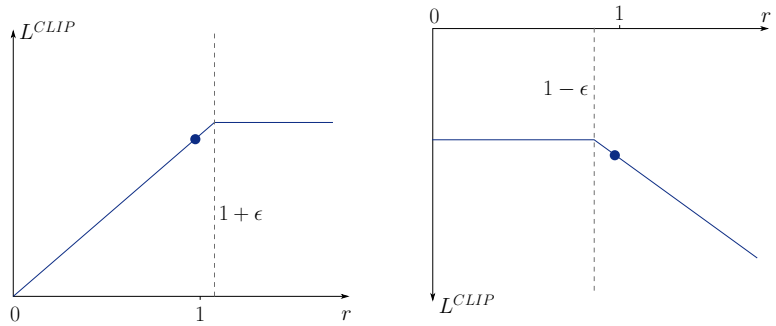


Figure 2.3: Surrogate objective function L^{CLIP} in the PPO as shown in [96].

Algorithm 4 Proximal Policy Optimization [96]

for all iteration=$1, 2, \ldots$ **do**
  for all actor=$1, 2, \ldots, N$ **do**
    Run policy $\pi_{\theta_{old}}$ for T timesteps
    Estimate the advantages $\hat{A}_1, \ldots, \hat{A}_T$
  end for
  Optimize surrogate L w.r.t. θ, with K epochs and minibatch size $M \leq NT$
  $\theta_{old} \leftarrow \theta$
end for

The PPO shows empirically better performance and requires less computational power than the TRPO [96].

The **DDPG** algorithm can be seen as an extension of the DQN algorithm to the continuous action domain [71]. It is an AC RL method that is model-free, uses a deterministic policy, and utilizes a continuous action-space. Contrary to the above-introduced algorithms – the TRPO and PPO algorithms – the DDPG algorithm can utilize a replay buffer and, thus, can make use of past experiences. Using a replay-buffer makes the DDPG algorithm more sample efficient and enables it to be trained offline using collected experiences. The DDPG algorithm is based on the deep policy gradient (DPG) algorithm introduced by Silver et al. [101]. The critic network $Q(s, a)$ is updated using the Bellman Equations 2.7 and the actor is updated using the chain rule as follows

$$\nabla_\theta J \approx \mathbb{E}_{s_t \sim \rho}[\nabla_a Q(s, a|\theta^Q)|_{s=s_t, a=\mu(s_t)} \nabla_{\theta^\mu} \mu(s, \theta^\mu)|_{s=s_t}] \tag{2.37}$$

with ρ being a random exploration process. One of the fundamental differences of the DDPG to the DPG algorithm is that it utilizes soft updates and target networks

for the critic network $Q(s, a)$ [71]. A soft update is achieved by changing the critic network slowly, by, e.g., using a linear interpolation

$$\theta' \leftarrow \tau\theta + (1 - \tau)\theta'. \tag{2.38}$$

that restricts the change of update of the policies and critic networks. The authors state that this was essential in achieving stable targets to avoid divergence of the critic [71]. The full DDPG algorithm is shown in Algorithm 5.

Algorithm 5 Deep Deterministic Policy Gradient [71]

Initialize θ^Q, θ^μ
$\overline{\theta}^Q \leftarrow \theta^Q, \overline{\theta}^\mu \leftarrow \theta^\mu$
$\mathcal{D} \leftarrow \varnothing$
for each iteration=$1, 2, \ldots$ **do**
  $a_t \sim \pi_\phi(a_t|s_t)$
  $s_{t+1} \sim p(s_{t+1}|s_t, a_t)$
  $\mathcal{D} \leftarrow \mathcal{D} \bigcup \{(s_t, a_t, r(s_t, a_t), s_{t+1})\}$
end for
for each iteration=$1, 2, \ldots$ **do**
  $\theta^Q \leftarrow \theta^Q - \nabla_{\theta^\mu} \frac{1}{N} \sum_i (r_i + \gamma \overline{\theta}^Q(s_{t+1}, \overline{\mu}(s_{t+1}|\overline{\theta}^\mu)) - Q(s_i, a_i))^2$
  $\nabla_{\theta^\mu} J \approx \frac{1}{N} \sum_i \nabla_a Q(s, a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^m u} \mu(s|\theta^\mu)|_{s_i}$
  $\overline{\theta}^Q \leftarrow \tau\theta^Q + (1 - \tau)\overline{\theta}^Q$
  $\overline{\theta}^\mu \leftarrow \tau\theta^\mu + (1 - \tau)\overline{\theta}^\mu$
end for
return θ_1, θ_2, ϕ

The **SAC** uses a stochastic policy and can either use discrete or continuous actions [43]. Like the TRPO and PPO, it utilizes stochastic policies but, contrary to these methods, can use off-policy experiences. Additionally, to maximizing the future cumulative expected reward, the future cumulative expected entropy is being maximized. Maximizing the entropy mitigates designing an external exploration process, as maximizing the future expected entropy provides a natural way for exploration. The entropy maximization can be seen as an incentive for the algorithm to learn policies that achieve high rewards while being as random as possible. Due to being off-policy and being able to utilize a replay-buffer, the SAC algorithm has a higher sample efficiency than the TRPO and PPO algorithms. It has been shown to outperform other off-policy methods that utilize replay buffers, such as the DDPG or TD3 methods [43]. Contrary to these methods, the SAC can utilize a stochastic policy that can be used in the exploration process by sampling — making the configuration space exploration more efficient. Since its proposal, a few versions of the SAC algorithm have been published. The soft-value function has been dropped in the latest version using the relation between state-value and state-action-value function. The SAC has three main update equations:

Algorithm 6 Soft Actor-Critic [43]

Initialize θ_1, θ_2, ϕ
$\overline{\theta}_1 \leftarrow \theta_1, \overline{\theta}_2 \leftarrow \theta_2$
$\mathcal{D} \leftarrow \varnothing$
for each iteration=$1, 2, \ldots$ **do**
  $a_t \sim \pi_\phi(a_t|s_t)$
  $s_{t+1} \sim p(s_{t+1}|s_t, a_t)$
  $\mathcal{D} \leftarrow \mathcal{D} \bigcup \{(s_t, a_t, r(s_t, a_t), s_{t+1})\}$
end for
for each iteration=$1, 2, \ldots$ **do**
  $\theta_i \leftarrow \theta_i - \lambda_Q \nabla_{\theta_i} J_Q(\theta_i)$ for i $\in \{1, 2\}$
  $\phi \leftarrow \phi - \lambda_\pi \nabla_\phi J_\pi(\phi)$
  $\alpha \leftarrow \alpha - \lambda \nabla_\alpha J(\alpha)$
  $\overline{\theta}_i \leftarrow \tau\theta_i - (1-\tau)\overline{\theta}_i$ for i $\in \{1, 2\}$
end for
return θ_1, θ_2, ϕ

- $\nabla_\theta J_Q(\theta)$ for the $Q(s, a)$ state-action-value function,
- $\nabla_\phi J_\pi(\phi)$ for the policy, and
- $\nabla_\alpha J(\alpha)$ for the temperature parameter.

The full SAC algorithm is outlined in Algorithm 6. Additionally, the SAC borrows the idea of using two Q-networks to avoid overestimation from double Q-learning [114]. The state-action-value function parameters can be optimized by minimizing the Bellman residual

$$J_Q(\theta) = \mathbb{E}_{(s_t,a_t)\sim\mathcal{D}}[\frac{1}{2}\left(Q_\theta(s_t, a_t) - r(s_t, a_t) + \gamma\, \mathbb{E}_{(s_{t+1})\sim p}[V_\theta(s_{t+1})]\right)^2] \tag{2.39}$$

with $V_\theta(s_{t+1})$ given as

$$V(s_t) = \mathbb{E}_{a_t\sim\pi}[Q(s_t, a_t, \overline{\theta}) - \alpha \log(\pi_\phi(a_t|s_t))]. \tag{2.40}$$

Equation 2.39 can be optimized using a stochastic gradient descent method. The objective function for the policy is given by

$$J_\pi(\phi) = \mathbb{E}_{s_t\sim\mathcal{D}}[\mathbb{E}_{a_t\sim\pi_\phi}[\alpha \log(\pi_\phi(a_t|s_t)) - Q_\theta(s_t, a_t)]]. \tag{2.41}$$

Using the reparameterization trick, actions $a_t \sim \pi_\phi(a_t|s_t)$ can be backpropagated through the Q-networks. This makes it possible to directly optimize the policy using Equation 2.41. The temperature parameter α needs to be updated during the learning to avoid manual fine-tuning. The temperature parameter α is updated using

$$J(\alpha) = \mathbb{E}_{a_t\sim\pi_t}[-\alpha \log(\pi_\phi(a_t|s_t)) - \alpha\mathcal{H}] \tag{2.42}$$

	Invariant	Memory Efficiency	Relational Information
Images	Yes	Low	No
Feature vectors	Yes†	High	Yes
Sets	Yes	High	No
Graphs	Yes	High	Yes
Grids	Yes	Medium	No

Table 2.1: Overview of input representations used in deep learning. †: invariant depending on the choice of state.

This work uses the SAC algorithm to learn autonomous vehicles' behavior policies. Using experiences from the past (in the replay buffer) improves the sample-efficiency and reduces the need for simulation episodes. This is crucial for efficiently learning behavior policies in simulation. Further, maximizing the cumulative expected entropy guides the exploration without any hyperparameter tuning. This is especially important when learning behavior policies for autonomous vehicles as many combinatorial options are present in traffic — e.g., merging behind or in front of the other vehicle. The SAC algorithm and its performance using several hyperparameters, architectures, and reward signals are benchmarked and evaluated in Section 5.2.

2.2 Input Representation for Learning Behavior Policies

The input representation in MDPs has large effects on the learning efficiency, required memory, and transferability. It mainly determines the performance during learning and of the resulting behavior policy — whether the behavior policy is invariant towards the number and order of vehicles or if it generalizes to novel scenarios. Some input representations hold unused information, such as the backdrop in images which makes these less computationally efficient. Some input representations, such as graphs, are more efficient and allow a clear separation between intrinsic and relative values. This section discusses several characteristics of input representations for learning behavior policies.

Ideally, the input representation should have the following characteristics:

- *Invariant*: It should be invariant to the number and order of objects in the environment as these tend to change in dynamic environments, such as real-world traffic.
- *Transferable*: It should be transferable to novel scenarios and learn the underlying dynamics and relations instead of memorizing certain situations.
- *Efficient*: It should be efficient in computation and memory usage.

A comprehensive overview of the most common input representations is listed in Table 2.1 and additional context is provided here.

Images are a widely used input representation for learning behavior policies [78, 12, 117]. These can be directly fed from, e.g., cameras into learning algorithms.

However, these are not efficient in memory usage and computation as they often contain unused information that needs to be processed, such as the sky over the horizon. These are, however, invariant towards the number and order of vehicles as the input representation itself does not change its size depending on these factors. Generalization using images and neural networks can be improved using common machine learning techniques, such as drop out and convolutional layers.

Feature vectors are a commonly used input representation and have been widely used in RL [14, 46, 127, 4]. These tend to be more efficient than images as they only contain information that is required for learning behavior policies. However, using a row-vector that concatenates the vehicles' states sorted by distance is not invariant towards the number and order of vehicles. This can be mitigated by an intelligent design of the feature vector, such as using relative distances or distance-sensing antennas.

Sets have been utilized as input representation to learn behavior policies [128]. These are invariant to the number and order of traffic participants. However, these cannot include relative values between the states and connections cannot define how the set's entities can interact. Sets can be seen as a sub-form of graphs — nodes without any edges between these.

Graphs have gained popularity in machine learning recently and have been utilized as input representation in RL [121, 26, 67]. Graphs are invariant towards the number and order of vehicles in a scene when used together with graph neural networks (GNNs) as is elaborated in Section 2.4.1. They are efficient as the only required information is stored and graphs allow the specification of relational information in the edges. As the network does not have to learn the relations from other vehicles' states, it is efficient in computation and avoids learning incorrect relations.

Grid worlds also have been utilized in combination with RL [90]. These can be seen as a sub-form of images that have a very coarse discretization. Thus, these methods offer the same advantages and suffer from the same drawbacks as images but are slightly better in computational and memory efficiency due to the discretization.

2.2.1 Feature Vector Representation

Feature vectors have been utilized in MDP settings, such as in [14, 46, 127, 4, 74]. The feature vector representation serves as baseline for developing and benchmarking the novel GNN actor-critic architecture presented in Section 2.4. In this work, a feature vector is defined as a one-dimensional row-vector that holds the information of objects in the semantic environment, such as the vehicles' states. Particularly, the states of the vehicles are sorted by distance and then concatenated into a single row-vector. In combination with DNNs this makes the input representation not invariant towards the number and order of vehicles as the input size has to be fixed. For the output of the DNNs to be well-behaved, the input representation should be normalized (e.g., in a range of $[-1, 1]$). However, as the input size of DNNs is fixed, this means that the rest of the feature vector has to be filled with default values if there are fewer vehicles — possibly introducing unwanted effects.

Relational information can be encoded into feature vector representations, e.g., by using an intelligent choice of reference frame. However, these usually allow only one point of reference as otherwise the input size would become dynamic. This also poses the challenge of how absolute information can then be included in the feature vector, such as absolute coordinates.

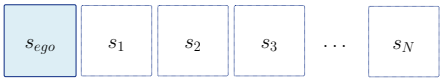


Figure 2.4: Feature vector representation with the ego vehicle's state in the first position and the other states sorted based on their distance to the ego vehicle.

Figure 2.4 depicts the feature vector representation with the ego vehicle's state depicted in blue. Algorithm 7 outlines a 'ClosestAgentsObserver' that outputs a feature vector comprised of objects in its close proximity (threshold radius r_{max}). A maximum number of vehicles N_{max} is defined that determines together with the 'GetTransformedState' the length of the observation.

Algorithm 7 ClosestAgentsObserver

function OBSERVE(*agents*, *ego_agent*)
  s'_{ego} = GetTransformedState(*ego_agent*), *states* = $\emptyset$
  states $\leftarrow s'_{\text{ego}}$
  sorted_agents = GetAgentsByDistance(*agents*, *ego_agent*)
  for all *agent* $\in$ *sorted_agents* **do**
    $s'_{\text{agent}} \leftarrow$ GetTransformedState(*agent*)
    if $d_{Euclidean}(s'_{\text{ego}}, s'_{\text{agent}}) < r_{\max}$ **and** *agent* $\neq$ *ego_agent* **then**
      states $\leftarrow s'_{\text{agent}}$
    end if
  end for
  return Concatenate(*states*)
end function

The 'GetTransformedState' function in Algorithm 7 returns a subset of the agent's state s' and transforms this subset further (e.g., by using relative instead of absolute values). The 'Concatenate' function concatenates all states and returns a fixed-size vector in which the empty spaces are filled with default values.

2.2.2 Graph Representation

Graphs have been utilized as input representation for learning behavior policies and predicting traffic in various works [121, 26, 67]. This work proposes a graph representation that can be used for learning behavior policies within an AC RL setting.

Graphs can have various characteristics, such as being cyclic and directed. This work utilizes directed graphs having nodes $n \in N$ and edges $e \in E$. Additionally,

both – the nodes and edges – have values, the node values h_i for the i-th node and edge values e_{ij} connecting the i-th with the j-th one. A full graph definition is then given by $G = (N, E)$.

The graph representation in combination with GNNs is invariant to the number and order of objects in the environment. The invariance stems from the fact that these graphs are being processed by GNNs that apply an approximation function to update the node and edge values element-wise.

Due to the graph representation having node and edge values, a clear separation of intrinsic and extrinsic values can be achieved — e.g., the node values storing the intrinsic state values and the edge values the relational ones to other objects. It also brings the advantage of the neural network not having to infer these relations as these are explicitly given.

Graphs are efficient as they only store the required information and include relational information for each node. As GNNs use a single neural network to update the node and edge values element-wise, these have a lower number of parameters compared to conventional DNNs.

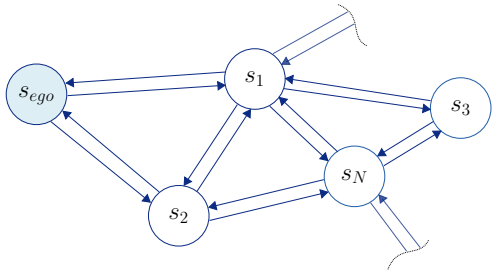


Figure 2.5: Graph representation with the ego vehicle's state depicted in blue.

For a semantic environment to be converted to a graph, a *GraphObserver* is used outlined in Algorithm 8. As in Algorithm 7, a 'GetTransformedState' function is used that selects a subset of the agent's state and transforms this subset into, e.g., relative edge features Δ. The graph observer loops through all agents nearby and adds these as nodes to the graph. Further, for the added node, it checks to which surrounding nodes it should be connected based on a defined maximum distance $r_{max,G}$ to the ego vehicle and $r_{max,L}$ if both nodes are near to each other. If a surrounding node is within this distance, the nodes are connected by adding it to the edge index and adding an edge value e_{ij}.

2.3 Reward Signals for Learning Behavior Policies

This section discusses the reward signal choice within MDP settings for learning behavior policies for autonomous vehicles. First, a general discussion on the design of reward signals is provided and, subsequently, potential-based reward shaping functions for learning behavior policies are discussed.

Algorithm 8 GraphObserver

function OBSERVE($agents$, ego_agent)
  s'_{ego} = GetTransformState(ego_agent)
  $sorted\_agents$ = GetAgentsByDistance($agents$, ego_agent)
  $node\_values = \emptyset, edge\_index = \emptyset, edge\_values = \emptyset$
  for $i, agent \in sorted\_agents$ **do**
    $s'_{\text{agent}} \leftarrow$ GetTransformedState(agent)
    $node\_values \leftarrow s'_{\text{agent}}$
    for $j, other\_agent \in sorted\_agents$ **do**
      $s'_{\text{i}} \leftarrow$ GetTransformedState($agent$)
      $s'_{\text{j}} \leftarrow$ GetTransformedState($other\_agent$)
      if $d_{\text{Euclidean}}(s'_{ego}, s'_i) < r_{max,G}$ **and** $d_{\text{Euclidean}}(s'_i, s'_j) < r_{max,L}$ **then**
        $edge\_index \leftarrow (i, j)$
        $\Delta \leftarrow ||s'_{\text{i}} - s'_{\text{j}}||_1$
        $edge\_values \leftarrow \Delta$
      end if
    end for
  end for
  return $node\_values, edge\_index, edge\_values$
end function

2.3.1 Design of the Reward Signal

The reward signal design is essential in setting up an MDP problem formulation and selecting a corresponding solution method. Some solution methods can, e.g., cope better with sparse reward signals than others. Reinforcement learning solely uses the reward signal r_t for learning how to behave.

One way is to define a sparse reward signal in which the agent only receives a positive reward for, e.g., reaching the goal or a negative reward for, e.g., having a collision. This, however, causes a credit assignment problem in which the algorithm has to figure out which actions lead to the desired or undesired outcome [64]. Further, using a sparse formulation can require the agent to explore an infeasible number of states in the configuration space, such as with the mountain car problem [41]. Thus, hindering the usage of RL in complex, real-world tasks.

Reward shaping transforms sparse rewards into continuous ones and increases the learning process and overall performance. Reward shaping alters the underlying MDP formulation from $M = \langle S, A, P, \gamma, R \rangle$ to $M' = \langle S, A, P, \gamma, R' \rangle$ where $R' = R + F$ is the reward function of the transformed MDP [81]. The reward shaping function $F : S \times A \times S$ is a bounded real-valued function. Ng et al. [81] introduce necessary and sufficient conditions for reward shaping functions to leave optimal policies invariant. This is an important characteristic as the reward shaping otherwise might introduce

unwanted behaviors into the resulting policy. A potential-based reward shaping function is given by

$$F(s_t, a_t, s_{t+1}) = \gamma\Phi(s_{t+1}) - \Phi(s_t) \tag{2.43}$$

with $\Phi(s)$ being the potential function, s_t the state at time t, and s_{t+1} the state in the next time step [81]. Additionally, as in the MDP formulation, a discout factor γ is utilized. Reward shaping has been shown to improve the overall learning performance and make complex problems feasible [72]. A potential-based reward shaping function also avoids negative cycles due to poorly defined guiding rewards [81].

As most state-of-the-art reinforcement learning algorithms use DNNs as function approximators, the reward signal is favored to be in a "well-behaved" range as neural networks have shown to perform better in these — e.g., by limiting the mean of the reward signal to a range of $[-1, 1]$ and the standard deviation in a range of $[0, 1]$. To achieve such a reward signal, the rewards can be transformed using a normalization as

$$r_t = \frac{\sum_{t=0}^{T} r_t - \mu}{\sqrt{\frac{\sum_{t=0}^{T}(r_t-\mu)^2}{T}} + \epsilon} \tag{2.44}$$

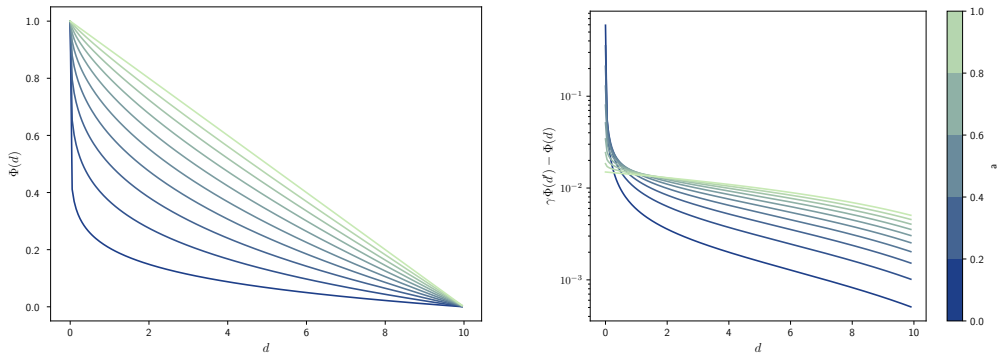
with μ being the mean of the rewards r_t in the replay buffer and ϵ a small value avoiding a division by zero.

Another way to improve the efficiency of RL algorithms is to use differentiable rewards as proposed in [122]. However, this is only possible if the models of other agents and the environment are also differentiable. In uncertain environments, such as real-world traffic, the models are mainly unknown, making formulating differentiable rewards challenging in praxis.

2.3.2 Potential-based Reward Shaping for Autonomous Vehicles

This section introduces a potential-based reward shaping for learning behavior policies for autonomous vehicles based on [81] and extends the work presented in [63]. The aim is to transform a sparse reward signal into a continuous one to foster efficiency during learning and mitigate the credit assignment problem. There are two primary considerations in formulating the reward signal for autonomous vehicles: safety and goal-oriented behaviors. Safety should be weighted more than the goal orientation as not causing collisions is essential to the operation of autonomous vehicles. Further, a goal-driven behavior is required to properly function as an autonomous system that is to be accepted by society. For example, it would not be desirable for autonomous vehicles to stand on a lane for minutes waiting to merge (the so-called "frozen robot problem").

A reward shaping function $F(s_t, a_t, s_{t+1})$ is used that transforms the original reward signal $r_t = R(s_t, a_t, s_{t+1})$ to a shaped reward signal $r_t^s = R(s_t, a_t, s_{t+1}) + F(s_t, a_t, s_{t+1})$ with t being the time. As proposed by Ng et al. [81], a potential-based formulation is used for the reward shaping function that is shown in Equation 2.43. This section proposes several potential-based reward shaping functions that aim to increase the sample efficiency of the used RL algorithm.



(a) Distance potential function $\Phi(d)$. **(b)** Distance reward shaping function F with $d' < d$.

Figure 2.6: On the left side, the distance potential function to the goal $\Phi(d)$ is shown for various exponents a and on the right side the corresponding reward shaping function.

The distance to the goal can be modeled as a potential function to incentivize a goal-driven behavior. The distance potential function is defined as

$$\Phi(d) = 1 - (\frac{d}{d_{max}})^a \tag{2.45}$$

with d being the distance to the target, d_{max} being the maximum distance, and a being the exponent determining the slope's steepness. Figure 2.6 shows the distance potential function on the left and the reward shaping function on the right. As the distance d becomes smaller, the potential function increases until it ulimatively reaches 1 at $d = 0$. The figure plots several distance potential functions having different exponentials a resulting in different slopes.

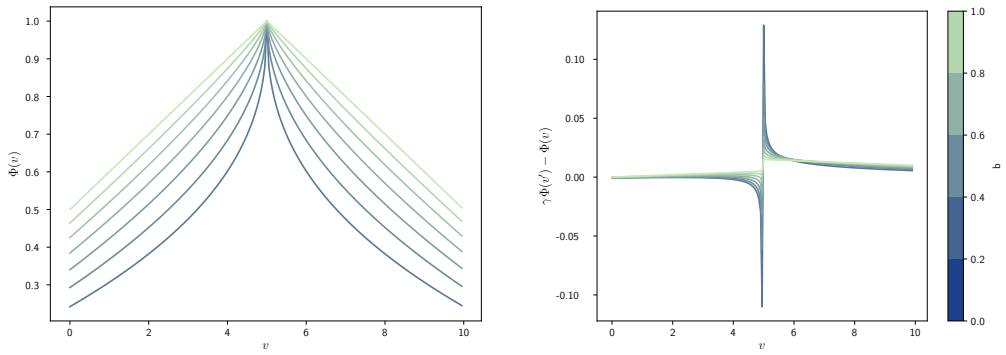
Similar to the distance potential function, the velocity potential function $\Phi(v)$ is defined as

$$\Phi(v) = 1 - (\frac{||v - v_{des}||}{\Delta v_{max}})^b \tag{2.46}$$

with v being the velocity, Δv_{max} being the maximum deviation of the desired velocity v_{des}, and b being an exponent determining the slope's steepness. Figure 2.7 shows the velocity potential function on the left and the resulting reward shaping function on the right. Combing the distance and velocity potential function yields a resulting potential function of

$$\Phi(d, v) = \frac{\Phi(d) + \Phi(v)}{2} \tag{2.47}$$

that is normalized to limit the reward signal to a well-behaved range of $[0, 1]$. If the RL algorithm normalizes the reward the normalization can be omitted. The resulting potential function $\Phi(d, v)$ is shown in Figure 2.9 (a) with a desired velocity of $v_{des} = 5\text{m/s}$.



(a) Velocity potential function $\Phi(v)$ **(b)** Velocity reward shaping function F with $v' < v$.

Figure 2.7: On the left side, the velocity potential function $\Phi(v)$ is shown for various exponents b and on the right side the corresponding reward shaping function.

The thus far introduced reward potentials do not include any static or dynamic objects of the environment. A potential function $\Phi_{other}(d_i)$ is introduced for other objects that returns a negative value the closer the ego vehicle is to the i-th object — returning negative one at a distance of zero. The distance potential function to other objects is defined as

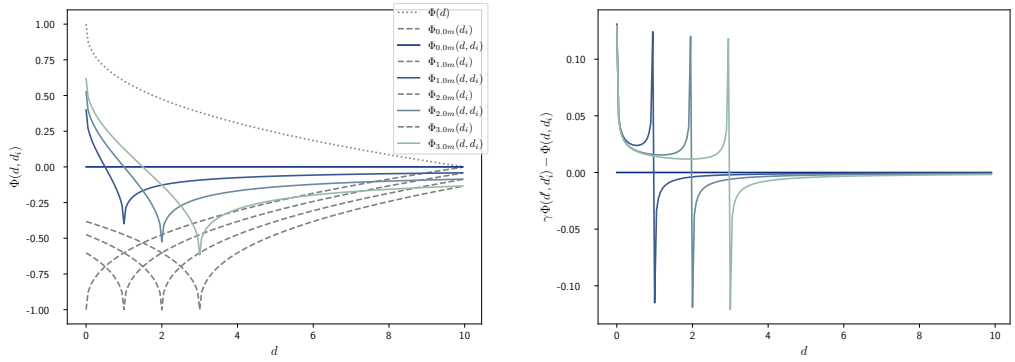
$$\Phi_{other}(d_i) = (\frac{||d_i||}{d_{max}})^c - 1. \tag{2.48}$$

with d_i being the Euclidean distance to the i-th object, d_{max} being the maximum assumed distance, and c being the exponent determining the slope's steepness. Thus, the potential functions' overall range is extended from $[0, 1]$ to $[-1, 1]$.

The potential function Φ can be split into potentials returning positive and negative values into Φ^+ and Φ^-, respectively. Both groups are normalized using their average to avoid too large positive or negative values. The resulting total potential Φ_t is then given by

$$\Phi_t = \frac{1}{P}\sum_{i=0}^{P}\Phi_i^+ + \frac{1}{L}\sum_{i=0}^{L}\Phi_{other,i}^-. \tag{2.49}$$

with P being the number of terms in the positive potential and L the number of terms in the negative potential. In Figure 2.8, the combined distance potential function $\Phi(d, d_i)$ is depicted on the left. The dotted line depicts the goal distance potential function $\Phi(d)$ and the dashed lines show the distance potential functions $\Phi(d_i)$ for various distances to other objects – the object being placed at the distances $d = 0, 1, 2, 3m$. On the right side in Figure 2.8, the resulting reward shaping functions for varying object distances are shown. As can be seen, the reward shaping returns a negative value with a decreasing distance d to the object. A negative value is returned because the object is closer than the desired goal on the x-axis, and the agent cannot reach the goal in one dimension without causing a collision before



(a) Distance potential function $\Phi(d, d_i)$.

(b) Reward shaping function F with $d' < d$.

Figure 2.8: The combined distance potential function $\Phi(d, d_i)$ is shown on the left and the resulting reward shaping functions on the right. Both potential functions use an exponent of $a = c = 0.4$.

reaching the goal. If both objects have the same distance, the resulting reward shaping function always returns zero. To avoid objects influencing the learning policy from too far, the distance potential function to other objects $\Phi(d_i)$ can be set to be active only if $d_i \leq d_{threshold}$. In Figure 2.9 (b), the combined potential function $\Phi(d, d_i)$ plotted over the Cartesian distances d_x and d_y is shown. There are two objects present in the plot, which can be seen as negative dents in the potential function's surface. On the left, where d_y is zero, the distance to the goal is zero, and the distance potential function $\Phi_d(d)$ increases its value to one.

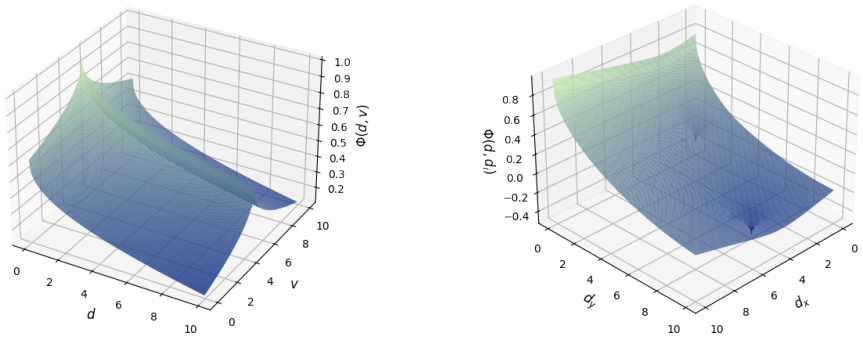
A potential function can be defined that includes the distance to the goal, other objects, and the desired velocity using all of the above-introduced potential functions. The total potential function $\Phi_t(d, d_i, v)$ can then be defined as

$$\Phi_t = \frac{\Phi(d) + \Phi(v)}{2} + \frac{1}{N} \sum_{i=0}^{N} \Phi(d_i). \tag{2.50}$$

with N vehicles in the scenario. It is also possible to use, e.g., multiple goal potentials if there are multiple goals the agent could reach. The proposed reward shaping functions are plotted for real-world scenarios and evaluated in Section 5.2.2.

2.4 Graph Neural Networks and Actor-Critic Reinforcement Learning

This section provides an overview of GNNs and how these can be utilized for learning behavior policies for autonomous vehicles. The unified graph network framework by Battaglia et al. [9] is outlined that can model a wide variety of GNN architectures. A novel GNN actor-critic architecture for learning behavior policies for autonomous



(a) $\Phi(d, v)$ **(b)** $\Phi(d, d_i)$

Figure 2.9: On the left, the potential function $\Phi(d, v)$ is shown and on the right the potential function $\Phi(d, d_i)$. Both potential functions use an exponent of $a = b = c = 0.4$.

vehicles is introduced. The novel architecture is invariant towards the number and order of vehicles and is able to learn directly on graphs.

2.4.1 Overview of Graph Neural Networks

Learning directly on graphs or manifolds is grouped under the umbrella term *geometric learning* [17]. Recently, vast progress has been made in the conventional, Euclidean domain using deep neural networks, such as convolutional neural networks (CNNs) [18]. However, these advances and methods cannot be straightforwardly applied to the non-Euclidean domain. A solution method trying to transfer and apply these concepts to the non-Euclidean domain and develop new ones are GNNs. GNNs are at the intersection of conventional deep learning and structured approaches. They were first applied by Sperduti et al. [105] to directed acyclic graphs in 1997. However, the notion of GNNs was first used by Gori et al. [40] in 2005 and further elaborated by Gallicchio et al. [35] in 2010. GNNs apply the same edge and node approximation functions to each edge and node per layer, respectively. Thus, these inherently support a form of combinatorial generalization [9]. For applications, such as learning behavior policies for autonomous vehicles, this is especially important due to the high number of combinatorial options (maneuver variants).

A comprehensive survey of GNNs is provided by Wu et al. [125]. In their work, they propose a taxonomy to divide GNNs into four categories:

- recurrent graph neural networks (RecGNNs),
- convolutional graph neural networks (ConvGNNs),
- graph autoencoders (GAEs), and
- spatial-temporal graph neural networks (STGNNs).

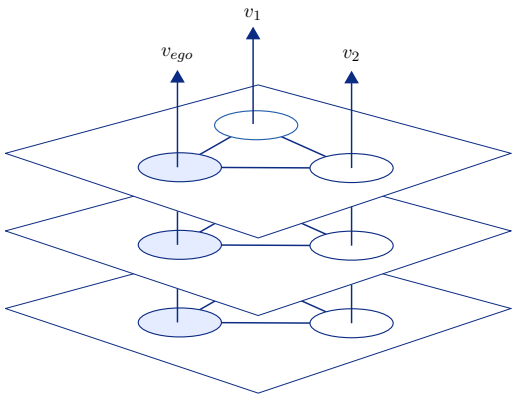


Figure 2.10: GNN having three layers with the ego vehicle's node depicted in blue. GNNs are graph-to-graph modules that output the same graph structure as the input.

RecGNNs can be viewed as the pioneers of GNNs [125]. These approaches utilize recurrent neural networks to learn node representations. Due to computational constraints during the early times, early research mainly focused on acyclic graphs [105, 76]. Gori et al. [40] extended these approaches to handle all types of graphs. The underlying idea is to update the nodes' states by exchanging the neighborhood information until a stable equilibrium is reached [125].

ConvGNNs utilize similar to Euclidean grid-based methods convolutional operations. The underlying idea is to aggregate the neighbors' features and update the node's representation using these. Contrary to RecGNNs, ConvGNNs stack multiple layers on top of each other to pass messages through the graph. ConvGNNs are more often utilized due to their higher efficiency than RecGNNs [125]. The ConvGNNs approaches can be further divided into two categories: spectral- and spatial-based methods.

Spectral-based methods have a strong mathematical background and emerged from the signal processing domain [59]. The first spectral ConvGNNs was proposed by Micheli [75] in 2009. Spectral methods either need to perform eigenvector computations or handle the whole graph at once [125]. As the eigenvector computation has a complexity of $O(n^3)$ these are only applicable for relatively small problems. Also, it has shown that spectral methods based on a Fourier basis generalize poorly and are limited to operating on undirected graphs [125]. Thus, in this work, the focus is put on spatial methods that are discussed in the following.

Spatial methods are often preferred over spectral ones due to their efficiency and flexibility [125]. Spatial-based models are more flexible and can operate on a wide range of graphs, such as graphs having edge inputs [40, 39], directed graphs [5], and others [24]. Figure 2.10 shows three layers of a ConvGNN with the ego vehicle's node depicted in blue.

GAEs just like their Euclidean counterpart encode the input graph into a latent space and reconstruct the input graph from this latent space. These methods are additionally used to generate novel graph structures from the latent space [21]. As autoencoders try to recreate the input graph from the latent space, these methods can be classified as unsupervised learning.

A relatively new class of GNNs are STGNNs. They aim to learn spatial information that is underlying in the graph and include an additional dimension — the time. Applications that require temporal information require such methodologies, such as traffic speed forecasting [69] and driver maneuver anticipation [53]. However, as RL assumes the state to be Markovian and the information to be included in the current state, STGNNs are not required. The usage of these within RL could be investigated in future work.

The next section utilizes the unified GNN framework representing ConvGNNs for learning behavior policies for autonomous vehicles.

2.4.2 Unified Framework: Graph Blocks and Interaction Networks

Battaglia et al. [9] present a unified graph network (GN) that consists of graph network blocks (GN blocks). The GN aims at generalizing and extending various GNNs concepts and to support constructing complex architectures from simple building blocks [9]. The term "neural" in "graph networks" is avoided as other function approximators can be used as well, such as table-based ones.

The central entity in the proposed GN is a GN block, which is a "graph-to-graph" module that takes an input graph, performs computations, and returns a graph having the same structure as the input graph but having updated node, edge, and global values. Graphs are used that have the following characteristics: directed, attributed, and multi-graph. Within the unified GN framework, a graph is defined as a 3-tuple $G = (\mathbf{u}, V, E)$. The $\mathbf{u}$ represents a global attribute, e.g., a gravitational field, $V = \{\mathbf{v_i}\}_{i=1:N^v}$ is the set of nodes (of cardinality N^v) where each $\mathbf{v_i}$ is a node attribute, and $E = \{(\mathbf{e}_k, r_k, s_k)\}_{i=1:N^e}$ is the set of edges (of cardinality N^e) where each $\mathbf{e}_k$ is an edge attribute with r_k being the sender's and s_k the receiver's index.

A GN block contains three update functions, $\phi^{\square}$, and three aggregation function, $\rho^{\square}$

$$\begin{aligned} \mathbf{e}'_k &= \phi^e(\mathbf{e}_k, \mathbf{v}_{r_k}, \mathbf{v}_{s_k}, \mathbf{u}) & \bar{\mathbf{e}}'_i &= \rho^{e\to v}(E'_i) \\ \mathbf{v}'_i &= \phi^v(\bar{\mathbf{e}}'_i, \mathbf{v}_i, \mathbf{u}) & \bar{\mathbf{e}}' &= \rho^{e\to u}(E') \\ \mathbf{u}' &= \phi^u(\bar{\mathbf{e}}', \bar{\mathbf{v}}', \mathbf{u}) & \bar{\mathbf{v}}' &= \rho^{v\to u}(V') \end{aligned} \tag{2.51}$$

with $E'_i = \{(\mathbf{e}'_k, r_k, s_k)\}_{R_k=i, k=1:N^e}$ and $V' = \{\mathbf{v}'_\mathbf{i}\}_{i=1:N^v}$. The edge update function ϕ^e is shared across all edges to compute per-edge updates, and the node update function ϕ^v is shared across all nodes to compute per-node updates. The aggregation function $\rho^{\square}$ takes a set as an input and reduces this set to a single element which represents the aggregated information [9]. The aggregation function has to be invariant to permutations and has to be able to take a variable number of arguments

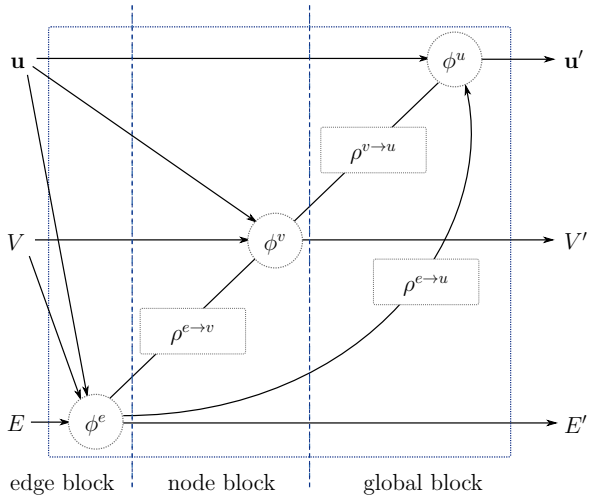


Figure 2.11: Unified graph block as proposed in [9]. Inputs are the global values $\mathbf{u}$, the node values V, and the edge values E and their respective output values are $\mathbf{u}', V', E'$.

(e.g., elementwise summation or the mean). By using the GN block outlined in Figure 2.11, a variety of GNs can be modeled. The full outline of a GN block is provided in Algorithm 9.

Algorithm 9 Graph Network Block [9]

function GRAPHNETWORK($E, V, \mathbf{u}$)
  for $k \in \{1 \dots N^e\}$ **do**
    $\mathbf{e}'_k \leftarrow \phi^e(\mathbf{e}_k, \mathbf{v}_{r_k}, \mathbf{v}_{s_k}, \mathbf{u})$
  end for
  for $i \in \{1 \dots N^v\}$ **do**
    let $E'_i = \{(\mathbf{e}'_k, r_k, s_k)\}_{r_k=i, k=1:N^e}$
    $\bar{\mathbf{e}}'_i \leftarrow \rho^{e\to v}(E'_i)$
    $\mathbf{v}'_i \leftarrow \phi^v(\bar{\mathbf{e}}'_i, \mathbf{v}_i, \mathbf{u})$
  end for
  let $V' = \{\mathbf{v}'_\mathbf{i}\}_{i=1:N^v}$
  let $E' = \{(\mathbf{e}'_k, r_k, s_k)\}_{k=1:N^e}$
  $\bar{\mathbf{e}}' \leftarrow \rho^{e\to u}(E')$
  $\bar{\mathbf{v}}' \leftarrow \rho^{v\to u}(V')$
  $\mathbf{u}' \leftarrow \phi^u(\bar{\mathbf{e}}', \bar{\mathbf{v}}', \mathbf{u})$
  return $(E', V', \mathbf{u}')$
end function

The interaction networks (INs) proposed by Battaglia et al. [8] can be modeled using GN blocks. Their work has shown that complex interactions, such as planetary

systems and their movements, can be learned using INs. The next section introduces the novel GNN AC RL architecture for learning behavior policies that utililzes GN blocks.

2.4.3 Graph Neural Network Actor-Critic Architecture

This section introduces a novel GNN actor-critic architecture that is invariant towards the order and number of vehicles as discussed in Section 2.2. This invariance makes it ideal for learning behavior policies in traffic where the number and order of vehicles can change rapidly. The novel architecture is based on the unified graph block framework presented in the previous section and is used in the actor- and critic-networks.

The unified graph block framework is a ConvGNN that stacks graph layers on top of each other enabling information to be passed throughout the graph. Information can flow between all connected nodes, which enables to model relations between objects by connecting these with edges. Thus, the graph structure determines which objects in the scenario can influence each other. Graphs as input representations and their creation have been discussed in Section 2.2. The idea of the novel GNN actor-critic architecture is to use several GN blocks layers to pass information between the nodes (vehicles) followed by using dense DNNs that then either learn an action distribution or a state-value function.

By chaining multiple GN blocks, information is propagated throughout the graph l times, with l being the number of ConvGNN layers. For example, if there are three ConvGNN layers, the ego vehicle could receive information from as far as three vehicles away. The chaining of GN blocks can be mathematically denoted as $G_L = GN_1 \circ GN_2 \circ \dots GN_l$ with G_L being the final outputted graph. Alternatively, this can be expressed as $G_L = GN_l(GN_{l-1}(\dots, GN_1(G_0)))$ with G_0 being the input graph. As GN blocks are "graph-to-graph" modules, the GN block GN_l takes the graph G_l as input and outputs the graph G_{l+1}. The GN blocks update all node and edge values at each processing step.

The idea of the novel architecture is for information to aggregate in a single node (e.g., the ego vehicle node) in an embedding that contains all required information for the ego vehicle to make informed decisions. Each node contains an embedding vector v_i^L for the i-th vehicle at the output layer L. The embedding vector size always remains of fixed size regardless of how many other vehicles there are in the scene, which makes the novel architecture invariant towards the number of vehicles. The node value v_{ego}^L is passed through a dense DNN to either learn a distribution of actions or a scalar value for the actor and critic network, respectively.

Applying this novel GNN architecture to, e.g., the SAC algorithm yields in total three GNN networks — two for the state-action-value networks $Q_1^\theta(s,a), Q_2^\theta(s,a)$ and one for the actor-network $\pi^\phi(a|s)$. The complete GNN actor-critic network architecture is outlined in Figure 2.12 having GN blocks and dense DNN layers.

Section 5.2 benchmarks the performance of the novel GNN actor-critic architecture against conventional DNNs, evaluates the generalization of the architectures, and

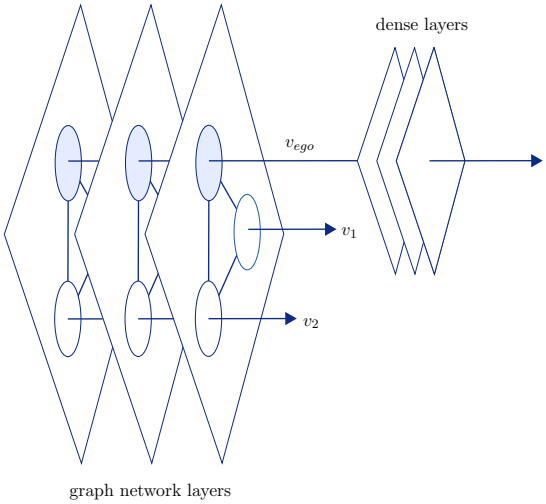


Figure 2.12: GNN architecture for actor-critic reinforcement learning (modified graphic from [45], ©2020 IEEE). In case of being used in the actor-network, the final dense layer outputs parameters for,e.g., a normal distribution. For the state-value-action function, the last layer outputs a deterministic value.

visualizes the edge values of the GNNs. It is shown that the novel architecture outperforms conventional DNN architectures and also generalizes better than these.

2.5 Summary and Remarks

Data-driven methods for learning behaviors can provide a range of benefits, such as reduced parameter tuning and no required prior knowledge about the environment's dynamics. These methodologies can learn how to behave by, e.g., collecting experiences and adapting their driving behavior over time.

This section provided a holistic view on learning behavior policies for autonomous vehicles using AC RL starting from the input representation, over the reward signal and shaping, to the selection of state-of-the-art AC RL solution methods. A novel GNN actor-critic architecture has been introduced that is invariant towards the number and order of vehicles and that has been shown to outperform conventional DNN architectures.

The MDP formulation is discussed in detail and its entities, the input representation, the reward signal and transition probabilites are discussed in Section 2.1.

An overview of common input representation for semantic environments for learning behavior policies is given, ranging from feature vectors to graphs. Observers that transform semantic environments to a machine learning suitable representation are discussed in detail in Section 2.2.

Further, sparse reward signals and reward shaping have been introduced and discussed. Sparse reward signals generally introduce the *credit assignment problem*

where the agent has to figure out which action led to a desired or undesired outcome. Several potential-based reward shaping functions are introduced in this thesis for learning behavior policies for autonomous vehicles to avoid this.

The solution methods for MDPs have been discussed in Section 2.1.4. The chosen solution methods have a large impact on the resulting behavior policy due to, e.g., the exploration mechanism, being on- or off-policy and its underlying theory in general. Reinforcement learning algorithms – in particular, actor-critic reinforcement learning – have been discussed in detail in Section 2.1.4.

A novel GNN actor-critic architecture for learning behavior policies in semantic environments has been introduced Section 2.4. The invariance towards permutations (e.g., the number and order of traffic participants in the scene) makes the novel architecture an ideal candidate for learning behavior policies for autonomous vehicles. Using GNNs allows a clear separation of agent-intrinsic states and relative values to other agents using node and edge values, respectively. Further, using GNNs makes visualizing the information flow between nodes possible to gain additional insights into the learned behavior policies.

Learning-based methods for behavior generation will become more prominent as data-driven methods are required to integrate well into mixed traffic and learn from experiences. This chapter provided a thorough investigation and overview of learning behavior policies for autonomous vehicles in MDP settings.

3 Evaluating Learned Behavior Policies for Autonomous Vehicles

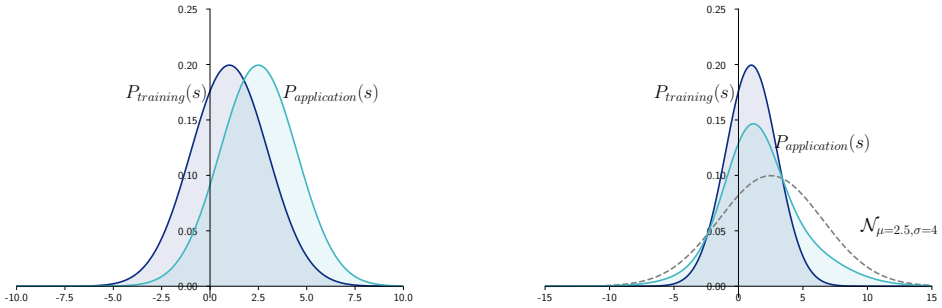
Approaches are outlined that enable utilizing learned behavior policies in safety-critical applications ranging from safe reinforcement learning (SRL), over runtime safety assurance (RTSA), to conventional methods utilizing learned behavior policies. SRL is an umbrella term for methodologies that aim to make reinforcement learning (RL) directly applicable to real-world, safety-critical applications. An overview of state-of-the-art RTSA frameworks is provided to handle all kinds of black-box approaches, such as deep neural networks (DNNs). Finally, approaches are discussed that utilize learned behavior policies, such as, e.g., search-based methods using learned behavior policies as heuristics.

This chapter introduces a counterfactual behavior policy evaluation (CBPE) that evaluates the performance of learned behavior policies at runtime using counterfactual worlds — worlds in which others' behaviors can be non-factual. Using counterfactual worlds, the learned behavior policies' generalization capabilities and how it copes with distributional shifts can be evaluated. The learned behavior policies' performance can then be used, e.g., in the Simplex decision logic as switching criterion. Using a performance-based measure avoids overapproximated safe operational regions and restricts the policies' usage merely based on its performance.

3.1 Introduction and Overview

Applying learned behavior policies for autonomous driving is challenging for various reasons ranging from the current standards to the challenges associated with machine learning approaches. Neither the ISO 26262 [52] nor the safety of the intended functionality (SOTIF) [51] standards have been designed with autonomous driving in mind [19]. The SOTIF merely states that "functional insufficiencies within the machine learning functions have to be minimized" [51]. Further, the SOTIF defines requirements for the general verification and validation (V&V) process as follows:

> Evaluate the safety of the intended function concerning known triggering events to provide an argument that the residual risk associated with hazards caused by known insufficiencies in the system is sufficiently low [51].
>
> Providing an argumentation that the residual risk associated with hazards caused by unknown insufficiencies in the system is sufficiently low for the intended function [51].



(a) Distributional shift in training versus application.

(b) Noise changing the distribution.

Figure 3.1: Distributional shift and noise that can differ during training and application of learned behavior policies.

However, the standard does not provide a clear, holistic view of how this can be achieved for machine learning components and learned behavior policies for autonomous vehicles. When dealing with DNNs, shifts in the input distributions [118], generalization capabilities [3], noise, and transferability [126] can lead to unsafe behaviors without the need of, e.g., a triggering event being present. Figure 3.1 visualizes different distributions during training and application — on the left caused by a distributional shift and on the right by noise. As is shown in the evaluation in Section 5.2.4, deviations of the other vehicle's behavioral parameters from training to application cause unsafe behaviors using learned behavior policies — if the deviations exceed a certain threshold.

Simulation can be used to a certain extent to generate an argumentation for the residual risk and the functional correctness of the driving properties of learned behavior policies using simulation frameworks, such as BARK [13]. With such frameworks, the generalization capabilities and how well DNNs can cope with distributional shifts can be evaluated. However, when using simulation having the goal in mind of deploying behavior policies in the real-world, the gap between the behavior of the simulated agents and real-world traffic has to be small. Simulating real-world traffic most likely requires a data-driven, possibly machine learning approach that faces the same issues as learning behavior policies using RL for the ego vehicle. Further, due to the vast scenario parameter space of traffic (including the behavioral parameters of traffic participants, the map, and the initial states of all objects), it is not possible to achieve full coverage of the whole space in simulation — leaving a residual risk for the learned behavior policy to malfunction and to produce unsafe driving behavior. Complete coverage would be essential in simulation when using function approximators, such as DNNs as already small shifts in the distribution can lead to unsafe behavior. There are various ongoing efforts in making learning-

based approaches applicable to real-world, safety-critical applications, which will be discussed in the next section.

3.2 State of the Art of Learning-Based Behavior Policies in Safety-Critical Applications

This section outlines methodologies that aim at making executing learned behavior policies safer – with some enabling the utilization of learned behavior policies in safety-critical applications. The discussion starts with SRL that makes RL safe for learning and execution in the real world by modifying the optimality criterion or restricting the exploration and exploitation process.

RTSA frameworks are discussed for ensuring safety for learned behavior policies or black-box approaches. These often build upon the Simplex architectures' principles having a high-performance controller, a high-assurance controller, and a decision logic that switches between these. Several RTSA approaches are outlined and discussed in Section 3.2.2.

Conventional approaches, such as search- and optimization-based approaches, can provide some safety guarantees. By employing a learned behavior policy, their performance can be increased as the offline computational power of learning-based approaches can be harnessed. These are discussed and outlined in Section 3.2.3.

3.2.1 Safe Reinforcement Learning

Safe reinforcement learning can be defined as the process of learning behavior policies that maximize the cumulative expected future reward but adhere to reasonable system performance, and safety constraints during learning and deployment [36]. In a more general perspective, SRL groups methodologies that aim to make RL utilizable in safety-critical and real-world applications. García et al. [36] propose a taxonomy for dividing SRL approaches into two classes:

- SRL that modifies the *optimality criterion* using a safety factor, and
- SRL that modifies the *exploration and exploitation* process using external knowledge or using risk metrics.

SRL that modifies the *optimalilty criterion* can further be subdivided into:

- *Worst case criterion*: A penalty for the variability induced by a given policy is included in the optimization criterion. The variability present can either be due to: (1) inherent uncertainties related to the stochastic nature of the system or (2) the parameter uncertainty that is not known in the MDP exactly. To avoid such, the agent maximizes the return associated with the worst-case scenario [47].

- *Risk sensitive criterion*: A balance between large rewards and avoiding catastrophic situations is tried to be achieved. Exponential utility functions can, e.g., be used to induce risk-averse or inducing behaviors [77].
- *Constrained criterion*: E.g., assuming an underlying constrained Markov decision process (MDP) that has a set of constraints [79].
- *Other optimization criteria*: E.g., including risk metrics [112] or estimating the density of the returns [80].

The second class of SRL algorithms control the *exploration and exploitation* process and can further be subdivided into:

- *External knowledge*: Using initial knowledge [28], deriving a policy from a finite set of examples [1], or guiding the exploration process [91].
- *Risk-directed exploration*: E.g., employing risk metrics [37].

SRL that modifies the optimality criterion alone is often not sufficient in safety-critical applications as this will lead to – at least initially – collisions during the exploration. In simulation frameworks, this is acceptable but not in real-world, safety-critical applications, such as autonomous driving. The second-class of SRL algorithms modify the exploration and exploitation process using external knowledge and risk metrics. These – also in combination with the later discussed RTSA frameworks – are more promising candidates for making learned behavior policies utilizable in safety-critical applications, such as autonomous driving. In the following, related work of SRL is outlined that has been applied to safety-criterial applications, such as autonomous driving.

Pek et al. [85] restrict the exploration and exploitation process using external knowledge and, thus, fall into the second category of SRL methods. By doing so, they have been shown to achieve collision-free behavior policies that perform better than rule-based agents. However, they do not modify the underlying MDP problem formulation, which means that the agent does not learn to avoid choosing unsafe actions. Their approach is built upon an architecture similar to Simplex architectures as a safety behavior is used to restrict the action space of the RL agent.

Cheng et al. [23] propose a combination of a model-free RL-based controller with a model-based controller utilizing control barrier functions and online learning of the unknown system dynamics to ensure safety during learning. They propose two different architectures for exploration and exploitation and have shown that their approach can generate collision-free behaviors in either operational mode. They do not incorporate the controller's corrections in the learning process, which means that the model-free RL-based controller cannot adapt its behavior. Their algorithm also falls into the second category of SRL approaches.

Zhou et al. [130] introduce an architecture where a learned behavior policy is paired with a model-based safety controller. The safety controller predicts whether the trained behavior policy will lead the system to an unsafe state and take over

control when necessary. They also propose to repair the trained policy using data produced by the safety controller at runtime to deviate minimally from the original policy. Their approach changes the optimality criterion by including external data to the SRL algorithm and restricts the exploration and exploitation process. However, repairing the policy using data generated by the safety controller might deteriorate the behavior policies' performance as safety controllers usually do not focus on producing performant behaviors.

3.2.2 Runtime Safety Assurance

This section outlines the state-of-the-art of RTSA frameworks applied to autonomous systems. With learning-based methodologies being applied for the behavior generation of autonomous systems, RTSA frameworks are a popular choice for assuring black-box systems' safety.

The *Simplex architecture* is a widely used RTSA architecture that has the underlying fundamental idea of "using simplicity to control complexity" [97]. The Simplex architecture consists of two subsystems: the *high-performance* and *high-assurance* controller. The high-performance controller can be any controller, e.g., black-box approaches that use DNNs. In the nominal case, the system is controlled by the high-performance controller. The high-assurance controller is developed using standard norms and conventional methodologies. A *decision logic* switches between these two operation modes based on some predefined logic. For example, the decision logic could switch from the high-performance to the high-assurance controller if the high-performance controller's control errors are larger than those of the high-assurance controller. Typically, a recovery region is defined in which it is still possible for the high-assurance controller to take over control safely. In complex, dense, real-world traffic scenarios, such region definitions become non-trivial, often requiring overapproximations. These overapproximations will limit the system's performance, as the high-assurance controller will take over control too early and often. Further, if operating in highly complex and dynamic environments, it is non-trivial to design safety controllers that perform better and safer than the high-performance ones. Some approaches utilizing Simplex architecture are discussed in the following.

SOTER provides a programming language for implementing and testing high-level reactive robotics software and an integrated RTSA framework that enables complex software stacks to be constructed as compositions of RTSA modules [25]. In SOTER, each component in the system is an RTSA module that makes sure that the sub-system operates within predefined bounds. They argue that reachability analysis is not feasible as this requires exact models of the environment. However, as they demonstrate their approach only in static environments, they would also require a methodology that defines the used "safe" and "safer" regions for dynamic environments.

Mehmood et al. [73] introduce a modified Simplex architecture in which the high-performance controller's output is not directly fed into the system but instead into a lookahead baseline controller. The high-performance controller controls the

system as long as the baseline controller can recover the system. If the system is in an operational state where the baseline controller fails to compute a solution, the decision module can still recover the system using the safe command sequence from the previous step. In their work, they provide an example of an aircraft and the feasibility of the proposed approach. However, in more dynamic environments where objects are very close to each other, such as autonomous driving, the previously computed safety trajectory might not be valid in the next time step.

Lazarus et al. [66] also build upon the Simplex architecture but frame the RTSA as an MDP problem formulation and propose to learn a meta-controller for switching between the high-performance and high-assurance controller. To avoid not being able to certify the learned controller by using DNNs, they resort to using a linear value function approximation for the used Q-learning algorithm. The overall approach has been shown to outperform conventional, hand-crafted methodologies.

The approaches above do not specify how safe-regions for dynamic systems can be defined but provide overall RTSA frameworks. In the following, an overview of methods is given that aim at providing safe regions for dynamic environments, such as autonomous driving.

Pek et al. [86] present a formal verification technique for guaranteeing legal safety in arbitrary urban traffic situations at runtime. They define legal safety as never causing an accident under the premise of other traffic participants being allowed to perform any behavior following the traffic rules. Their proposed methodology is built on three pillars. First, they perform an *online situation assessment* in which the situation is assessed, and all legal future evolutions are predicted while also accounting for measurement uncertainty. Second, *fail-safe operation* computes a safety trajectory that can transfer the system from the current operational-state to a safe-state. Third, *correct by construction* ensures that the autonomous vehicle operates in compliance with legal safety at all times regardless of the used motion planning framework. A downside of this approach might be a too restrictive behavior due to predicting all legal future evolutions of the scenario and that the other vehicles do often not adhere to the traffic rules as shown in [30].

Shalev-Shwartz et al. [98] introduced the responsibility-sensitive safety (RSS) as a mathematical model for RTSA. It formalizes the interpretation of "duty of care" from tort law that states how an individual should exercise "reasonable care" while performing acts that could harm others. The RSS formalizes the following five "common sense" rules. The first one, "Do not hit someone from behind.", stipulates the need for not hitting another traffic participant from behind as this would the fault of the autonomous vehicle. The second one, "Do not cut-in recklessly.", tries to avoid dangerous lane-changing situations and the third one, "Right-of-way is given, not taken.", mitigates the issue of vehicles aggressively trying to enforce the right-of-way rule. Another common-sense rule is "Be careful in areas with limited visibility.", which makes sense in areas, such as intersections or occluded areas with parking vehicles. However, it is open for discussion what the authors understand by the phrase "be careful" and it is not straightforward to implement such a quantity. The last one of the five rules "If you can avoid an accident without causing another one,

you must do it." states that the duty shall be to avoid an accident even if you are not at fault. The authors formally prove and empirically have shown that when applying the RSS model, the required 10^{-8} probability of severely injured traffic participants per hour of driving is achieved. The authors show that if all vehicles would adhere to the RSS, an "utopia" could be achieved having absolutely no collisions. The RSS faces similar issues as [86] by possibly restricting the configuration space too strictly as it assumes worst-case behaviors of others.

In summary, RTSA frameworks play a crucial role in making learned behavior policies utilizable in safety-critical applications, such as autonomous driving — at least up to a point, where DNNs can be verified and guaranteed to be safe on their own.

Section 3.3 introduces the CBPE that evaluates learned behavior policies using counterfactual worlds – worlds in which others behave non-factual. The CBPE can be employed in the decision logic of a Simplex architecture to switch between the high-performance and high-assurance behavior policy. Using the policies' performance to restrict its usage is potentially less restrictive than the above-outlined reachability-based approaches.

3.2.3 Combining Conventional and Learning-Based Methodologies

Utilizing learning-based behavior policies in conventional methods, such as optimization- or search-based approaches that have been outlined in Section 1.2 can also make learning-based behavior policies applicable in safety-critical applications. In the following, methodologies combining these classes are outlined, and the synergies that this creates are discussed.

AlphaGo was one of the first approaches that combined state-of-the-art RL with a Monte-Carlo tree search (MCTS) that achieved better than human performance playing the game Go — even beating the current world champion Lee Sedoul [103]. In follow-up work, they introduced AlphaZero that did not require any prior knowledge as AlphaGo did [100]. Similar concepts combining search-based approaches with machine learning have been developed and applied for autonomous vehicles' behavior generation.

Hoel et al. [49] combine MCTS with RL for tactical decision making that is based on the AlphaGo and AlphaZero algorithms and extend these to the continuous domain. They use trained DNNs to guide the MCTS to relevant regions of the search tree and the MCTS is used to improve the training process of the DNN behavior policy.

Wang et al. [120] propose a neural rapidly-exploring random tree* (RRT*) that uses a non-uniform sampling by using a learned behavior policy. They train a convolutional neural network and use samples generated from an A* algorithm to learn optimal solutions. The learned distribution is then used in the expansion (sampling) process of the RRT*. They show that challenging scenarios can be solved online.

Bernhard et al. [14] propose an experience-based heuristic-search algorithm that integrates learned Q-functions into a hybrid A* planner. The learned policy is used as a guiding heuristic for the A* planner to boost the online capability and aims at overcoming the statistical failure rate of deep RL.

Optimization-based methodologies can provide optimal solutions for the problem formulation in Equation 1.1. With global optimization being computationally expensive and local optimization methods requiring "well-performing" initial estimates, such can be obtained using learned behavior policies.

Pan et al. [84] investigate convergence properties of trajectory optimization and learn whether a collision-free solution can be obtained with a given initialization. However, for autonomous driving, there are many other factors required, such as if the learned behavior policy reaches its defined goals, interacts with other traffic participants, and more.

Koller et al. [61] present SAFEMPC, a safe model predictive control (MPC) scheme that guarantees the existence of feasible return trajectories to a safe region of the state space at every time step with high probability. In their work, they combine ideas from policy-based RL with ideas from the control domain.

In Section 4.4, a post-optimization for learned behavior policies is introduced. By iteratively calling the learned behavior policy, an initial trajectory is obtained, which is used as an initial estimate and to derive state constraints. The comfort is then maximized while adhering to the extracted constraints and additionally enforcing the optimized solution to be close to the initial estimate for the interactions with other vehicles to remain valid.

3.3 Counterfactual Behavior Policy Evaluation

This section introduces a CBPE that uses counterfactual (non-factual) worlds to analyze the performance of learned behavior policies at runtime based on the work presented in [44]. Distributional shifts and generalization capabilities can cause unsafe driving behavior when using learned behavior policies that utilize DNNs as shown in the evaluation in Section 5.2.4. Full coverage in offline V&V cannot be achieved due to the vast parameter space real-world traffic scenarios span — motivating the need of RTSA frameworks. These "only" have to evaluate the current scenario and to estimate whether the learned behavior policy is safe in the current scenario. State-of-the-art RTSA frameworks have been discussed in detail in Section 3.2.2. Further, as most of these RTSA frameworks are built upon the concept of Simplex architectures, they require evaluation methods for deciding whether the learned behavior policy produces safe operational states for the current situation. Many approaches employ reachability analysis [86, 98, 49]. However, these might restrict the free configuration space too strictly, assuming all possible reachable sets or worst-case scenarios.

The CBPE evaluates the performance of learned behavior policies using counterfactual assumptions of the current scenario and by predicting these into the future. As the other vehicles behave not nominal (non-factual) in these, generalization capa-

bilities of the learned behavior policy and how well it can handle distributional shifts can be evaluated at runtime.

Definition 1 (Non-factual Behavior) *Behavior policy π' that differs from the actual assumed behavior policy π by parameter- or distributional-shifts that lead to a future outcome of the scene that differs from the assumed one.*

The CBPE allows raising and answering counterfactuals, such as "What would have happened, if vehicle v_i had behaved differently?". The proposed CBPE methodology can be used in the decision logic of a Simplex architecture. Restricting the policies' usage based on its performance is potentially less restrictive than deploying reachability analysis as the policies' performance is only restricting the free configuration space.

First, a definition of counterfactual worlds is given, and it is outlined how these are generated. Next, the forward simulation and evaluation of the counterfactual worlds at runtime are discussed. Finally, it is outlined how the CBPE can be used to extract correlations between the vehicles' behavior policies.

3.3.1 Definition of a Counterfactual World

A *counterfactual world* is derived from the actual (assumed) world. In a counterfactual world, one or multiple behavior policies of other traffic participants are exchanged. In the following, it is assumed that the intentions of others (e.g., whether they want to change lanes) are embedded in the vehicle's behavior policy π.

A world W is defined to be parameterized by a set of vehicles $\mathcal{V} = (v_0, v_j, \dots, v_N)$ with N being the number of vehicles and by a set of behavior policies $\mathcal{B} = (\pi_0, \pi_m, \dots, \pi_M)$ with M being the number of behavior policies. There can be infinite many behavior policies in set $\mathcal{B}$ if the behavior policies are, e.g., sampled stochastically. A complete world parametrization $W = ((v_0, \pi_0), \dots, (v_{ego}, \pi_{ego}), \dots, (v_N, \pi_M))$ is given by a set of tuples pairing the vehicles with a respective behavior policy with one of the N vehicles being the ego vehicle v_{ego} that is controlled by the behavior policy π_{ego}. As described in Section 2.1.4, a behavior policy $\pi(a_t|s_t)$ outputs an action a_t given a world state $s_t = \rho(W_t)$ with ρ transforming the semantic world W_t at time t into a machine learning usable input representation as discussed in Section 2.2. A counterfactual world $W_t^{v_i \sim \pi_j}$ denotes a world where the i-th vehicle is controlled by the behavior policy π_j at time t — either by its original behavior policy or a non-factual one. The notation for multiple behavior models being exchanged is denoted, e.g., by $W_t^{v_0 \sim \pi_1, v_1 \sim \pi_2}$ where vehicle v_0 is controlled by the behavior policy π_1 and vehicle v_1 by the behavior policy π_2.

As is shown in the variational studies in Section 5.2.4, learned behavior policies using DNNs can only cope with deviations in the other vehicle' behaviors up to a certain degree between training and execution. Such deviations can be induced in the set of behavior policies $\mathcal{B}$ by either defining these deterministically or sampling these stochastically. These deviations allow for evaluating the learned behavior policy at

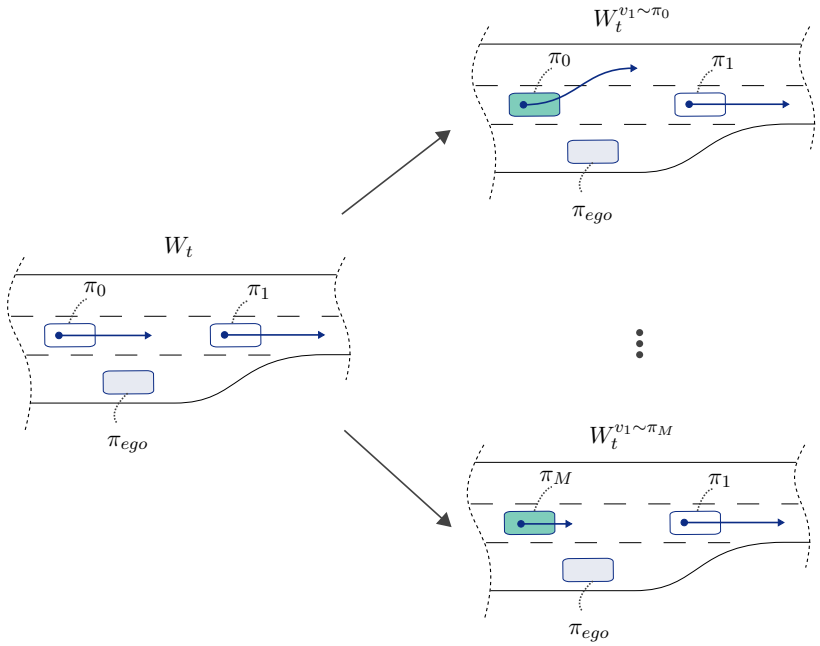


Figure 3.2: The actual (assumed) world W_t is depicted on the left. In the counterfactual world $W_t^{v_1 \sim \pi_0}$, vehicle v_1 (depicted in green) is controlled by the behavior policy π_0 and performs a lane change to let the ego vehicle v_{ego} merge. In the counterfactual world $W_t^{v_1 \sim \pi_M}$, vehicle v_1 decelerates possibly letting the ego vehicle merge as well (modified graphic from [44], ©2020 IEEE).

runtime using counterfactuals, such as "If the other vehicle had behaved slightly differently, would a collision have occurred?".

Both – a deterministically and stochastically defined set of behavior policies $\mathcal{B}$ – have advantages when applied for evaluating the performance of learned behavior policies. As argued in Section 1.2, stochastic methods require a large number of samples to provide statistically significant results. The required large number of samples makes these unable to generate statistically significant results at runtime. However, stochastic sampling avoids hand-crafted features that might not consider certain aspects, such as unknown-unknowns. In this work, a deterministically defined set of behavior policies $\mathcal{B}$ is used to model pre-defined deviations. Knowing the behavioral parameters of other traffic participants during training, deviations can be precisely modeled.

3.3.2 Counterfactual Behavior Policy Evaluation at Runtime

The counterfactual worlds defined in the previous section alone are not sufficient to evaluate learned behavior policies. A forward simulation of these counterfactual worlds is required to obtain traces of actions and states for each vehicle v_i that are

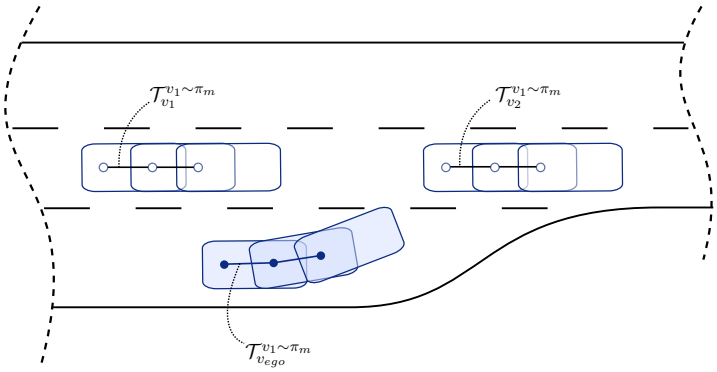


Figure 3.3: Forward simulation of the counterfactual world $W_t^{v_1 \sim \pi_m}$ where vehicle v_1 is controlled by the behavior policy π_m. The traces for each vehicle of the forward simulation are shown with, e.g., the ego vehicle's trace being denoted as $\mathcal{T}_{v_{ego}}^{v_1 \sim \pi_m}$.

used to evaluate the behavior policies. The learned behavior policy can be evaluated in terms of its safety, performance, and other criteria using the evolution and the traces of all vehicles.

Counterfactual worlds are derived from world W_t using the set of behavior policies $\mathcal{B}$ to evaluate learned behavior policies. The behavior policies of vehicles in the proximity $d(v_i, v_{ego}) \leq r_{max}$ of the ego vehicle v_{ego} are exchanged using the set of behavior policies $\mathcal{B}$. For example, if there are three vehicles $N = 3$ nearby and four behavior policies $M = 4$ in $\mathcal{B}$, the resulting number of counterfactual worlds is $NxM = 12$. These counterfactual worlds are then forward-simulated with the ego vehicle v_{ego} being controlled by the learned behavior policy π_{ego} and traces of the ego vehicle in each counterfactual world are collected. The trace of, e.g., the ego vehicle in the counterfactual world $W_t^{v_0 \sim \pi_m}$ is denoted by

$$\mathcal{T}_{v_{ego}}^{v_0 \sim \pi_m} = W_{t,v_{ego}}^{v_0 \sim \pi_m} \cdots \rightarrow W_{t+T-2,v_{ego}}^{v_0 \sim \pi_m} \rightarrow W_{t+T-1,v_{ego}}^{v_0 \sim \pi_m} \rightarrow W_{t+T,v_{ego}}^{v_0 \sim \pi_m} \tag{3.1}$$

with T being the time horizon of the forward simulation. Figure 3.3 shows the trace of three vehicles in a counterfactual world.

Using the trace of the ego vehicle $\mathcal{T}_{v_{ego}}^{v_0 \sim \pi_m}$ and the traces of nearby vehicles $\mathcal{T}_{v_i}^{v_0 \sim \pi_m}$ of, e.g., the counterfactual world $W^{v_0 \sim \pi_m}$, the performance of the behavior policy π_{ego} can be evaluated for this specific counterfactual world. Evaluating traces $\mathcal{T}_{v_{ego}}^{\square}$ of the ego vehicle v_{ego} generated using all counterfactual worlds, an estimate can be obtained whether the learned behavior policy $\pi(a_t|s_t)$ generalizes to all of these and how it performs in these.

Several metrics can be used to evaluate these obtained traces, such as, e.g., the minimum distance of the ego vehicle v_{ego} to other objects. Using the traces of the ego vehicle v_{ego} of all counterfactual worlds $\mathcal{T}_{v_{ego}}^{\square}$, the minimum distance of the ego vehicle

Algorithm 10 Counterfactual Behavior Policy Evaluation

```
function CPE(W_t, B, ego_vehicle, Δt)
    V = NearbyVehicles(W_t, ego_vehicle)
    T = ∅
    for i in 0...N do
        for j in 0...M do
            W_t^{v_i∼π_j} = GetCounterfactualWorld(W_t, V_i, B_j)
            T^{v_i∼π_j} = ForwardSimulate(W_t^{v_i∼π_j}, Δt)
            T ← T^{v_i∼π_j}
        end for
    end for
    return Evaluate(T)
end function
```

to any object is obtained. Determining the minimum distance in all counterfactual worlds can be mathematically be denoted as

$$d_{v_{ego}}^{any} = \min(d_{v_{ego}}(\mathcal{T}_{v_{ego}}^{\square})) \qquad \forall\ W_t^{\square} \tag{3.2}$$

with $W_t^{\square}$ indicating all counterfactual worlds and $d_{v_{ego}}(\cdot)$ returning the minimum distance. $d_{v_{ego}}^{any} = 0$ indicates a collision of the ego vehicle v_{ego} with an object in one of the counterfactual worlds.

The performance of the ego vehicle's learned behavior policy over all counterfactual worlds can then, e.g., be utilized in the decision logic of a Simplex architecture. If, e.g., the safety distance is maintained in all counterfactual worlds, the learned behavior policy is safe to be executed. As others' behavior policies are unknown, a counterfactual world could well be the actual world, reinforcing the need for the learned behavior policy to handle all counterfactual worlds to be considered safe. The full CBPE algorithm is outlined in Algorithm 10. In the 'Evaluate' function, e.g., Equation 3.2 can be utilized.

3.3.3 Insights into Learned Behavior Policies

Apart from evaluating the traces $\mathcal{T}$ as described in the previous section, the CBPE can also be used as a diagnostic tool to gain insights into learned behavior policies. Correlations between the vehicles' behavior policies in a scenario can be extracted using the CBPE. Understanding the correlations can foster the understanding of encoded driving behaviors in learned behavior policies. In perception, methods that provide additional insights into learned DNNs are available, such as saliency maps [104]. Similar frameworks are required for learned behavior policies to make these applicable in safety-critical applications and better understand the encoded driving behavior.

The influence of assigning a different behavior policy to a single vehicle onto all other vehicles in the scene can be measured using the CBPE. For example, it can

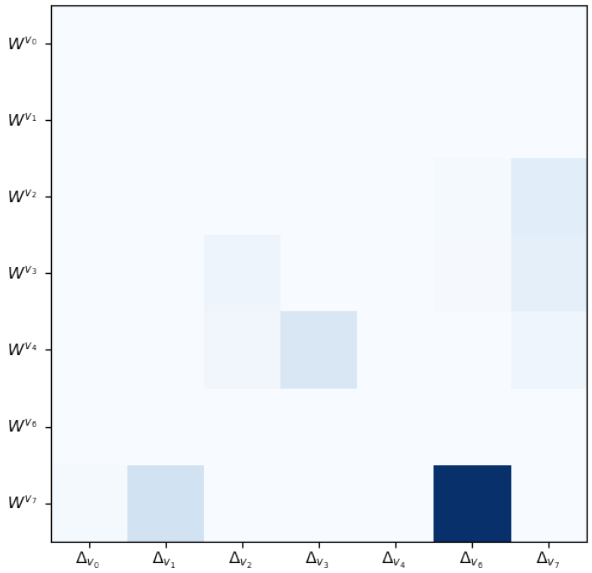


Figure 3.4: Influence heatmap how the behavior policies influence each other. The counterfactual worlds are plotted on the y-axis and the vehicle's state influence is plotted over the x-axis.

be evaluated what happens if a vehicle in the scene brakes instead of driving with a constant velocity and which surrounding vehicles it influences. The traces that have been introduced in the previous section can be used for the evaluation of these effects. Therefore, a function ψ_{states} is introduced that returns a vector $\underline{x}_t$ that consists of the concatenated states of all vehicles of a world W_t at time t ordered by the vehicle's IDs. For a trace $\mathcal{T}$, the function ψ_{states} returns a matrix with each row belonging to a world time t in ascending order in form of $\underline{x} = [\underline{x}_t, \ldots, \underline{x}_{t+T-1}, \underline{x}_{t+T}]$. A ground-truth is obtained by forward-simulating the actual world W_t with the assumed behaviors. The ground-truth trace of vehicle v_i is denoted as $\mathcal{T}_{v_i}$ and the same notation for the counterfactual world traces from the previous section provided in Equation 3.1 is used. The deviation between, e.g., the ego vehicle's states in the counterfactual and ground-truth world is defined as

$$\delta_{v_{ego}}^{v_i \sim \pi_m} = ||\psi_{states}(\mathcal{T}_{v_{ego}}^{v_i \sim \pi_m}) - \psi_{states}(\mathcal{T}_{v_{ego}})||_2. \tag{3.3}$$

The mean influence of replacing the behavior policy of vehicle v_i onto the ego vehicle v_{ego} can be denoted by

$$\Delta_{v_{ego}}^{v_i} = \frac{1}{M} \sum_{j=0}^{M} \delta_{v_{ego}}^{v_i \sim \pi_j}. \tag{3.4}$$

By evaluating every possible combination between all vehicles, an influence (correlation) heatmap can be obtained showing how the behavior policies influence each other as shown in Figure 3.4.

Independent behavior policies do not react to changes in their environment, such as to the preceding vehicle. Using independent behavior policies allows for a clean extraction of how exchanging a specific behavior policy influences the other vehicles. The downside is that independent behavior policies tend to cause collisions if larger simulation horizons are being used.

Obtaining information on how the vehicle's behavior policies influence each other can be an essential criterion for RTSA. If, e.g., a learned behavior policy is not being influenced by a nearby vehicle – or by changes in the nearby vehicle's behavior policy – this could potentially indicate dangerous and unsafe behavior. The proposed runtime CBPE is evaluated in Section 5.3 in terms of the insights it provides in the learned behavior policy.

4 Optimization Theory and Post-Optimizing Behavior Policies

This chapter outlines the theory of gradient-based optimization, discusses trajectory optimization, and introduces a post-optimization for learned behavior policies to obtain smooth trajectories.

An introduction and overview of gradient-based optimization methodologies are provided, and a division into local and global approaches is discussed. Constrained optimization problem formulations and their corresponding solution classes are introduced. After the theory has been laid out, a theoretical view on trajectory optimization is provided — ranging from the used vehicle models, over numerical methods, over the problem formulation, to optimization solution classes.

A novel post-optimization is introduced that utilizes learned behavior policies to obtain initial estimates and derive constraints from these. As learning-based methodologies use imperfect function approximators, such as deep neural networks (DNNs) the resulting behaviors often do not provide smooth behaviors. The trajectories' smoothness is vital as an underlying trajectory-following controller might otherwise not be able to minimize the control error — possibly leading to unsafe behaviors. The post-optimization minimizes the jerk of the trajectories generated by the learned behavior policy while adhering to the extracted constraints — increasing the comfort while being able to guarantee the same safety level. Moreover, by enforcing the optimized trajectory to be close to the learned trajectory, the learning-based behavior policies' interactions remain valid. Further, using the learned behavior policy, an optimum of the nonconvex problem is chosen, mitigating optimizing multiple to obtain good performance.

4.1 Introduction to Optimization

Optimization is a widely used tool that allows the formulation and solution of a large number of problems. This chapter provides an overview of optimization methods. An optimization problem with equality and inequality constraints can be formulated as

$$\text{minimize } f_0(x) \tag{4.1}$$

$$\text{subject to } f_i(x) \leq a_i, \qquad i = 1, \ldots, m \tag{4.2}$$

$$h_i(x) = b_i, \qquad i = 1, \ldots, r \tag{4.3}$$

with $x = (x_1, \ldots, x_n)$ being the optimization vector, $f_0 : \mathbb{R}^n \to \mathbb{R}$ being the objective function, $f_i : \mathbb{R}^n \to \mathbb{R}$ being the inequality constraints, $h_i : \mathbb{R}^n \to \mathbb{R}$ being the equality

constraints, and $a_1, \ldots, a_i$ and $b_1, \ldots, b_i$ being the bounds for the constraints. A problem where the objective function $f_0(x)$ and the constraint functions $f_i(x)$ satisfy

$$f_i(\alpha x + \beta y) \leq \alpha f_i(x) + \beta f_i(y) \tag{4.4}$$

for all $x, y \in \mathbb{R}^n$ and all $\alpha, \beta \in \mathbb{R}$ with $\alpha + \beta = 1, \alpha \geq 0, \beta \geq 0$ is called a convex optimization problem. A fundamental property of convex optimization problems is that any locally optimal point is also globally optimal [16]. Thus, given a convex problem, local optimization methods can be used rather than global ones — making it possible to use local optimization to obtain the global optimum. Local optimization yields the advantage of being much less computationally expensive and offering faster convergence. However, many real-world problems are nonconvex, and the initial estimate is of high importance when employing local optimization. If the objective function $f_0(x)$ and the constraints $f_i(x)$ are nonlinear and convex, the optimization problem is called *nonlinear optimization*. In autonomous driving, nonlinear vehicle models and nonconvex constraints are often used, resulting in nonconvex optimization problems having multiple local minima. Solving nonlinear optimization problems is nontrivial, and there are no effective methods for solving the general nonlinear programming problem [16]. There are two main solution classes for nonlinear optimization problems: local and global optimization.

In *local optimization methods*, a compromise is taken by not seeking the global optimal optimization vector x^*, which minimizes the objective function $f_0(x)$ over all feasible points. Instead, x^* only will be locally optimal. Local optimization makes it possible to solve time-consuming problems, such as large-scale problems, rather quickly. A good initial starting point or estimate is crucial in local optimization as it can significantly affect the obtained solutions [16].

In *global optimization methods*, the globally optimal optimization vector x^* is obtained over all feasible points. However, in the worst-case, the complexity of global optimization methods grows exponentially with the problem size [16]. Due to this complexity and the requirement of trajectory planning for autonomous driving to be real-time capable, global optimization is not always feasible to be used in real-time.

All optimization solution methods minimize an objective function $f_0(x)$ that represents some cost. Local solution methods can further be divided into constrained and unconstrained optimization. The theory of trajectory optimization for autonomous vehicles is outlined in Section 4.3.

4.2 Constrained Optimization

Many optimization problems require the optimization vector x to be constrained — e.g., to restrict the steering rate or a particular state of an autonomous vehicle.

Newton's method is outlined that can solve problems having equality constraints. Finally, interior-point methods are discussed that are capable of handling equality as well as inequality constraints. Figure 4.1 depicts a constrained optimization problem having an objective function f_0 and an inequality constraint function f_1.

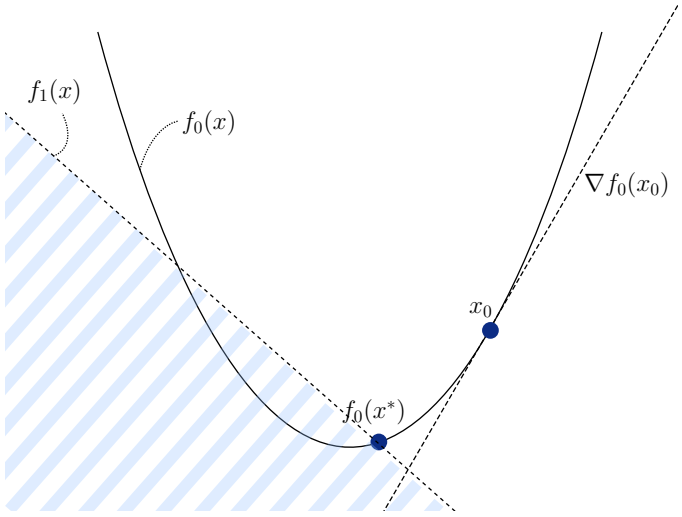


Figure 4.1: Constrained optimization problem with f_0 being the objective and f_1 the constraint function.

4.2.1 Constrained Newton's Method

A widely used method for equality constrained optimization problems is the constrained Newton's method. An optimization problem that is only constrained by equality constraints is given by

$$\text{minimize } f_0(x) \tag{4.5}$$
$$\text{subject to } Ax - b = 0. \tag{4.6}$$

The constrained Newton's method can be derived starting from the Taylor approximation. The infeasible start Newton's method can be derived starting from the optimality conditions $Ax^* = b$ and $\nabla f(x^*) + A^T w = 0$. The aim is to find a step Δx so that $x + \Delta x$ satisfies or least approximately satisfies the optimality conditions. Therefore, x^* is substituted with $x + \Delta x$ in the optimality condition, and a first-order approximation is used. The resulting equations are then given by

$$\nabla f_0(x) + \nabla^2 f_0(x)\Delta x + A^T w = 0 \tag{4.7}$$
$$A(x + \Delta x) - b = 0. \tag{4.8}$$

This can also be expressed in matrix form as

$$\begin{bmatrix} \nabla^2 f(x) & A^T \\ A & 0 \end{bmatrix} \begin{bmatrix} \Delta x \\ w \end{bmatrix} = -\begin{bmatrix} \nabla f(x) \\ Ax - b \end{bmatrix} \tag{4.9}$$

with the matrix on the very left being called the Karush-Kuhn-Tucker (KKT) matrix. The Newton step is only defined when the KKT matrix is not nonsingular as the

matrix is not invertible otherwise. By choosing an initial optimization vector x and a dual variable $w \geq 0$ and by solving for Δx and w, the constrained optimization problem can be solved. The Newton step is given by

$$\begin{bmatrix} \Delta x \\ w \end{bmatrix} = - \begin{bmatrix} \nabla^2 f(x) & A^T \\ A & 0 \end{bmatrix}^{-1} \begin{bmatrix} \nabla f(x) \\ Ax - b \end{bmatrix}. \tag{4.10}$$

As with the unconstrained Newton's method, there can either be a pure or a damped constrained Newton's method using, e.g., backtracking line-search.

4.2.2 Interior-Point Methods

Interior-point methods are another solution class for constrained optimization problems. The barrier method is an interior-point solution method with a relatively straightforward concept using a barrier term to contain the optimization vector within a defined feasible region. A barrier optimization problem can be denoted in

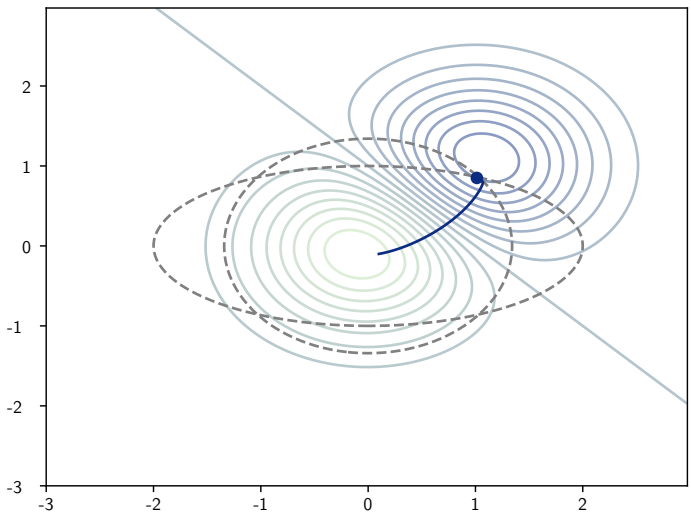


Figure 4.2: Depiction of the solution process of an interior-point method. The constraints are illustrated as dashed ellipses and the objective function is shown using contours. The blue line visualizes the solution trajectory.

the following form

$$\text{minimize } f_0(x) + \sum_{i=1}^{m} I_{-}(f_i(x)) \tag{4.11}$$

$$\text{subject to } Ax = b. \tag{4.12}$$

with, e.g., an indicator function, such as

$$I_{-}(u) = \begin{cases} 0, & \text{if } u \leq 0 \\ \text{inf}, & 0 \end{cases}. \tag{4.13}$$

A choice is to use a logarithmic indicator function $I_{-}(u) = -(1/t)\log(u)$ with $t > 0$. These methods are then commonly referred to as log-barrier methods. It is empirically shown that optimizing a sequence of optimization problems of Equations 4.12 with an increasing t works well [16]. Figure 4.2 depicts a barrier constrained optimization problem having two inequality constraints in the form of ellipses. The resulting barrier optimization problem is then given by

$$\text{minimize } tf_0(x) + \phi(x) \tag{4.14}$$

$$\text{subject to } Ax = b. \tag{4.15}$$

with $\phi(x) = -\sum_{i=1}^{m} \log(-f_i(x))$.

As discussed in Section 4.2.1, the optimization problem in Equations 4.14-4.15 is an equality constrained problem and can, e.g., be solved using the Newton's method. Barrier optimization problems are often referred to as central path methods as they start in the center of convex polytopes. More insights on central path methods are provided in [16].

Primal-dual interior-point methods are often more efficient than barrier methods, especially when high accuracy is required [16]. They can be derived starting from the modified KKT conditions that are given by

$$r_t(x, \lambda, \nu) = \begin{bmatrix} \nabla f_0(x) + Df(x)^T\lambda + A^T\nu \\ -\mathbf{diag}(\lambda)f(x) - (1/t)\mathbf{1} \\ Ax - b \end{bmatrix} \tag{4.16}$$

with $t > 0$ and

$$f(x) = \begin{bmatrix} f_1(x) \\ \vdots \\ f_m(x) \end{bmatrix}, Df(x) = \begin{bmatrix} \nabla f_1(x)^T \\ \vdots \\ \nabla f_m(x)^T \end{bmatrix}. \tag{4.17}$$

The modified KKT conditions in Equation 4.16 consist of a stationary term, relaxed inequality and equality terms for primal feasibility. Expanding Equation 4.16 using a Taylor expansion yields the following system

$$\begin{bmatrix} \nabla^2 f_0(x) + \sum_{i=1}^{m} \lambda_i \nabla^2 f_i(x) & Df(x)^T & A^T \\ -\mathbf{diag}(\lambda)Df(x) & -\mathbf{diag}(f(x)) & 0 \\ A & 0 & 0 \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta\lambda \\ \Delta\nu \end{bmatrix} = -\mathbf{r}_{residuals} \tag{4.18}$$

having the residual vector $\mathbf{r}_{residuals}$ that is given by

$$\mathbf{r}_{residuals} = \begin{bmatrix} r_{dual} \\ r_{cent} \\ r_{primal} \end{bmatrix} = \begin{bmatrix} \nabla f_0(x) + Df(x)^T\lambda + A^T\nu \\ -\mathbf{diag}(\lambda)f(x) - (1/t)\mathbf{1} \\ Ax - b \end{bmatrix}. \tag{4.19}$$

Solving Equation 4.18 yields the search direction Δx for the constrained optimization problem. As the method does not require the iterates to produce feasible points, these cannot be used to evaluate the duality gap at any step besides at convergence. Thus, for the primal-dual interior-point method a surrogate duality gap is used that is defined by

$$\widehat{\eta}(x, \lambda) = -f(x)^T \lambda. \tag{4.20}$$

This surrogate duality is then used as one of the convergence criteria for the interior-point method as outlined in Algorithm 11.

Algorithm 11 Primal-Dual Interior-Point Method [16]

given x that satisfies $f_1(x) < 0, \ldots, f_m(x) < 0, \lambda \succ 0, \mu > 1, \epsilon_{feas} > 0, \epsilon > 0$.
repeat
1. Set $t := \mu m / \widehat{\eta}$.
2. Compute primal-dual search direction.
3. Line-search and update: $x := x + s\Delta x$ with step size s.

until $\|r_{primal}\| < \epsilon_{feas}$, $\|r_{dual}\| < \epsilon_{feas}$, and $\widehat{\eta} \leq \epsilon$

Since the primal-dual interior-point method has a faster convergence than linear convergence, it is common to choose a small ϵ_{feas} and ϵ in Algorithm 11 [16].

4.3 Trajectory Optimization for Autonomous Vehicles

Trajectory optimization methodologies provide optimal solutions to the planning problem formulation stated in Equation 1.1. An overview of the state-of-the-art trajectory optimization methods in autonomous driving has been given in Section 1.2.2. This section provides a theoretical view on trajectory optimization and its solution classes.

Due to the complexity of trajectory optimization problem formulations, most of these can only be solved by employing numerical methods. Numerical methods in trajectory optimization can be divided into two main categories: indirect and direct methods. Indirect methods are based on the calculus of variation used to determine first-order optimality conditions of the original trajectory optimization problem [131]. Direct methods discretize the states and controls of the trajectory optimization problem and employ nonlinear optimization to find optimal solutions [56].

Nonlinear optimization solution methods can be divided into gradient- (also referred to as local) and heuristic-based (also referred to as global) optimization methods [131]. The theory of constrained gradient-based optimization methodologies has been outlined in the previous sections. Gradient-based methods for trajectory optimization utilize an initial optimization vector $\underline{u}$ that provides the initial solution and then is optimized. Heuristic optimization methods perform a search in a stochastic manner instead of a deterministic one [131]. These can find globally optimal solutions, such as genetic algorithms [34], simulated annealing [109], and particle swarm optimization

[58]. However, these are often unable to provide a (globally) optimal solution in real-time for complex environments, such as real-world traffic.

Direct, gradient-based methods are often used due to their effectiveness, robustness, and simplicity [132, 42, 82]. In Section 4.3.3, direct shooting methods are discussed for solving the optimal trajectory optimization problem formulation. In direct shooting methods, the trajectory is approximated using a simulation and the cost of the objective function is minimized while adhering to defined constraints [56].

The single-track vehicle model utilized in this work and described in Section 4.3.1 and numerical integration and differentiation methods are discussed in Section 4.3.2. The presented trajectory optimization theory lays the foundation for the post-optimization of learned behavior policies in Section 4.4.

4.3.1 Dynamic Vehicle Model

In trajectory optimization for autonomous vehicles, a forward simulation is required when using direct methods, such as single- or multiple-shooting. As the resulting trajectory should be executable by the vehicle, its dynamics must be integrated into the trajectory optimization. Being executable by the vehicle can be achieved in two ways: constraining the trajectories' differential values or by using vehicle models. Vehicle models provide more accurate trajectories that can be executed by the vehicle while introducing little to no computational overhead. The computational overhead depends on the used vehicle model. Simplistic vehicle models, such as a single-track model [87] are computationally cheap, whereas more sophisticated ones taking slip and oversteer [89] into account are more expensive. In most nominal operational design domains (ODDs) in which the vehicle operates far away from its physical limits, the single-track model poses a sufficiently good approximation of the vehicle's dynamics.

Figure 4.3 depicts a vehicle with θ being the vehicle angle, l the wheelbase of the vehicle, κ the curvature, v the longitudinal velocity, and δ the steering rate. A no-slip condition is assumed (both wheels have the same longitudinal velocity), leading to the same rotation-rate of the wheels. Using the geometric relation of $\tan(\delta) = l\kappa$ and the dynamic relation of $\dot{\theta} = \omega = \frac{v}{l}$, a differential equation for the change of the vehicle angle θ can be obtained that is given by $\dot{\theta} = v\frac{\tan(\delta)}{l}$. The resulting single-track vehicle model consists of four first-order differential equations that can be expressed in state-space form as

$$\underline{x}_{k+1} = \dot{f}(\underline{x}_k, \underline{u}_k) = \begin{bmatrix} v_k \cos(\theta_k) \\ v_k \sin(\theta_k) \\ v_k \frac{\tan(\delta)}{l} \\ a \end{bmatrix} \tag{4.21}$$

with the state $\underline{x}_k = [x_k, y_k, \theta_k, v_k]$ and the control commands $\underline{u}_k = [\delta_k, a_k]$. For small angles and constant velocities, a linearized form of the single-track model can be derived using the trigonometric apprxoimations $\sin(\theta) \approx \theta$, $\cos(0) \approx 1$, and

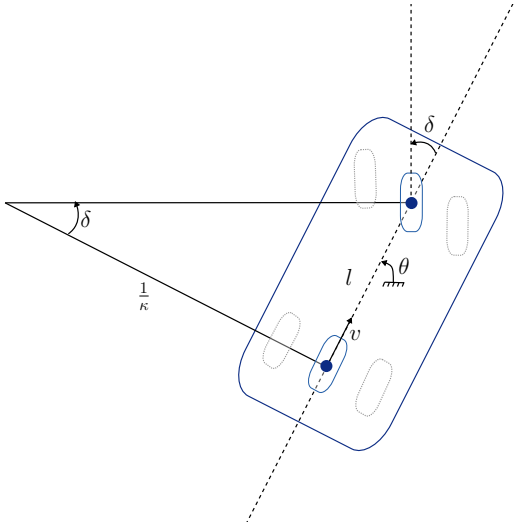


Figure 4.3: Illustration of the single-track vehicle model. The steering rate of the vehicle is given by δ, the curvature by κ, the wheelbase by l, the vehicle's velocity by v, and the vehicle's angle by θ.

$\tan(\delta) \approx \delta$. Assuming the velocity of the vehicle v_k to be constant, the linearized model can be written in matrix-form using the Euler forward integration method as

$$\underline{x}_{k+1} = \underbrace{\begin{bmatrix} 1 & 0 & 0 & \Delta t v_k \\ 0 & 1 & \Delta t v_k & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}}_{\mathbf{A}} \underline{x}_k + \underbrace{\begin{bmatrix} 0 \\ 0 \\ \Delta t \frac{v_k}{l} \\ 0 \end{bmatrix}}_{\mathbf{B}} \underline{u}_k \tag{4.22}$$

with $\underline{x}_k = [x, y, \theta, 1]$ and $\underline{u}_k = [\delta]$. More elaborate linearized models have been proposed using operational regions and lookup tables (LUTs) as in [31].

4.3.2 Numerical Integration and Differentiation Methods

For trajectory optimization in autonomous driving, the resulting trajectory should include the dynamics and non-holonomy of vehicles. Forward-integrating a dynamic vehicle model, as outlined in the previous section, makes sure that the resulting trajectory can be executed by the vehicle.

Time-marching (also referred to as time-stepping) methods calculate the solution of differential equations in each time-step t_k using current and previous information about the solution [131]. These can be divided into two main categories: multiple-step and multiple-stage methods.

In *multiple-step* methods, the solution for time t_{k+1} is obtained from a set of previous values $t_{k-j}, \ldots, t_k$ with j being the number of steps. If $j = 1$, these methods

are referred to as single-step methods, such as the Euler method. The general form of discrete Euler methods can be denoted as

$$\underline{x}_{k+1} = \underline{x}_k + \Delta t(\theta \dot{f}(\underline{x}_k) + (1-\theta)\dot{f}(\underline{x}_{k+1})) \tag{4.23}$$

with $\dot{f}(\cdot)$ being a dynamic model (e.g., the single-track vehicle model) and the values $\theta = (1, 0.5, 0)$ correspond to the Euler forward, Crank-Nicolson, and Euler backward methods, respectively [131]. The Crank-Nicolson and Euler backward methods are referred to as implicit methods as the value $\underline{x}_{k+1}$ also appears on the right-hand side of Equation 4.23. More complex and commonly used multiple-step integration methods are the Adams-Bashforth and Adams-Moulton multiple-step methods [131].

Multiple-stage methods divide the time interval $[t_i, t_{i+1}]$ into K subintervals $[\tau_j, \tau_{j+1}]$ where

$$\tau_j = t_i + \Delta t \alpha_j, j = 1, \ldots, K \tag{4.24}$$

with $0 \leq \alpha \leq 1$. Each τ_j is referred to as a stage. One of the most-popular multiple-stage integration methods is the classical Runge-Kutta method [20].

Implicit, multiple-step, and multiple-stage integration methods are computationally more expensive than, e.g., the single-step Euler forward method. For small integration time-steps and "well-behaved" vehicle models, methods, such as the Euler forward method, suffice to obtain accurate and stable results with direct shooting methods and modern solvers as discussed in Section 4.3.3.

Using the linearized single-track vehicle model as outlined in Equation 4.22, a forward trajectory τ can be obtained as follows:

$$\underline{x}_1 = \mathbf{A}\underline{x}_0 + \mathbf{B}\underline{u}_0 \tag{4.25}$$
$$\underline{x}_2 = \mathbf{A}[\mathbf{A}\underline{x}_0 + \mathbf{B}\underline{u}_0]\underline{x}_1 + \mathbf{B}\underline{u}_1 \tag{4.26}$$
$$\underline{x}_3 = \mathbf{A}[\mathbf{A}[\mathbf{A}\underline{x}_0 + \mathbf{B}\underline{u}_0]\underline{x}_1 + \mathbf{B}\underline{u}_1]\underline{x}_2 + \mathbf{B}\underline{u}_2 \tag{4.27}$$
$$\tag{4.28}$$
$$\vdots \tag{4.29}$$
$$\underline{x}_K = \mathbf{A}[\mathbf{A}\underline{x}_{K-2} + \mathbf{B}\underline{u}_{K-2}]\underline{x}_{K-1} + \mathbf{B}\underline{u}_{K-1} \tag{4.30}$$

with $\mathbf{A}$ being the system matrix and $\mathbf{B}$ the input matrix of the dynamic model. The equation above can also be written in a batch-matrix form allowing for fast computation by, e.g., using graphics processing units (GPUs). In the nonlinear dynamic model case, the Euler forward integration process can be denoted by

$$\underline{x}_1 = \underline{x}_0 + \Delta t \dot{f}(\underline{x}_0, \underline{u}_0) \tag{4.31}$$
$$\underline{x}_2 = \underline{x}_1 + \Delta t \dot{f}(\underline{x}_1, \underline{u}_1) \tag{4.32}$$
$$\vdots \tag{4.33}$$
$$\underline{x}_K = \underline{x}_{K-1} + \Delta t \dot{f}(\underline{x}_{K-1}, \underline{u}_{K-1}) \tag{4.34}$$

resulting in a state-space trajectory $\tau = [\underline{x}_0, \ldots, \underline{x}_K]$.

Besides integration methods, differentiation methods are required in trajectory optimization to, e.g., calculate the jerk of a trajectory. These can either be analytical, rely on complex calculus, or utilize numerical methods. One way to calculate approximate derivatives of a trajectory is finite difference approximation, such as the forward and central differencing methods. The forward difference approximation is given by

$$\frac{df}{dx} \approx \frac{f(x + \Delta t) - f(x)}{\Delta t} \tag{4.35}$$

and the central difference approximation is given by

$$\frac{df}{dx} \approx \frac{f(x + \Delta t) - f(x - \Delta t)}{2\Delta t} \tag{4.36}$$

with Δt being the step-time. It should be noted that small Δt introduce round-off errors in the computation [131]. The central approximation method is more accurate than the forward one but is also computationally more expensive.

A linearization could be performed at every time step to enable such methodologies to be used. Apart from this, numerical integration methods in multi-step and multi-stage are often employed for the forward integration. For simplistic nonlinear vehicle models and sufficiently small integration deltas, even single-step methods, such as the explicit Euler's method, suffice.

4.3.3 Direct Shooting and Nonlinear Trajectory Optimization

Direct methods, such as single shooting, discretize the trajectory optimization problem's controls and states, transforming an infinite optimization problem into a finite one. Direct shooting methods used for trajectory optimization of autonomous vehicles often constitute nonlinear and nonconvex optimization problems as often, either the dynamic model or the constraints, such as e.g., the collision constraints, are nonconvex. Direct methods transform infinite-dimensional optimization problems into finite-dimensional, nonlinear ones making these possible to be solved using nonlinear solvers.

Single shooting methods use a sequence of discrete control commands $\underline{u} = [\underline{u}_0, \ldots, \underline{u}_N]$ and numerical integration methods, such as time-marching as outlined in Section 4.3.2 to obtain a state-space trajectory $\tau = [\underline{x}_0, \ldots, \underline{x}_{N+1}]$ having $N + 1$ states and starting from an initial state $\underline{x}_0$. These can only offer approximately optimal solutions as they approximate the actual infinite-dimensional optimization problem. However, direct methods are widely used due to their robustness and easy applicability [27]. As the first state in the trajectory $\underline{x}_0$ is fixed in single shooting methods, these kinds of optimization problems are referred to as initial value problem (IVP). The trajectory is then optimized starting from the initial state $\underline{x}_0$ over a time-horizon of $[t_{start}, t_{end}]$. It is necessary to integrate the trajectory for each perturbation to calculate gradient information using single shooting methods [15]. In single shooting methods, the optimization vector $\underline{u}$ is used to obtain a trajectory τ. The cost of the trajectory τ is then calculated using an objective function $f_0(\tau, \underline{u})$

that evaluates the trajectory and the input commands. In the unconstrained case and when using gradient-based methods, the objective's cost is then minimized and the process is iteratively repeated until convergence. In the constrained case, the constraints are additionally evaluated in every iteration. Constrained and gradient-based optimization solution methods as outlined in Section 4.2 can be used.

A drawback of single shooting methods is that small changes introduced early in the trajectory can propagate into very nonlinear changes at the end of the trajectory [15]. These nonlinear changes can be mitigated using multiple shooting that has the fundamental idea of breaking the trajectory into shorter pieces or segments. However, this creates the need for additional constraints being introduced that join the segments at the boundaries. As single and multiple shooting solve equivalent nonlinear optimization problems, these have the same discretization error and can utilize the same optimization solution methods, such as sequential quadratic program (SQP) and interior point (IP) methods [27]. Extensive research has led to extremely versatile and robust software programs for numerically solving nonlinear optimization problems [131], such as SQP solvers (OSQP [108], SNOPT [38]) and IP solvers (LOQO [116], IPOPT [119]).

In autonomous driving, the objective $f_0(\tau, \underline{u})$ and the constraints $f_i(\tau, \underline{u}), i \geq 1$ should provide a safe and comfortable trajectory. All safety-relevant terms should be handled by constraints to guarantee collision-free trajectories at convergence. Otherwise, not having collisions would boil down to a parameter-tuning problem in which collisions are weighted more heavily than other terms in the objective. Therefore, in the later introduced post-optimization in Section 4.4, the objective function $f_0(\tau, \underline{u})$ is merely responsible for the comfort while the constraint functions $f_i(\tau, \underline{u}), i \geq 1$ are responsible for adhering to safety-constraints and for the resulting trajectory to be executable by the ego vehicle. When using direct shooting for nonlinear and nonconvex trajectory optimization, the initial optimization vector $\underline{u}$ (initial estimate) is essential as it influences to which local optimum the optimization converges. Choosing poor local optima can either lead to poor performance or even to unsafe behaviors. In this work's proposed post-optimization, the initial estimate is obtained from a learned behavior policy mitigating the problem of optimizing multiple starting points or just a single one, possibly leading to deteriorating performance.

4.4 Post-Optimization of Behavior Policies

Local, gradient-based optimization methodologies' performance greatly depends on the initial estimates — poor initializations lead to poor performing local minima. This section introduces a local post-optimization that utilizes learned behavior policies to obtain interactive initial estimates and derive constraints from this initial estimate. The proposed methodology is based on the work presented in [46], the trajectory optimization theory discussed in Section 4.3, and on the learning-based behavior policies introduced in Chapter 2.

As described by Bender et al. [11], when using local optimization approaches, the combinatorial aspects of behavior generation have to be considered. In their work, they propose a partition of the trajectory space into discrete solution classes, with each of these having a local optimum — so-called *maneuver variants*. To find the global optimum amongst the maneuver variants, each of these has to be optimized. Optimizing all maneuver variants is not always tractable in real-time due to computational constraints. Optimizing a sub-set of the maneuver variants might lead to poor local minima and, thus, to poor performance.

To mitigate this problem and to choose "well-performing" maneuver variants, conventional, e.g., search-based methods can be utilized for generating an initial solution that is then optimized [22]. However, search-based methodologies cannot always provide solutions in real-time due to computational limitations. Several works propose leveraging learning-based methodologies that harness off-line computational power to learn initial estimates [6] and rate their effectiveness [84]. However, these works are of supervised nature and do not take dynamic environments and interactions with other traffic participants into account. This work uses a learned behavior policy π_{ego} for generating initial estimates in a game-theoretic fashion – meaning that all agents choose actions simultaneously and influence each other [46]. Apart from choosing a maneuver variant, a corresponding optimization vector $\underline{u}_{ego}^{init}$ is obtained for warm-starting the optimization reducing the overall computational effort.

Further, learning-based behavior policies do not always provide feasible solutions (e.g., not reaching the goal or causing collisions), which necessitates the need for fail-safe trajectories that can be executed instead. The proposed post-optimization framework can be embedded into runtime safety assurance (RTSA) frameworks as discussed in Section 3.3.2, such as into Simplex architectures [97].

Most conventional trajectory optimization approaches are not game-theoretic and predict all traffic participant's trajectories before the actual optimization. The gap between prediction and optimization can lead to too passive and unsafe behaviors as changes in the optimized trajectory τ_{ego}^{opt} do not influence the other traffic participants τ_i^{opt}. An initial estimate is generated using a game-theoretic learning-based behavior policy that includes interactions with others. The interactions with other traffic participants are assumed to remain valid by enforcing the optimized trajectory to be "sufficiently" close to the learning-based trajectory.

In summary, the proposed post-optimization reduces computational costs by leveraging learned behavior policies and using initial estimates generated by these. It ensures collision-freeness by extracting constraints from the learned solution and enforcing these during optimization. By constraining the optimized trajectory to be close to the initial one, interactions that the learning-based approach had with other traffic participants remain valid. Finally, by minimizing the jerk, the comfort of the optimized trajectories is equal or, in most cases, better than the one of the learning-based solutions as is shown in the evaluation in Section 5.4.

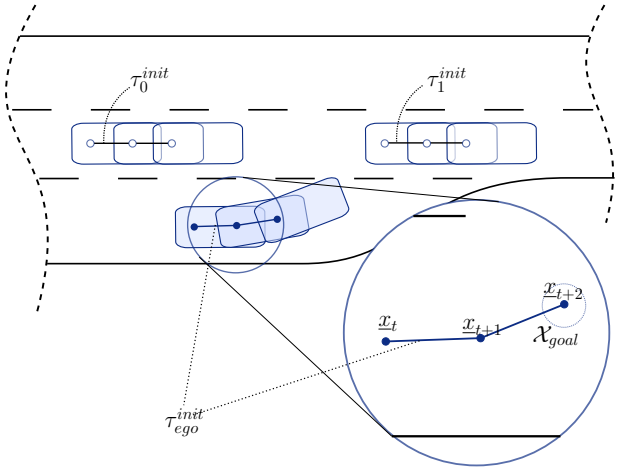


Figure 4.4: Forward simulation of the world for three time-steps in a game-theoretic fashion with all vehicles choosing actions simultaneously. The ego vehicle is depicted in blue and its trajectory is denoted by $\mathcal{T}_{ego}^{init}$. The trajectories of the other vehicles are denoted as $\mathcal{T}_{i}^{init}$.

4.4.1 Initial Estimates and Constraints

The initial estimate is generated by forward-simulating a semantic world (all objects being represented in an object list) in a game-theoretic fashion — where each vehicle chooses an action a_t and influences other vehicles or is being influenced at time t. A semantic world W_t is parameterized by a set of vehicles $\mathcal{V} = [v_0, \ldots, v_N]$ controlled by a respective behavior policy π_n with N being the number of vehicles in the scene. Generating initial estimates is similar to the forward world simulation of the counterfactual behavior policy evaluation (CBPE) as outlined in Section 3.3. In each simulation step, all vehicle's behavior policies are called and return an action a_t, which is then executed. The forward simulation starts at the current world time t_{world} and ends once an evaluator has determined it to be terminal — e.g., if the goal has been reached or a collision has occurred. Figure 4.4 depicts a forward simulation of the semantic world W_t for three time-steps and shows the trajectories of the vehicles.

As the terminal state is determined by an evaluator and the behavior policy π, the time-horizon of the initial trajectory τ_{ego}^{init} can vary significantly and, thus, the optimization's computational complexity. The trajectories' length can especially become an issue when using higher-order methods, such as second-order methods that require the computation of the Hessian matrix and its inverse. Solving for x in an equation $Ax = b$ having a matrix $A \in \mathcal{Z}^{n \times n}$ and $b \in \mathcal{Z}^{n \times 1}$ results in a computational complexity of $O(n^3||A|| + ||b||)$ [29]. Thus, for second-order methods to be used for large time-horizons and in real-time, methods such as the Broyden-Fletcher-Goldfarb-Shanno (BFGS) algorithm have to be utilized to approximate the Hessian matrix.

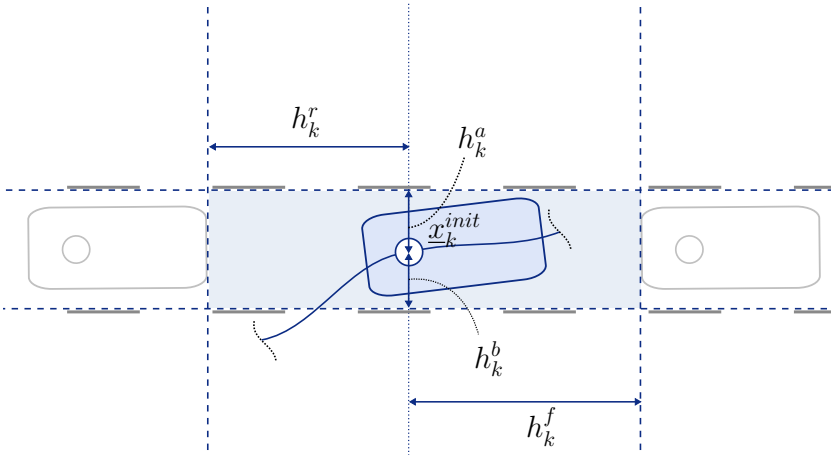


Figure 4.5: Visualization of the constraints for a single time-step k. A line-search along the Cartesian coordinates defines the maximum free configuration space for the ego vehicle (depicted in blue) defined by $h_k^f, h_k^r, h_k^a, h_k^b$.

The initial optimization vector $\underline{u}_{ego}^{init}$ and the resulting trajectory τ_{ego}^{init} of the ego vehicle generated by the learned behavior policy π_{ego} can be sub-divided into feasible and infeasible initial estimates. In the feasible case, the initial estimate brings merit to the optimization as it successfully and in a game-theoretic fashion solves the scenario. Besides choosing a maneuver variant, it also provides an initial optimization vector $\underline{u}_{ego}^{init}$ that corresponds to the maneuver variant and provides a trajectory reaching the goal. In the infeasible case, an RTSA framework can be utilized to plan and execute a fail-safe trajectory as, e.g., proposed in [86].

In the feasible case, state constraints can be derived from the initial trajectory of the ego vehicle τ_{ego}^{init} and by using the other vehicles trajectories τ_i^{init}. As each vehicle's state is obtained during the forward simulation, the maximum free-space of the ego vehicle can be extracted at each time-step k. The extraction of the maximum free-space for the time-step k is shown in Figure 4.5. The state $\underline{x}_k^{init}$ is provided by the learning-based behavior policy being forward-simulated as shown in Figure 4.4. A line-search along the x- and y-coordinates is performed for each state in the initial trajectory $\underline{x}_k^{init}$ to extract bounding boxes relative to the initial states. These are defined by h_k^f representing the distance to the front of the vehicle, h_k^r representing the distance to the rear, h_k^a representing the distance to the top, and h_k^b the distance to the bottom. These four values (h_k^f, h_k^r, h_k^a, h_k^b) then span a bounding box for each discrete time-step k relative to the initial state $\underline{x}_k^{init}$.

In Figure 4.6, the initial state of the ego vehicle is denoted by $\underline{x}_k^{init}$ and the optimized state is denoted as $\underline{x}_k^{opt}$ for a discrete time-step k. The initial state $\underline{x}_k^{init}$ is the reference system for defining the constraits via the maximum deviations $h_k^{\square}$ at time-step k. Using the deviations $\Delta x = x_k^{opt} - x_k^{init}$ and $\Delta y = y_k^{opt} - y_k^{init}$ between the initial state $\underline{x}_k^{init}$ and the optimized state $\underline{x}_k^{opt}$ and geometric approximation functions that project the vehicle shape based on its orientation onto Cartesian coordinates,

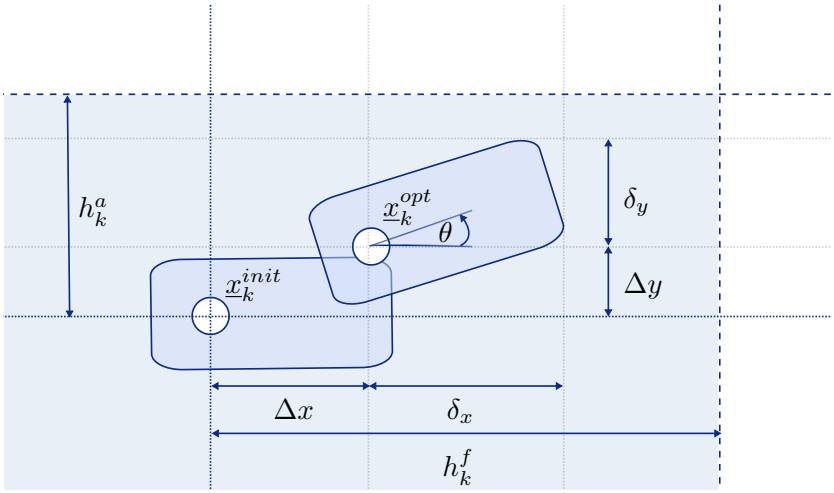


Figure 4.6: Cartesian deviations Δx and Δy of initial state $\underline{x}_k^{init}$ and the optimized state $\underline{x}_k^{opt}$. In the depicted case, the constraints are fulfilled as $\Delta x + \delta_x \leq h_k^f$ and $\Delta y + \delta_y \leq h_k^a$.

relative constraints can be defined. $\delta_x^f(\theta_k^{opt})$ and $\delta_x^r(\theta_k^{opt})$ return the distance to the front and rear of the vehicle shape projected onto the x-coordinate, respectively. $\delta_y^a(\theta_k^{opt})$ and $\delta_y^b(\theta_k^{opt})$ return the distance to the top and bottom point of the vehicle shape projected onto the y-coordinate, respectively.

This results in four constraint functions that can be denoted as

$$f_k^f(\underline{x}_k^{init}, \underline{x}_k^{opt}) = \Delta x_k + \delta_x^f(\theta_k^{opt}) \tag{4.37}$$
$$f_k^r(\underline{x}_k^{init}, \underline{x}_k^{opt}) = \Delta x_k + \delta_x^r(\theta_k^{opt}) \tag{4.38}$$
$$f_k^a(\underline{x}_k^{init}, \underline{x}_k^{opt}) = \Delta y_k + \delta_y^a(\theta_k^{opt}) \tag{4.39}$$
$$f_k^b(\underline{x}_k^{init}, \underline{x}_k^{opt}) = \Delta y_k + \delta_y^b(\theta_k^{opt}). \tag{4.40}$$

Equations 4.37-4.40 define the maximum allowed deviation or free configuration space at time-step k. However, if the deviations are too large from the learning-based initial solution, the interactions with other traffic participants might not remain valid. Thus, in the post-optimization problem formulation presented in the next section, an additional trust-region constraint is introduced to enforce proximity between the initial and optimized trajectories. The state-wise trust-region is illustrated in Figure 4.7.

4.4.2 Post-Optimization Problem Formulation

As described in Section 4.3, two characteristics are essential for behavior generation in autonomous driving: safety and comfort. The comfort shall be maximized while adhering to the safety constraints to obtain an optimal solution for the optimal trajectory planning problem given in Equation 1.1. The previous section discussed

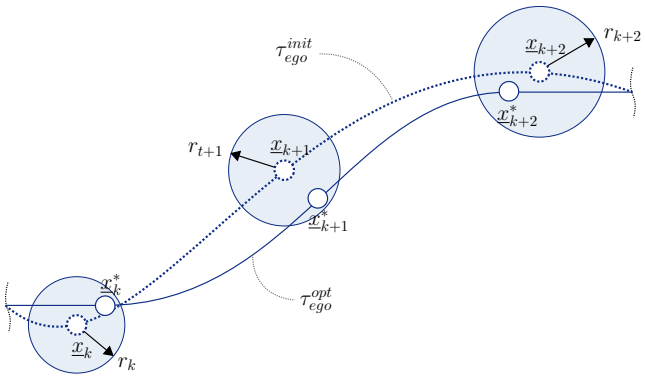


Figure 4.7: Proximity of the optimized trajectory τ_{ego}^{opt} to the learning based initial trajectory τ_{ego}^{init}.

how the initial optimization vector $\underline{u}^{init}$ and the initial learning-based trajectory of the ego vehicle τ_{ego}^{init} are obtained and how relative state-constraints to the initial trajectory can be defined. This section introduces and discusses the post-optimization problem formulation for learned behavior policies.

The proposed post-optimization is a direct method that utilizes single shooting. Therefore, the trajectory τ_{ego}^{opt} is obtained by sequentially integrating the single-track vehicle model introduced in Section 4.3.1 using, e.g., a time-marching integration method as outlined in Section 4.3.2. As the trajectory is generated using a vehicle model, the dynamic constraints do not have to be included explicitly in the problem formulation but are implicitly included due to forward-integrating the vehicle model as described in Section 4.3.2. The time-horizon of optimized trajectory τ_{ego}^{opt} is determined by the length of the initial trajectory τ_{ego}^{init} provided by the learned behavior policy π_{ego}. The constrained, single-shooting, nonlinear optimization problem formulation can then be denoted as

$$\begin{aligned}
&\text{minimize } f_0(\tau_{ego}^{opt}) && && (4.41)\\
&\text{subject to } f_k^f(\underline{x}_k^{init}, \underline{x}_k^{opt}) \leq h_k^f, && k = 1, \dots, N && (4.42)\\
&f_k^r(\underline{x}_k^{init}, \underline{x}_k^{opt}) \leq h_k^r, && k = 1, \dots, N && (4.43)\\
&f_k^a(\underline{x}_k^{init}, \underline{x}_k^{opt}) \leq h_k^a, && k = 1, \dots, N && (4.44)\\
&f_k^b(\underline{x}_k^{init}, \underline{x}_k^{opt}) \leq h_k^b, && k = 1, \dots, N && (4.45)\\
&||\underline{x}_k^{init} - \underline{x}_k^{opt}||_2 \leq r_k, && k = 1, \dots, N && (4.46)\\
&f_k(\underline{u}_k^{opt}) \leq \underline{u}_k^{max} && k = 1, \dots, N-1. && (4.47)
\end{aligned}$$

The objective function $f_0(\tau_{ego}^{opt})$ in the proposed methodology is solely responsible for maximizing the comfort by minimizing the jerk of the trajectory. For the calculation of the jerk, a forward numerical differentiation method as outlined in Section 4.3.2 is utilized and, thus, requires at least three past states of the ego vehicle for computing

the jerk for each point of the optimized trajectory. The objective function $f_0(\tau_{ego}^{opt})$ is defined as the sum of the squared jerk terms (j_k^x, j_k^y) and can be denoted as

$$f_0(\tau_{ego}^{opt}) = \sum_{k=0}^{N} (j_k^x)^2 + (j_k^y)^2. \tag{4.48}$$

The velocity and other guiding values are not included in the objective function as these are handled by constraints. The proximity of the optimized trajectory to the learning-based trajectory is enforced using constraints. By bounding the last state of the optimized trajectory tight to the initial, learning-based solution, the goal is guaranteed to be reached by the optimized trajectory.

Equations 4.42-4.45 make sure that the optimized solution is within the extracted free-configuration space as outlined in Section 4.4.1. These constraints constitute the maximum deviation of the initial trajectory τ_{ego}^{init} to the optimized trajectory τ_{ego}^{opt}. If the optimized solution deviates too much from the initial, learning-based trajectory τ_{ego}^{init}, the interactions with other vehicles cannot be assumed to remain valid. Therefore, trust-region constraints are introduced in Equation 4.46 to ensure that the optimized trajectory is close to the initial, learning-based solution. Additionally, Equation 4.47 ensures that the control commands are bounded and are executable by the ego vehicle.

In summary, the post-optimization problem formulation provided in Equations 4.41-4.47 offers smooth, interactive, and comfortable trajectories for autonomous vehicles. Due to the safety being handled using constraints, the solution is guaranteed to be safe at convergence — if the other vehicles behave as expected. Additionally, by enforcing the trajectories to be close to each other, the game-theoric learning-based policies' interactiveness is maintained. Further, the optimized trajectories' jerk is at most as high as the learning-based initial trajectory as the sole objective is to minimize the jerk. The proposed post-optimization is evaluated in terms of its performance in detail in Section 5.4.

4.4.3 Nonlinear Trajectory Optimization Solution Methods

The post-optimization trajectory optmization problem formulation stated in Equations 4.41-4.47 constitutes a finite-dimensional, nonlinear optimization problem that can be solved using optimization solution methods, such as SQP and IP. The problem is an inequality constrained one having a nonlinear objective function $f_0(\tau_{ego}^{opt})$. As the other traffic participants in the scenario are included using constraints and the objective function is merely dependent on the ego vehicle's trajectory τ_{ego}, the optimization problem can be solved straightforwardly without the need of constantly re-computing the distance to other objects during the optimization process. The constraints for each time-step k can be seen as bounding boxes $(h_k^f, h_k^r, h_k^a, h_k^b)$ defined relative to an initial, learning-based state $\underline{x}_k^{init}$ of the ego vehicle. Further, as the same dynamic model is used in the simulation as in the time-marching integration of the optimized trajectory, the initial optimization vector $\underline{u}_{ego}^{init}$ produces a trajectory

that lies within the defined bounding boxes, dynamic limits, and that adheres to the vehicle model and, therefore, is a feasible solution for the current scenario. Feasible solutions eliminate the need for optimization solution classes to handle infeasible starting points, such as solving a boundary problem before the actual optimization process [119].

The above-stated post-optimization problem formulation poses a standard nonlinear program that can be solved using various approaches capable of handling inequality constraints. The most popular ones in this category are SQP and IP methods due to their simplicity and robustness [131].

SQP solves the nonlinear optimization problem iteratively by modeling nonlinear optimization subproblems using quadratic approximation and solving these for a time-step k. This process is then iteratively repeated until convergence or an abortion criterion has been reached [7].

The solution process of IP methods can be thought of as a central path approach where the solution starts in the center of the constraints and then moves towards a direction that minimizes the objective function. IP methods have been discussed in detail in Section 4.2.

For both of the solution classes, powerful solvers and libraries are available, such as for SQP (OSQP [108], SNOPT [38]) and IP (LOQO [116], IPOPT [119]). Both methods work well in solving the post-optimization problem formulation. This thesis utilizes IP methods to solve the constraint optimization problem. Results of applying the post-optimization to learned behavior policies are shown in Section 5.4.

5 Experiments and Results

This chapter provides a systematic evaluation of the proposed and discussed methodologies ranging from learning behavior policies, evaluating these at runtime, and post-optimizing these to obtain smooth driving behaviors.

First, the simulation frameworks Behavior benchmARK (BARK) and its machine learning extension Behavior benchmARK - Machine Learning (BARK-ML) are introduced, in which all of the results presented in this chapter have been obtained. The need and basic concepts of these frameworks are briefly elaborated, and the scenario used for training and evaluation is introduced — a dense merging scenario that requires interactive behavior policies to merge onto the other lane successfully.

Results and evaluations for learning behavior policies using actor-critic (AC) reinforcement learning (RL) are shown and discussed. The learned behavior policies' evaluation starts with an architecture and hyperparameter search for conventional deep neural network (DNN) and graph neural network (GNN) architectures. These two architectures are initially compared during training and then benchmarked. The potential-based reward shaping is evaluated and benchmarked against using a sparse reward signal. The edge features of the GNN architecture are visualized, and quantitative results for the magnitudes of these are obtained. As GNNs are at the intersection of structured and DNN approaches, the information flow in these can be visualized and is shown in the evaluation. An essential characteristic of learned behavior policies is how well these generalize. Therefore, variational studies are performed in which the behaviors of others is modified.

The next section shows results of applying the counterfactual behavior policy evaluation (CBPE) to learned behavior policies and provides additional insights. A set of independent and dependent behavior policies is used to replace the other vehicles' behaviors to ask and answer counterfactuals, such as "Would the behavior policy be collision-free if the other vehicle had, e.g., decelerated?". The CBPE allows for evaluating the learned behavior policies' generalization capabilities and how well it copes with distributional shifts at runtime.

Finally, the proposed behavior policy post-optimization is evaluated that utilizes the learned behavior policies. The boundary extraction process in the dense merging scenario and qualitative results for the constraint extraction are presented. Qualitative, as well as quantitative results, are shown and discussed.

5.1 Simulation and Benchmarking

Many behavior generation approaches have been tailored towards specific use-cases and are often fine-tuned for specific scenarios — making them non-comparable to

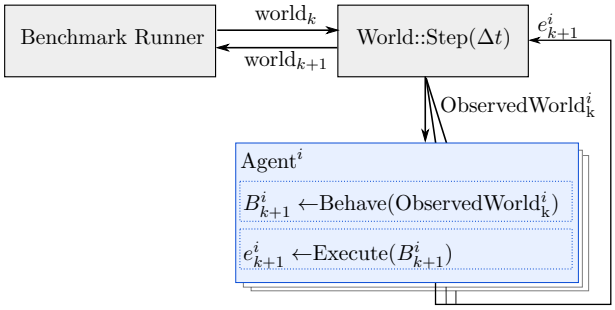


Figure 5.1: *ObservedWorld* concept of BARK. Each agent receives an *ObservedWorld* in which it executes its behavior and execution model (modified graphic from [13], ©2020 IEEE).

other state-of-the-art approaches. A behavior benchmarking framework BARK[1] has been developed to mitigate this issue and make behavior generation approaches comparable [13]. BARK has been developed together with Julian Bernhard, Klemens Esterle, and Tobias Kessler throughout this work's scope. All of the results presented in the evaluation section have been obtained using BARK and its machine learning extension BARK-ML[2]. Further, the used scenarios for training and evaluating the behavior policies and approaches presented in this work are described in this section.

5.1.1 BARK: A Semantic Simulation Framework

BARK is a semantic simulation framework for developing and benchmarking behavior generation algorithms. Its deterministic nature and semantic representation offer a computationally lightweight simulation that allows for benchmarking behavior generation algorithms over many scenarios.

BARK offers a variety of functionalities that makes developing and integrating novel behavior models easy. It offers extensive geometry functionalities that implement collision checks, distance functions, and other features. BARK utilizes the OpenDrive map format in which a wide variety of scenarios are represented in research and industry. By converting these maps into a graph that contains road- and lane-corridors, routing and free-space queries are handled with ease. The functionalities of BARK enable the fast development of elaborate behavior generation approaches.

BARK is a multi-agent simulation in which each agent is controlled by its behavior, dynamic, and execution model. In each simulation time-step t, an agent (vehicle) receives an *ObservedWorld* derived from the actual world as shown in Figure 5.1. An *ObservedWorld* clones the current world state and, additionally, can alter the world state to, e.g., model perturbations, such as occlusions. The behavior model plans and returns a trajectory based on the received *ObservedWorld*. This trajectory is then executed by the execution model that, e.g., can be a controller.

[1] `https://github.com/bark-simulator/bark`
[2] `https://github.com/bark-simulator/bark-ml`

A wide variety of behavior models are available in BARK ranging from rule-based models, such as the intelligent driver model (IDM) [113] and minimizing overall braking induced by lane changes (MOBIL) [57], to search-based models, such as an Monte-Carlo tree search (MCTS). In general, there are two simulation modii available in BARK: open- and closed-loop simulation. In the closed-loop simulation, the other vehicles are controlled by behavior models, such as the IDM and MOBIL. Using more sophisticated or learned behaviors is also possible, but one needs to consider the trade-off between elaborate and computationally inexpensive behaviors. In the other modus, the open-loop simulation, the other agents can be controlled, e.g., by using a dataset of recorded trajectories. One dataset that is natively supported in BARK is the INTERACTION dataset [129]. In the latter operational modus, the open-loop case, the other agents do not react to the ego vehicle's actions. Therefore, BARK provides a third modus that mixes the open- and closed-loop simulation in which some of the agents can be controlled by using a dataset and others by using behavior models.

With BARK offering a fast runtime, reproducible scenarios, and integrating various state-of-the-art behavior generation algorithms, it offers an ideal platform for developing and benchmarking behavior generation algorithms. More details about BARK can be found in [13].

5.1.2 BARK-ML: Machine Learning Framework for BARK

BARK-ML[3] is framework based on BARK that implements various machine learning capabilities. Its primary focus is on learning (deep) behavior policies by, e.g., utilizing RL. BARK-ML was developed as part of this work and implements a runtime that extends the one of BARK with an *observer* and *evaluator*.

The observer transforms the semantic world state into a suitable, machine learning usable representation for deep neural networks as discussed in Section 2.2. The evaluator evaluates the world's state, returns a reward signal, and determines whether an episode is terminal as described in Section 2.3.1.

BARK-ML provides an OpenAI-Gym[4] interface that allows it to be used by most open-source machine learning frameworks. Several standard scenarios and their configuration (blueprints) are provided – intersections, merging, and highways – that make training and evaluating novel scenarios easy. All BARK-ML behaviors can be plugged directly into BARK and, e.g., be used to control other traffic participants.

BARK-ML offers state-of-the-art RL algorithms to train behavior policies for BARK, such as the proximal policy optimization (PPO) and soft actor critic (SAC) that are based on the TF-Agents libaray[5]. All of the work presented in the learning section is available in BARK-ML.

[3] `https://github.com/bark-simulator/bark-ml`
[4] `https://gym.openai.com/`
[5] `https://github.com/tensorflow/agents`

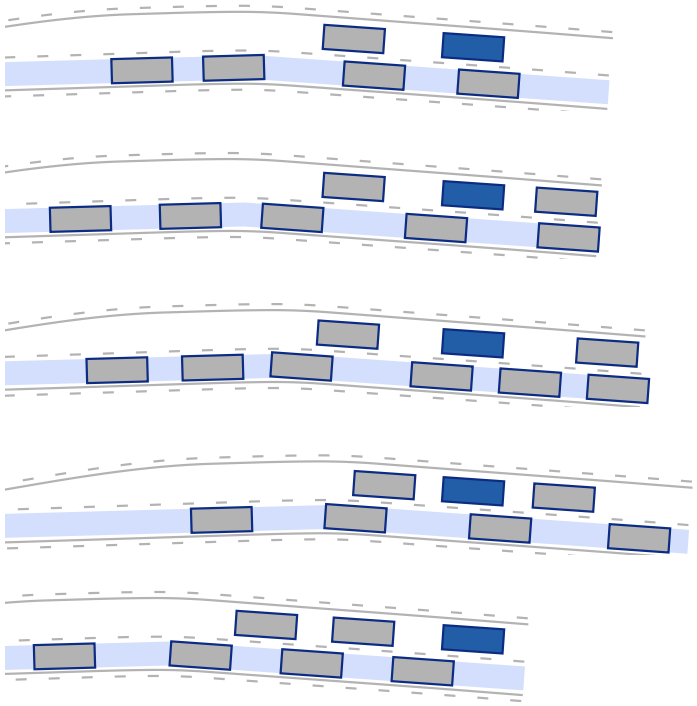


Figure 5.2: The merging scenario used for the training and evaluation of behavior policies. The ego vehicle is depicted as the blue vehicle and its goal is shown as the light blue polygonal area. The figure shows multiple sampled initial scenario states.

5.1.3 Training and Evaluation Scenarios

A merging scenario having a high traffic density is chosen to evaluate the proposed methodologies in challenging and interactive scenarios. Dense traffic scenarios require the ego vehicle's behavior policy to be proactive and interact with other traffic participants to merge onto the other lane. The merging scenario is modeled after the 'DR_DEU_Merging_MT' from the INTERACTION dataset [129]. The right lane is ending, and vehicles are required to merge to the left. The ego vehicle is placed on the right lane as depicted in blue in Figure 5.2. The ego vehicle's goal is placed on the left lane and is depicted by the light blue polygonal area. Thus, requiring the ego vehicle to change to the other lane. Additionally to the spatial goal definition, the ego vehicle's state needs to be within a speed of $[5\text{m/s}, 15\text{m/s}]$ and have a vehicle angle deviation smaller than $0.15rad$ to the goal's centerline. The vehicles drive initially with a speed of $9 - 11\text{m/s}$ which resembles higher traffic density speeds – which eventually could build up to a traffic jam. All vehicles – including the ego vehicle – are sampled uniformly along the lane's centerlines with an initial minimum distance of $7m$ and a maximum distance of 12m. The other vehicles are being controlled by the MOBIL model with the desired velocity set to 10m/s. The simulation-time of

an episode is 12s using a time-delta to step the simulation of 0.2s. The merging scenario shown in Figure 5.2 has a high traffic density. Not all starting positions pose scenarios where collisions can be avoided – even if emergency patterns are used, such as a stated-braking pattern [98]. Close distances make collisions merely by braking inevitable for the ego vehicle if the preceding vehicle decelerates with maximum deceleration. The other traffic participants' full behavioral parameters are stated in Appendix A.1.

5.2 Learning Behavior Policies for Autonomous Vehicles

This section provides results and insights on learning behavior policies for autonomous vehicles. First, several hyperparameters and architectures are evaluated to find well-performing architectures. Next, the proposed potential-based reward signal shaping functions are evaluated and compared against a sparse reward signal. Conventional DNN and GNN architectures are compared in terms of their performance during learning and are then benchmarked. Further, the edge values of the final GNN layer are visualized to provide additional insights into the learned behavior policies' information flow. Finally, variational studies are performed to evaluate the learned behavior policies' generalization capabilities and proposed architectures.

5.2.1 Hyperparameter and Architecture Search

Both the network architecture and the hyperparameters impact the resulting performance of learned behavior policies significantly. Therefore, this section compares and evaluates various architectures to obtain well-performing behavior policies. Two types of neural network architectures are evaluated: conventional DNN and GNN architectures.

For the evaluations performed in this section, the reward shaping function $\Phi(d, v)$ as introduced in Section 2.3.1 is used as sparse reward signals require an infeasible amount of training having limited computational resources. Other reward signals and shaping functions are evaluated in detail in the next section.

For learning behavior policies, the SAC algorithm is used that allows for off-policy updates, is capable of producing continuous actions, and that additionally maximizes the future cumulative expected entropy besides the reward that has been outlined in Section 2.1.4. The algorithm's hyperparameters are stated for the DNN and GNN architecture in Appendix A.1.

In the DNN architecture search, the number of layers is fixed, and the number of neurons is varied. A feature vector input representation is used as described in Section 2.2 which takes the four closest vehicles in the ego vehicle's surrounding into account. Additionally, the feature vector is sorted by the distance to the ego vehicle. The architectures for the DNNs are divided into *DNN small* and *DNN large*. The *DNN small* configuration has three layers with $[512, 256, 256]$ neurons and ReLU activation functions. The *DNN large* configuration has three layers with

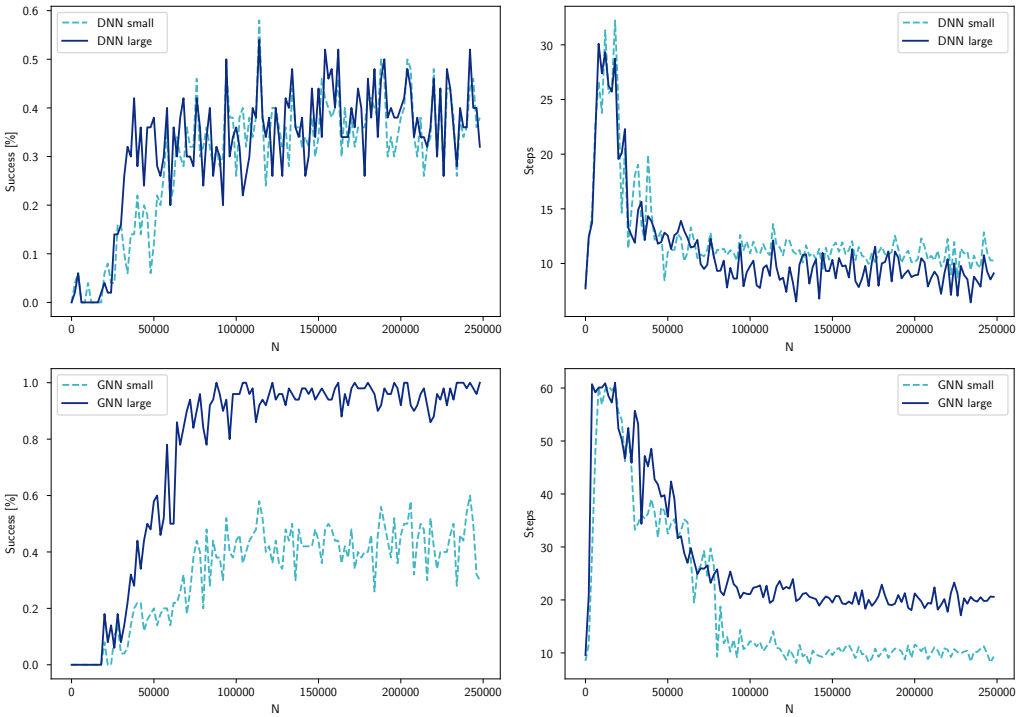


Figure 5.3: The first row shows the results for the DNN architectures and the second row shows the results for the GNN architectures during training evaluated every 2000 episodes using 25 evaluation scenarios. The GNN architectures outperform the conventional ones.

$[512, 512, 512]$ neurons and ReLU activation functions. The *DNN small* has 210.948 and the *DNN large* 540.164 trainable parameters.

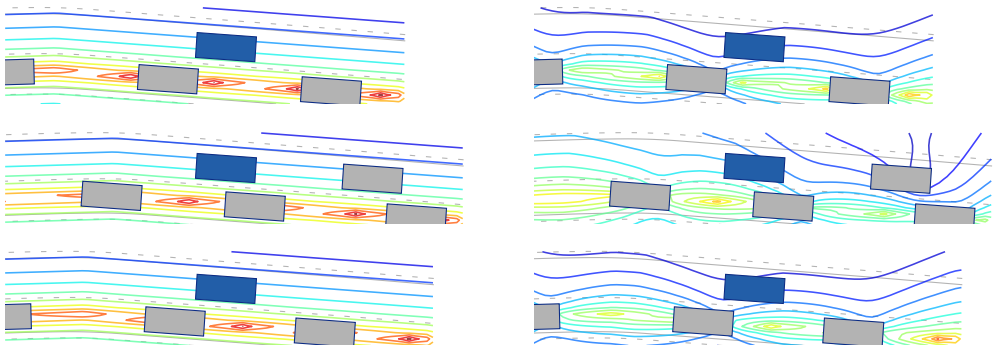
In the GNN architecture, the number of neurons and the embedding size of the GNN, is varied. A graph input representation is used as described in Section 2.2. Similar to the feature vector representation, the four closest vehicles are included in the graph. Each vehicle is connected with its three nearest vehicles. The two architectures for the GNNs are divided into *GNN small* and *GNN large*. The *GNN small* configuration has two GNN layers followed by a dense DNN having two layers which each layer having 256 neurons with ReLU activation functions. Each graph layer has two networks having 100 and 20 neurons to update the node and edge features, respectively. Therefore, The node values of the graph have a fixed size of 20. The *GNN large* configuration has two GNN layers followed by a dense DNN having two layers with 512 neurons each and ReLU activation functions. Additionally, there are two more dense networks per graph layer having 100 and 80 neurons to update the node and edge features. Therefore, The node values of the graph have a fixed size of 80. The *GNN small* has 104.032 and the *GNN large* 427.472 trainable parameters.

As the initial seed can significantly influence the learning performance, the training and simulation are started with different initial seeds (0, 1000) to gain more reliable performance estimates. In Figure 5.3, all curves represent the average value of the training runs. Figure 5.3 shows the success and average steps during training evaluated every 2000 episodes using 25 episodes. As can be seen, both the DNN and GNN architectures learn to reach the goal eventually. In terms of the average steps, they initially behave similarly, with the number of steps being large and then becoming smaller over time. The *GNN large* architecture outperforms all other architectures by far while the smaller networks are stuck in local optima. The training for the merging scenario is performed for $250,000$ episodes using $10,000$ sampled scenarios. Table 5.1 shows the architectures' resulting performance in terms of success, collisions, and the average steps using 2000 episodes. As can be seen in Table 5.1, even the *GNN small* architecture outperforms both of the DNN architectures. The *GNN large* architecture handles the dense merging scenario well with a success of 98.1%. Another important criterion is the number of parameters the networks have. The GNNs have fewer parameters due to updating the node and edge values element-wise as described in Section 2.4.2. As shown in the next section, the *GNN small* architecture can achieve similar results as the *GNN large* when using better-performing reward shaping functions. Compared with the *DNN large* architecture, the *GNN small* architecture has approximately five times less parameters.

	Success [%]	Collisions [%]	Steps [N]	Parameters [N]
DNN small	39.7%	60.3%	10.49	210.948
DNN large	39.9%	60.1%	9.79	540.164
GNN small	44.1%	55.9%	9.042	**104.032**
GNN large	**98.1 %**	**1.9 %**	19.65	427.472

Table 5.1: Different architectural configurations and their performance evaluated over 2000 dense merging scenarios. Values in bold represent the best value for a column.

With longer training times and in less dense scenarios, the conventional DNN architecture can eventually also achieve good performances as shown in [13]. There, the AC RL has been shown to outperform a state-of-the-art MCTS planner. However, as shown in the published work, the MCTS outperforms learned behavior policies if the other vehicle's behaviors have not been seen during training — motivating the need for approaches with better generalization capabilities. As shown in Section 5.2.4, the GNN actor-critic architecture can cope with deviating behavior of others and generalizes better. In the next section, the *GNN small* is used to conduct studies on the reward signal and shaping and whether the learning performance can be improved further when using a better reward shaping. Succesful and colliding scenarios are visualized in Appendix A.2.



(a) $\Phi(d)$ **(b)** $\Phi(d, d_i)$

Figure 5.4: (a) Potential-based reward shaping functions using the distance to the goal and (b) the distance to the goal and to others.

5.2.2 Reward Signal and Shaping

As discussed in Section 2.3.1, the reward signal impacts the performance during training and the resulting behavior policy significantly. For complex problems, sparse reward signals might render the problem infeasible to be learned in a finite time due to issues, such as the credit assignment and exploration problem as discussed in Section 2.3.1. This section compares sparse reward signals to potential-based reward signals as introduced in Section 2.3.1 in terms of their performance and efficiency. Figure 5.4 shows contour plots of reward potential-functions for the dense merging scenario — $\Phi(d)$ that takes the distance to the goal (the other lane's centerline) into account and $\Phi(d, d_i)$ that takes the distance to the goal and the distance to the other vehicles into account. The gradient from red to dark blue indicates a slope of the potential function. In Figure 5.4 (a), the lines are parallel to the centerline, with the potential function increasing linearly with a decreasing distance to the centerline. In Figure 5.4 (b), the same characteristics can be observed with the differences of sinks being present at the other vehicle's rear axle points.

In the sparse reward signal case, the agent is rewarded with a reward of $+1$ for reaching the goal, a reward of -1 for having a collision or leaving the drivable area, and a reward of 0 for having no collision and not leaving the drivable area. Figure 5.5 shows the performance during training for the sparse reward signal as well as for several reward shaping functions evaluated every 2000 episodes using 25 evaluation episodes applied to the dense merging scenario. As can be seen, when using a sparse reward signal (denoted as *None*), the ego vehicle does not reach the goal too often and reaches it for the first time at around $100,000$ episodes of training. It shows in the received reward that instead of learning to reach the goal, the agent learns to avoid collisions with other traffic participants.

The other curves shown in Figure 5.5 apply reward shaping to the sparse reward signal to obtain continuous reward signals. As in the previous section, all simulations

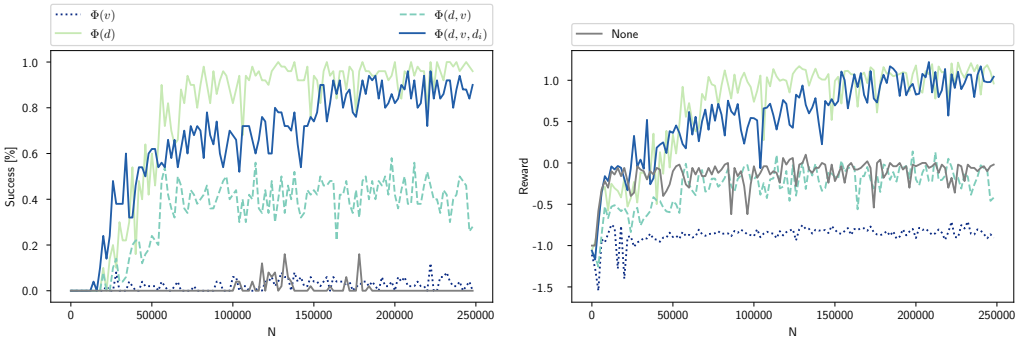


Figure 5.5: Success and the average reward achieved during training using a sparse reward signal and reward shaping functions evaluated every 2000 episodes using 25 evaluation episodes.

and training runs have been obtained using two random seeds, and the average values are being plotted. As can be seen, all reward shaping functions outperform the sparse reward signal concerning reaching the ego vehicle's goal. Even the worst-performing reward shaping function – only using the velocity potential $\Phi(v)$ – leads to higher success than using a sparse reward signal. The combined distance and velocity potential $\Phi(d, v)$ increases the behavior policies' performance in reaching the goal — to about 40%. The reward shaping functions increasing the success the most are the distance reward shaping $\Phi(d)$ and all potentials combined $\Phi(d, v, d_i)$ (distance, velocity, and the distance to other vehicles) as described in detail in Section 2.3.1. The *GNN small* architecture reaches the goal more than 90% of the time using the latter two reward shaping functions. The fact that the distance potential function performs better than all combined potentials $\Phi(d, v, d_i)$ might be due to using a point-wise distance between the rear-axles and not taking the actual vehicle shapes into account.

All results in this section have been obtained using the *GNN small* configuration that achieved a success of 44.1% in the previous section using the potential function $\Phi(d, v)$. By, e.g., using the distance reward shaping function $\Phi(d)$ the *GNN small* configuration can achieve similar success as the *GNN large* architecture. The increased performance shows the importance of how the Markov decision process (MDP) is set up and the impact reward signals can have on learning behavior policies and their resulting performance. Even the *GNN small* network can achieve success of over 90% in the dense merging scenario using the "right" reward shaping function.

5.2.3 Visualizing Information Propagation in Graph Neural Networks

As described in Section 2.4.1, GNNs are at the intersection of structured approaches and deep learning. The structured part of GNNs can be visualized to better understand the encoded driving behavior in the learned behavior policies by visualizing the flow of information in the network. In the GNN architecture, the information is

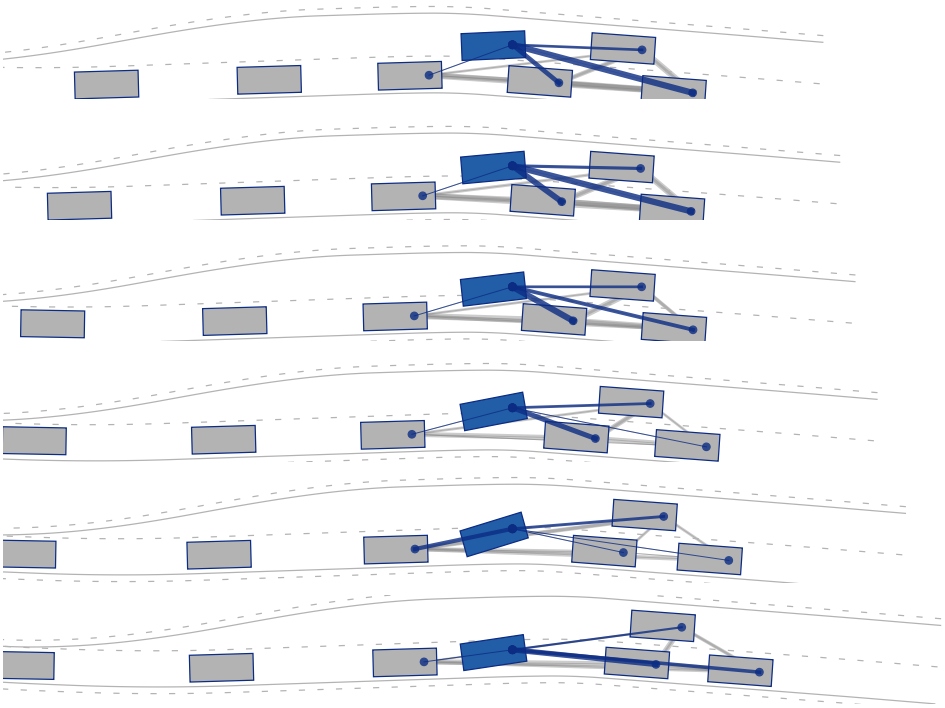


Figure 5.6: Graph edges visualized over the course of a scenario (each row is $\Delta t = 0.4$ seconds spaced).

first passed through the GNN layers forcing these layers to pass and extract relevant information from the input graph. As ReLU activation functions are used, all edge values are comprised of positive numbers. The normalized sum of the incoming edge values can be used to visualize the information propagation or flow — from here on referred to as the magnitude M. For example, the information flow through the edge e_{ij} from the i-th to the j-th vehicle can be measured by summing its values and by normalizing these with the sum of all incoming edges to vehicle v_j. In Figure 5.6 on the left, the input graph is displayed with the thickness of the lines corresponding to the magnitude M of the final layer of the GNN. It can be seen that besides information flowing to the ego vehicle (edges depicted in blue) from all surrounding vehicles, information is propagated between all vehicles throughout the graph. The strongest flow of information in Figure 5.6 is from the two vehicles that are behind the ego vehicle. Visualizing graph edges can provide insights into which vehicle the learned behavior policy takes into account and on which factors its decisions are based on.

In Figure 5.7, a histogram of the edge value magnitudes over the relative values is shown that has been generated using 1000 scenarios. The magnitude M of the edge values over the relative Cartesian x-distance Δx is plotted. It shows that information mainly flows through the GNN if the other vehicle is behind the ego vehicle or far in

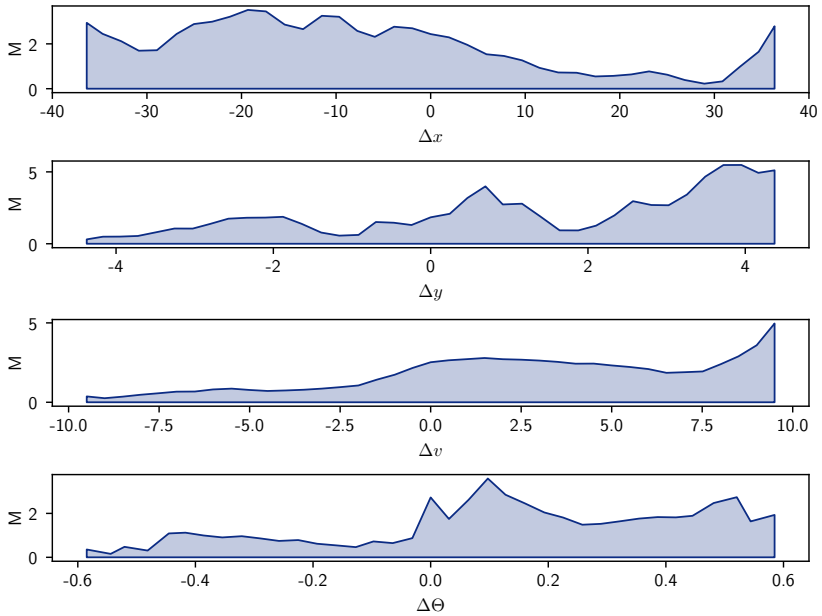


Figure 5.7: Histograms for the edge value magnitudes M plotted over the relative edge values. The histogram has been generated using 1000 episodes.

front. Paying attention to vehicles behind certainly makes sense in case of merging onto the other lane. Looking far ahead can be used to decide whether to merge in front or behind the other vehicle. In the second row, the magnitude M is plotted over the relative Cartesian y-distance Δy. The information flow is the strongest when the ego vehicle is still on the right lane and vehicles are to its left. Taking vehicles to the left more into account is expected as this mainly should influence its decision to change lanes. In the third row, the magnitude M is plotted over the relative velocity Δv. The edge values' largest magnitudes can be seen when the ego vehicle is slow and the other vehicles have higher velocities. Putting emphasis on vehicles having higher velocities is expected, considering that the ego vehicle tries to merge onto a faster lane and needs to keep track of faster moving vehicles to avoid collisions. For the relative vehicle angle $\Delta\theta$, it can be observed that the information flow is the largest around zero or with small positive angles indicating the process of changing lanes to the left.

This section showed that GNN architectures do not only outperform conventional DNN architectures but also allow for additional insights into learned behavior policies and how the information flows through these. More scenarios visualizing graphs and the information flow can be seen in Appendix A.3.

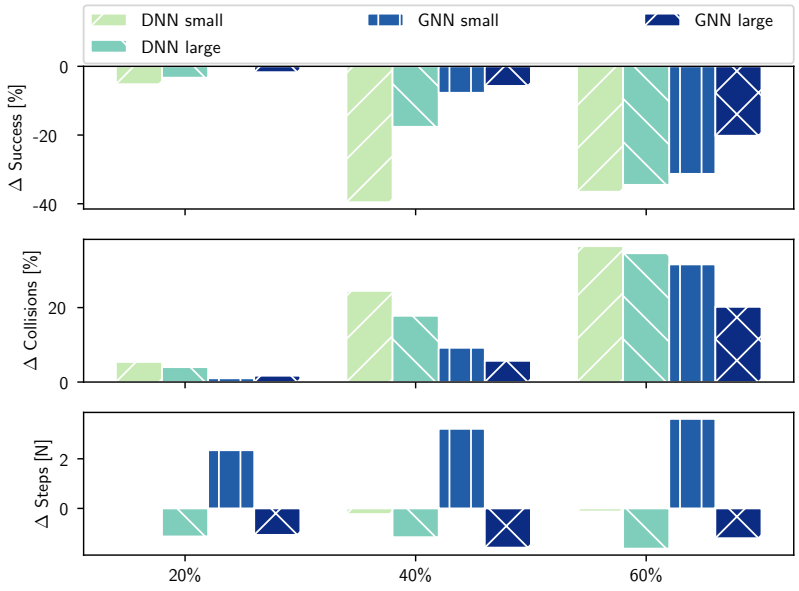
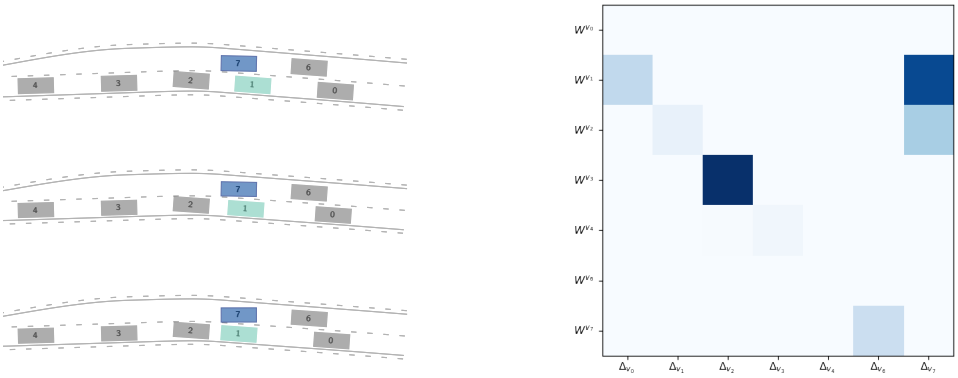


Figure 5.8: Variational studies for the conventional DNN and GNN architectures plotted over the percentage of variation in the desired time-headway parameter of the MOBIL model.

5.2.4 Variational Studies and Generalization of Learned Behavior Policies

This section conducts variational studies in which the behavior of other traffic participants differs from the ones seen during training to conduct studies on the generalization and transferability of learned behavior policies. Generalization in the context of behavior generation is defined as the ability of a (learned) behavior policy to perform well in scenarios with behavioral parameters that have not been seen during training. Specifically, the behavior policies of other vehicles are modified from the training to the validation phase. Similar to [13], the time-headway parameter T_{head} of the MOBIL model is varied to generate more passive and aggressive driving styles of the other vehicles. Initially, during training, the time-headway T_{head} is set to 1.5s. For the evaluation, the parameter is reduced by $20\%, 40\%$ and 60%, resulting in time-headways of $[1.2\text{s}, 0.9\text{s}, 0.6\text{s}]$, respectively. All networks presented in Section 5.2.1 are evaluated in the variational studies to see whether DNN or GNN architectures generalize better in the used merging scenario. Figure 5.8 shows the results of the variational study and how much the performance of each policy changes with a decreasing time-headway parameter T_{head}. It can be seen that minor deviations of around 20% in the other vehicles' time-headway parameter do not have a vast impact on any network's success and collisions. However, by changing the time-headway parameter by 40%, especially the conventional DNN architectures perform worse, indicating that these do not generalize well to the changed behavioral



(a) $W^{v_1}_{t=1.2\text{s}}$ **(b)** $H_{t=1.2\text{s}}$

Figure 5.9: (a) Counterfactual worlds W^{v_1} in which the behavior policy of vehicle v_1 is exchanged multiple times with independent behavior policies at time-step $t = 1.2$s. (b) Corresponding influence heatmaps $H_{t=1.2\text{s}}$ showing the influence of the counterfactual worlds on the behavior policies.

parameters. The *DNN small* configuration performs much worse with a decrease of the success of almost 40%. The GNN architectures perform only slightly worse in the 40% variational case than in the nominal scenario with a decrease in reaching the goal of a few percent. When the time-headway is changed by 60%, the GNN architectures still outperform any of the conventional DNN architectures, although the performance also deteriorates significantly. However, a time-headway T_{head} of $0.6s$ models very aggressive driving styles of the other vehicles in which the collisions in the real world with human drivers most probably also would increase.

5.3 Counterfactual Behavior Policy Evaluation

This section applies the proposed CBPE presented in Section 3.3 to learned behavior policies at runtime. The *GNN large* configuration from Section 5.2.1 is used for the evaluation of the CBPE. The previous section's variational studies show that learned behavior policies can only cope to a certain extent with deviating behavior policies from training. By applying the CBPE at runtime, estimates can be obtained on whether the learned behavior policy can cope with non-factual changes in others' behavior policies in the current scenario by asking and answering counterfactuals. Two different studies are conducted: replacing the others' behavior policies with independent and dependent ones. Behavior policies independent of their surroundings allow for precise extraction of the influence but might cause collisions as discussed in Section 3.3. Dependent behavior policies might be influenced more by their surroundings than the actual parameter change. Both of these approaches have advantages that are evaluated and elaborated on in this section.

5.3.1 Independent Behavior Policies

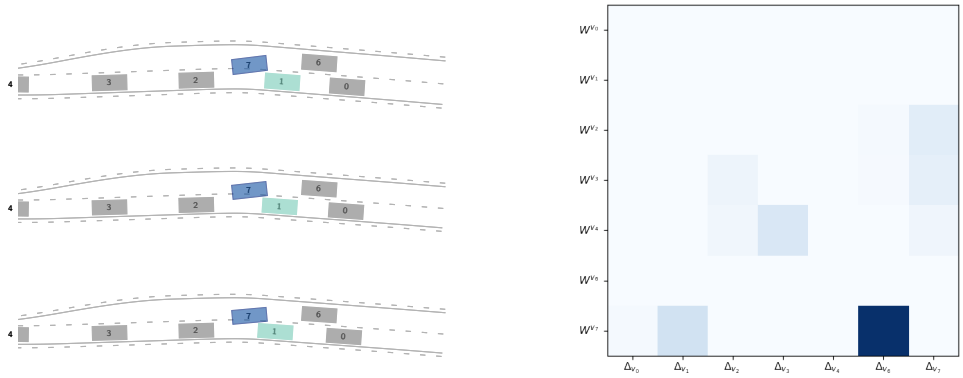
Figure 5.9 shows results of replacing the other vehicles' behavior policies with independent ones at time $t = 1.2\text{s}$. A constant acceleration behavior model B^{const} is used. Sampling many accelerations to achieve statistically significant results at runtime is not feasible due to computational limitations and bounded runtime. Therefore, a discrete behavior pool $\mathcal{B} = [\pi_0, \pi_1, \pi_2]$ is used with the respective accelerations $[-2\text{m/s}^2, 0\text{m/s}^2, 2\text{m/s}^2]$. Each row in Figure 5.9 (a) corresponds to a constant acceleration. The counterfactual worlds are shown at the forward simulation time $t = 1.2\text{s}$. The top row shows the counterfactual world $W^{v_1 \sim \pi_0}_{t=1.2\text{s}}$, the middle row $W^{v_1 \sim \pi_1}_{t=1.2\text{s}}$, and the bottom row $W^{v_1 \sim \pi_2}_{t=1.2\text{s}}$. The ego vehicle v_7 is depicted in blue and controlled by the *GNN large* architecture from Section 5.2.1. Using such non-factual behaviors allows evaluating the learned behavior policy using counterfactuals in the form of "Would the behavior policy be collision-free if the other vehicle had decelerated/driven with constant acceleration/or accelerated?". The behavior policy π_{ego} can handle all counterfactual worlds without in the shown case.

In Figure 5.9 (b), the influence heatmap $H_{t=1.2\text{s}}$ for all counterfactual worlds $W^{\square}_{t=1.2\text{s}}$ is shown. The y-axis represents counterfactual worlds, with each row corresponding to a vehicle v_i. The x-axis shows the influence of the counterfactual worlds onto a single vehicles v_i using Equation 3.4. The darker the colors are in the heatmap, the stronger the influence from the counterfactual worlds onto a vehicle. It can be seen that the preceding vehicles have a considerable influence on their tailing vehicles. For example, vehicle v_3 has a large influence onto vehicle v_2. The ego vehicle v_7 is influenced by counterfactual worlds W^{v_1} and W^{v_2} where the behaviors of vehicle v_1 and v_2 have been exchanged, respectively.

5.3.2 Dependent Behavior Policies

Figure 5.10 shows results of replacing the other vehicles' behavior policies with dependent ones at time $t = 2.4\text{s}$. The IDM is used and its time-headway parameter T_{head} is varied as in the variational studies in Section 5.2.4. A discrete behavior policy pool $\mathcal{B} = [\pi_0, \pi_1, \pi_2]$ is used with the respective time-headway parameters $[1.2\text{s}, 0.9\text{s}, 0.6\text{s}]$. The smaller the time-headway parameter is, the more aggressive the behavior of the vehicle is. The counterfactual worlds are shown at the forward simulation time $t = 2.4\text{s}$. The top row shows the counterfactual world $W^{v_1 \sim \pi_0}_{t=2.4\text{s}}$, the middle row $W^{v_1 \sim \pi_1}_{t=2.4\text{s}}$, and the bottom row $W^{v_1 \sim \pi_2}_{t=2.4\text{s}}$. Each row in Figure 5.10 (a) corresponds to a varied time-headway parameter. The same scenario is evaluated as with the independent behavior policies. The behavior policy π_{ego} can handle all counterfactual worlds without causing dangerous situations.

In Figure 5.10 (b), the influence heatmap $H_{t=2.4\text{s}}$ for all counterfactual worlds $W^{\square}_{t=2.4\text{s}}$ is shown. For example, vehicle v_3 has a large influence onto vehicle v_2. The ego vehicle v_7 is influenced mainly in the counterfactual worlds W^{v_1}, W^{v_2}, and W^{v_3} where the behaviors of vehicle v_1, v_2, v_3 have been exchanged, respectively.



(a) $W^{v_1}_{t=2.4\text{s}}$ **(b)** $H_{t=2.4\text{s}}$

Figure 5.10: (a) Counterfactual worlds W^{v_1} in which the behavior policy of vehicle v_1 is exchanged multiple times with dependent behavior policies at time-step $t = 2.4$s. (b) Corresponding influence heatmaps $H_{t=2.4\text{s}}$ showing the influence of the counterfactual worlds on the behavior policies.

5.3.3 Summary and Remarks

This evaluation shows that the CBPE provides a framework for analyzing the performance, generalization capabilities, and how vehicles influence each other at runtime. Further, the CBPE brings insights into the encoded driving behavior of learned behavior policies, how they react to changes in the environment, and how they would behave in hypothetical, counterfactual, possibly edge-case worlds. The variational studies presented in Section 5.2.4 are performed at runtime to evaluate the policies' generalization capabilities and whether they can handle distributional shifts. The independent behavior policies are not influenced by others and correlations between the behavior policies can precisely be extracted. However, these tend to cause collisions when larger time horizons are used. Collisions can be avoided using dependent behavior policies at the cost of not precisely extracting the relations between the vehicles' behavior policies. The proposed CBPE can be used in the decision logic of, e.g., Simplex architectures for switching between the high-performance and high-assurance controller. Restricting the usage of learned behavior policies on highway scenarios has been shown to reduce the overall collisions significantly in [44]. The CBPE merely restricts the usage of the learned behavior policy based on its performance and potentially is less conservative than, e.g., reachability analysis.

5.4 Post-Optimization of Learned Behavior Policies

This section applies and evaluates the post-optimization that has been introduced in Section 4.4. Extracted constraints using the line-search on the x- and y-direction as described in Section 4.4.1 are shown. The results of applying the post-optimization

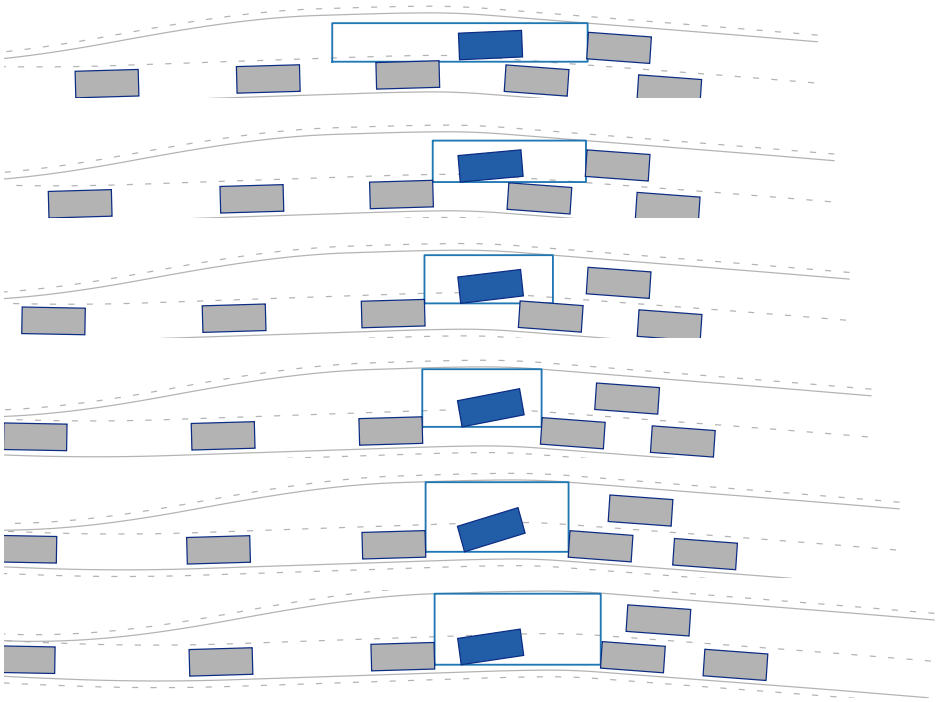


Figure 5.11: Constraint extraction for the world W_t during an episode at times $t = 0.4\text{s} * i$ with i being the respective row. The ego vehicle is depicted in blue and the extracted polygonal areas are obtained using a line-search in the x- and y-directions.

in an exemplary scenario are shown and compared and quantitative results for the post-optimization are provided.

Figure 5.11 shows the extracted free configuration space using line-search over the course of an episode. The ego vehicle is depicted in blue and is controlled by a learned behavior policy π_{ego} and the others using the MOBIL model. The maximum free-space is extracted using a line-search along the x- and y-axis starting from the rear-axle of the ego vehicle as described in Section 4.4.1. The line-search terminates either when intersecting the road boundaries (drivable area) or other objects in the environment — the other vehicles are depicted in gray. Further extracted constraints can be seen in Appendix A.4.

Additionally to the visualized constraints in Figure 5.11, the input commands are limited for the resulting trajectory to be executable by the ego vehicle — the steering rate δ is limited to a range of $[-0.2\text{rad/s}, 0.2\text{rad/s}]$ and the acceleration is limited in a range of $[-5\text{m/s}^2, 4\text{m/s}^2]$. To enforce closeness to the initial, learning-based trajectory and for the interactions of the learned behavior policy to remain valid, the maximum threshold radius is limited as stated in Equation 4.46 by the radius r_k. In this work, the trust-region radius is constant with $r_k = 2\text{m}$ besides the

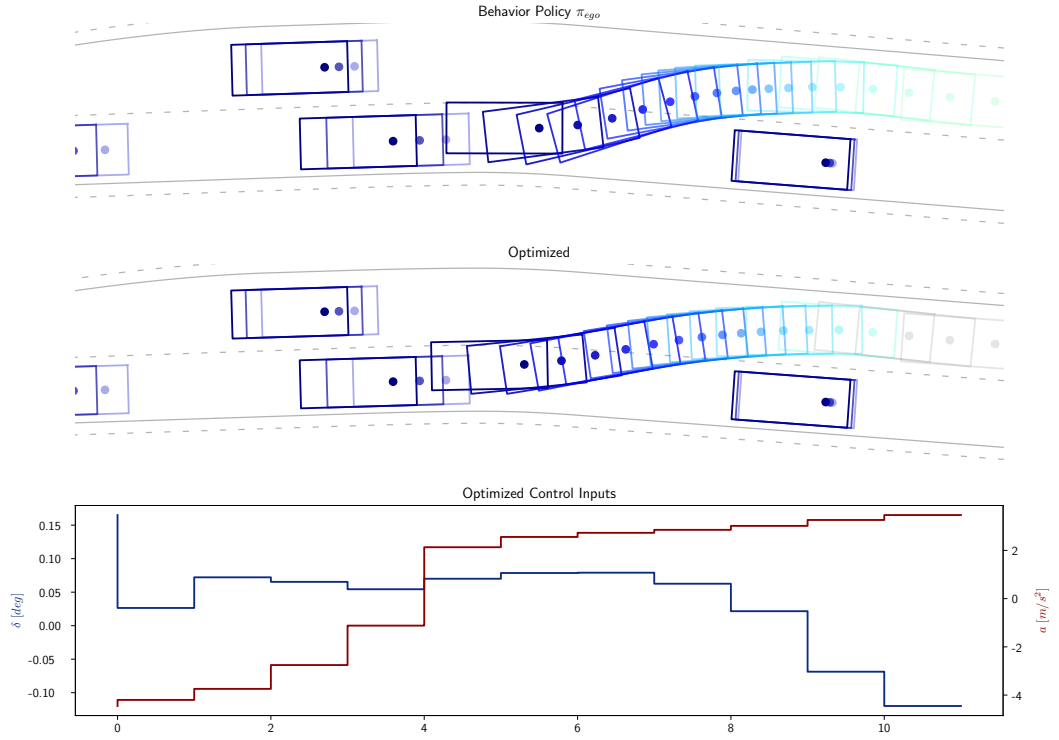


Figure 5.12: At the top, the trajectory generated by the learned behavior policy is shown. In the middle row, the optimized trajectory that is smoother and that adheres guaranteed to the defined constraints. At the bottom, the input signals of the optimization are shown.

last state, which is restricted tightly to ensure that the goal-conditions are met in the optimized trajectory. In the scenario shown in Figure 5.11, the final sate is constrained to be within 0.1m in the Cartesian coordinates, to have a velocity deviation in a range of $[-0.1\text{m/s}, 0.1\text{m/s}]$, and for the vehicle angle θ to be within a range of $[-0.05\text{rad}, 0.05\text{rad}]$.

Figure 5.12 shows the trajectory τ_{ego}^{init} of the learned behavior policy π_{ego} in the top row. The colors indicate the time dependency of the trajectory, with darker blue colors indicating states that are further ahead in time. The learned behavior policy can reach the defined goal without causing any collisions.

In the middle row in Figure 5.12, the optimized trajectory τ_{ego}^{opt} in which the jerk of the initial trajectory τ_{ego}^{init} is reduced is shown. As can be seen, the optimized trajectory τ_{ego}^{opt} is smoother overall — leading to more comfortable driving behavior. The first three states (depicted in gray) are not optimized as the numerical forward differentiation requires three past states to calculate the jerk. The same final state is reached and the goal condition is met.

The bottom row of Figure 5.12 shows the corresponding control commands produced by the post-optimization. As the objective is to decrease the squared jerk

term, the resulting control commands of the post-optimization are smoother than those produced by the learned behavior policy π_{ego}. Further, as the optimized trajectory has been generated using direct shooting, a dynamic vehicle model, and time-marching integration, the resulting trajectory can directly be executed by the ego vehicle.

The post-optimization reduces the squared jerk as states in Equation 4.48 by 85.78% for the dense merging scenario evaluated over 100 episodes. Having well-performing initial estimates allows for local optimization in nonconvex environments, such as traffic, without optimizing multiple local optima. More optimized scenarios are shown in Appendix A.5.

6 Conclusion

This section summarizes the thesis' work, discusses its specific contributions, and provides an outlook on future work.

6.1 Summary

The thesis motivated and discussed behavior generation methods – their advantages and shortcomings and motivated the need for learning-based methods. Learning-based methods are data-driven methods that can adapt and learn how to behave in traffic. Markov decision processes (MDPs) and its solution methods have been discussed in detail and a MDP formulation for learning behavior policies for autonomous driving has been provided. Several input representations have been discussed in terms of their characteristics. The invariance towards the number and order of vehicles as well as their generalization capabilities have been pointed out. Further, the reward signal for learning behavior policies in an MDP has been discussed and is benchmarked. It is shown that the reward signal and reward shaping have profound effects on the learning process and the resulting performance of the behavior policy. To avoid the *credit assignment problem* – where an agent only receives sparse rewards – several potential-based reward shaping functions have been proposed and benchmarked. A novel graph neural network (GNN) actor-critic architecture has been proposed that is invariant towards the number and order of vehicles. It also makes insights into the information flow between the vehicles (nodes) possible by evaluating the edge values. Further, an extensive benchmark and variational studies have been conducted that show that the novel architecture provides a higher performance and also generalizes better than conventional deep neural networks (DNNs).

Further, runtime safety assurance (RTSA) methods have been discussed as a method to apply behavior policies using DNNs as function approximator in safety-critical environments. As it is infeasible to evaluate all combinatorial options in simulation offline, an online evaluation of the learned behavior policies is required. A counterfactual behavior policy evaluation (CBPE) has been introduced that evaluates the learned behavior policies using counterfactual worlds in which other vehicles non-factual. Using these counterfactual worlds, questions, such as "Would the behavior policy have caused a collision if the other vehicle had accelerated?" can be answered at runtime. The performance of the learned behavior policy over all counterfactual worlds can then be evaluated to see how the other vehicles influence each other and if the learned behavior policy performs well in these using various metrics, such as, e.g., the collision rate.

Behavior policies that utilize machine learning and DNNs as approximation functions cannot guarantee smooth behavior trajectories. This is, e.g., due to the highly nonlinear nature of DNNs or can be due to not exploring the full configuration space during training. This thesis introduces a post-optimization that utilizes the learned behavior policy to generate initial estimates and to derive constraints for each time step. The initial estimate is generated by calling each vehicle's behavior policy iteratively and the constraints are extracted using all vehicle's states. A constrained optimization problem formulation is then introduced that uses the initial estimates and constraints to provide smooth trajectories whilst adhering to the constraints. For the interactions of the ego vehicle with its surrounding vehicles to remain valid, a threshold radius is introduced that defines the maximum deviation to the initial, learning-based states. As a dynamic model is used and the input control commands are constrained to a defined range, the resulting trajectory is executable by the ego vehicle. Thus, the post-optimization smoothens the initial estimate provided by the learning-based behavior policy whilst guaranteeing the same safety constraints.

6.2 Discussion

This thesis provided a thorough discussion of learning behavior policies within an MDPs setting, proposing and discussing input representations, reward signals and shaping functions, as well as proposing a novel GNN actor-critic architecture. The input representations are discussed in detail – their advantages and shortcomings – and are benchmarked in terms of their performance and generalization. Although various input representations have been discussed, a more thorough investigation, e.g., in the form of an architecture search could be performed to generate a holistic view. Furthermore, other input representations could be considered, such as transforming the vehicle's states into the Frenet coordinate system. Several reward signals are proposed and potential-based reward shaping functions for learning behavior policies for autonomous vehicles are proposed and benchmarked. It is shown that the reward signal has significant effects on the learning performance and the resulting behaviors. To further validate these results, various learning approaches should be benchmarked as some algorithms might be able to cope better with sparse reward signals than others. Finally, to overcome the invariance towards the number and order of vehicles as well as to obtain insights into the learned behavior policy, a GNN actor-critic architecture has been proposed. The novel architecture has been shown to outperform conventional DNNs in terms of their performance. Variational studies have been conducted where the behavior of others has been varied from the nominal training scenario. The novel GNN architecture generalizes better than conventional DNNs ones. Further investigations are required to find the cause of the novel architecture generalizing better – e.g., be it due to the relative values or the graph structure in itself.

A RTSA method, the CBPE has been proposed to evaluate learned behavior policies at runtime. It is able to provide insights on how the behavior policies

of vehicles influence each other and to ask and answer counterfactual questions. Counterfactual worlds are used in which the behavior of other vehicles has been altered. Using these insights, the behavior policies' usage can be restricted using the performance over all counterfactual worlds. However, even if the learned behavior policy is considered unsafe for a given scenario, a fallback behavior policy is required that transfers the system-state into a safe state – which is non-trivial in complex traffic scenarios.

A post-optimization has been introduced that smoothens the learned behavior policies' trajectory whilst adhering to extracted constraints. It is formulated as a constrained optimization problem and solved using interior-point solution methods. Other methods capable of solving constrained optimization problems, such as sequential quadratic programs (SQPs) could be evaluated to, e.g., speed up the time to convergence. The proximity constraint in the problem formulation might still cause non-smooth behaviors if chosen too small, as the jerk cannot be reduced significantly then. If the proximity constraint is chosen too large, interactions with other vehicles that the learning-based behavior policy had might not remain valid. Thus, the choice of the proximity constraint is crucial to the performance and comfort and should be evaluated further.

6.3 Future Work

This section discusses how the proposed methodologies for learning, evaluating, and optimizing behavior policies can be extended in future work — ranging from extending the GNN actor-critic architecture to a multi-agent formulation, over extending the graph structure with more verbose information, to using interactive prediction models in the post-optimization.

6.3.1 Multi-Agent Graph Neural Network Reinforcement Learning

This work proposes a GNN actor-critic architecture to learn behavior policies for the ego vehicle. However, at the last GNN layer, only the ego vehicle's node embedding is utilized and the other node embeddings are disregarded (not used). In future work, the single-agent learning problem could be extended to a multi-agent architecture by utilizing all node values in the output layer of the GNN. Actions could then be learned for all agents simultaneously using a single neural network. Also, the node embeddings of the other vehicles might already contain all the information to output meaningful actions for each vehicle due to the combinatorial generalization GNNs provide. These characteristics could make such an architecture ideal for learning cooperative driving maneuvers to optimize the traffic flow, such as in dense merging scenarios.

6.3.2 Extending the Graph Structure Using Environmental Information

The graphs used in this thesis only include information on the vehicles in the node and edge features. However, verbose information in the graph is required to generate a universal behavior generation approach, such as environmental information. The node values could then be transformed regarding this environmental information, such as waypoints, to break the dependency on global coordinates. This would enable to include information, such as waypoints and routing in general and also would enable to include additional traffic information, such as traffic signs and lights. An extension of the graph structure would be essential in transferring the proposed approach into the real world.

6.3.3 Interactive Post-Optimization of Learned Behavior Policies

The proposed post-optimization initially calls the behavior model of each vehicle in the scene to obtain trajectories. Using these trajectories, constraints for the ego vehicle at each time-step can be derived as well as the initial trajectory for the ego vehicle. The post-optimization itself is not interactive with other vehicles – meaning that the other vehicles do not react to changes the optimizer does. To mitigate this problem, a trust-region radius has been introduced in this work that enforces the optimized trajectory to be close to the initial, learning-based one. In future work, the post-optimization could be extended using interactive prediction models that react to changes the optimizer makes. This would yield more realistic trajectories as the other vehicles' interactions are also included in the optimization-based solution.

Bibliography

[1] Pieter Abbeel, Adam Coates, and Andrew Y. Ng. "Autonomous helicopter aerobatics through apprenticeship learning". In: *International Journal of Robotics Research* 29.13 (2010), pp. 1608–1639. ISSN: 02783649. DOI: 10.1177/0278364910371999.

[2] Abbas Abdolmaleki, Jost Tobias Springenberg, Yuval Tassa, Remi Munos, Nicolas Heess, and Martin Riedmiller. "Maximum a posteriori policy optimisation". In: *arXiv* (2018). ISSN: 23318422. arXiv: 1806.06920.

[3] Subutai Ahmad and Gerald Tesauro. "Scaling and Generalization in Neural Networks: A Case Study". In: *Advances in Neural Information Processing Systems 1* (1989), pp. 160–168.

[4] Smruti Amarjyoti. "Deep reinforcement learning for robotic manipulation-the state of the art". In: *arXiv* 10.2 (2017). ISSN: 23318422. arXiv: 1701.08878.

[5] James Atwood and Don Towsley. "Diffusion-convolutional neural networks". In: *Advances in Neural Information Processing Systems* Nips (2016), pp. 2001–2009. ISSN: 10495258. arXiv: 1511.02136.

[6] Somrita Banerjee, Thomas Lew, Riccardo Bonalli, Abdulaziz Alfaadhel, Ibrahim Abdulaziz Alomar, Hesham M. Shageer, and Marco Pavone. "Learning-based Warm-Starting for Fast Sequential Convex Programming and Trajectory Optimization". In: *IEEE Aerospace Conference Proceedings* (2020). ISSN: 1095323X. DOI: 10.1109/AERO47225.2020.9172293.

[7] The Basic, S Q P Method, Introductory Definitions, Assumptions Sequential, Quadratic Programming, The Nlp, and Quadratic Programming. "Sequential Quadratic Programming". In: *Optimization Theory and Methods* (2006), pp. 523–560. DOI: 10.1007/0-387-24976-1_12.

[8] Peter Battaglia, Razvan Pascanu, Matthew Lai, Danilo Rezende, and Koray Kavukcuoglu. "Interaction networks for learning about objects, relations and physics". In: *Advances in Neural Information Processing Systems* (2016), pp. 4509–4517. ISSN: 10495258. arXiv: 1612.00222.

[9] Peter W. Battaglia, Jessica B. Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, Caglar Gulcehre, Francis Song, Andrew Ballard, Justin Gilmer, George Dahl, Ashish Vaswani, Kelsey Allen, Charles Nash, Victoria Langston, Chris Dyer, Nicolas Heess, Daan Wierstra, Pushmeet Kohli, Matt Botvinick, Oriol Vinyals, Yujia Li, and Razvan Pascanu.

"Relational inductive biases, deep learning, and graph networks". In: *arXiv* (2018), pp. 1–40. arXiv: 1806.01261.

[10] Marc G. Bellemare, Will Dabney, and Rémi Munos. "A distributional perspective on reinforcement learning". In: *34th International Conference on Machine Learning, ICML 2017* 1 (2017), pp. 693–711. arXiv: 1707.06887.

[11] Philipp Bender, Omer Sahin Tas, Julius Ziegler, and Christoph Stiller. "The combinatorial aspect of motion planning: Maneuver variants in structured environments". In: *IEEE Intelligent Vehicles Symposium, Proceedings* 2015-Augus.Iv (2015), pp. 1386–1392. DOI: 10.1109/IVS.2015.7225909.

[12] Christopher Berner, Greg Brockman, Brooke Chan, Vicki Cheung, Przemysław Psyho Dębiak, Christy Dennison, David Farhi, Quirin Fischer, Shariq Hashme, Chris Hesse, Rafal Józefowicz, Scott Gray, Catherine Olsson, Jakub Pachocki, Michael Petrov, Henrique Pondé De Oliveira Pinto, Jonathan Raiman, Tim Salimans, Jeremy Schlatter, Jonas Schneider, Szymon Sidor, Ilya Sutskever, Jie Tang, Filip Wolski, and Susan Zhang. "Dota 2 with large scale deep reinforcement learning". In: *arXiv* (2019). ISSN: 23318422. arXiv: 1912.06680.

[13] Julian Bernhard, Klemens Esterle, Patrick Hart, and Tobias Kessler. "Bark: Open behavior benchmarking in multi-agent environments". In: *arXiv* (2020). ISSN: 23318422. DOI: 10.1109/IROS45743.2020.9341222. arXiv: 2003.02604.

[14] Julian Bernhard, Robert Gieselmann, Klemens Esterle, and Alois Knol. "Experience-Based Heuristic Search: Robust Motion Planning with Deep Q-Learning". In: *IEEE Conference on Intelligent Transportation Systems, Proceedings, ITSC*. Vol. 2018-Novem. 2018, pp. 3175–3182. ISBN: 9781728103235. DOI: 10.1109/ITSC.2018.8569436.

[15] John T. Betts. "A Survey of Numerical Methods for Trajectory Optimization". In: *Journal of guidance, control, and dynamics* (1998), pp. 193–207. ISSN: 00263788.

[16] Stephen Boyd and Lieven Vandenberghem. *Convex Optimization*. Cambridge university press, 20048. DOI: 10.1201/9781420049503-c34.

[17] Michael M. Bronstein, Joan Bruna, Yann Lecun, Arthur Szlam, and Pierre Vandergheynst. "Geometric Deep Learning: Going beyond Euclidean data". In: *IEEE Signal Processing Magazine* 34.4 (2017), pp. 18–42. ISSN: 10535888. DOI: 10.1109/MSP.2017.2693418. arXiv: 1611.08097.

[18] Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. "Spectral networks and deep locally connected networks on graphs". In: *2nd International Conference on Learning Representations, ICLR 2014 - Conference Track Proceedings* (2014), pp. 1–14. arXiv: 1312.6203.

[19] Simon Burton and Richard Hawkins. *Assuring the safety of highly automated driving : state-of-the-art and research perspectives*. 2020.

[20] Butcher. "A history of Runge-Kutta methods". In: *Applied numerical mathematics* 20.3 (1996), pp. 247–260.

[21] Shaosheng Cao, Wei Lu, and Qiongkai Xu. "Deep neural networks for learning graph representations". In: *30th AAAI Conference on Artificial Intelligence, AAAI 2016* (2016), pp. 1145–1152.

[22] Runqi Chai, Antonios Tsourdos, Al Savvaris, Senchun Chai, and Yuanqing Xia. "Two-Stage Trajectory Optimization for Autonomous Ground Vehicles Parking Maneuver". In: *IEEE Transactions on Industrial Informatics* 15.7 (2019), pp. 3899–3909. ISSN: 15513203. DOI: 10.1109/TII.2018.2883545.

[23] Richard Cheng, Gábor Orosz, Richard M. Murray, and Joel W. Burdick. "End-to-end safe reinforcement learning through barrier functions for safety-critical continuous control tasks". In: *33rd AAAI Conference on Artificial Intelligence, AAAI 2019, 31st Innovative Applications of Artificial Intelligence Conference, IAAI 2019 and the 9th AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019* (2019), pp. 3387–3395. ISSN: 2159-5399. DOI: 10.1609/aaai.v33i01.33013387. arXiv: 1903.08792.

[24] Tyler Derr, Yao Ma, and Jiliang Tang. "Signed Graph Convolutional Networks". In: *Proceedings - IEEE International Conference on Data Mining, ICDM* 2018-Novem (2018), pp. 929–934. ISSN: 15504786. DOI: 10.1109/ICDM.2018.00113. arXiv: 1808.06354.

[25] Ankush Desai, Shromona Ghosh, Sanjit A. Seshia, Natarajan Shankar, and Ashish Tiwari. "SOTER: A Runtime Assurance Framework for Programming Safe Robotics Systems". In: *Proceedings - 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, DSN 2019* (2019), pp. 138–150. DOI: 10.1109/DSN.2019.00027. arXiv: 1808.07921.

[26] Frederik Diehl, Thomas Brunner, Michael Truong Le, and Alois Knoll. "Graph neural networks for modelling traffic participant interaction". In: *IEEE Intelligent Vehicles Symposium, Proceedings*. Vol. 2019-June. 2019, pp. 695–701. ISBN: 9781728105604. DOI: 10.1109/IVS.2019.8814066. arXiv: 1903.01254.

[27] Moritz Diehl. *Direct single and multiple shooting - Lecture notes*. 2013.

[28] Kurt Driessens and Sašo Džeroski. "Integrating guidance into relational reinforcement learning". In: *Machine Learning* 57.3 (2004), pp. 271–304. ISSN: 08856125. DOI: 10.1023/B:MACH.0000039779.47329.3a.

[29] Wayne Eberly, Mark Giesbrecht, Pascal Giorgi, Arne Storjohann, and Gilles Villard. "Solving Sparse Integer Linear Systems". In: *arXiv* (2006). arXiv: 0603082 [cs].

[30] Klemens Esterle, Luis Gressenbuch, and Alois Knoll. "Formalizing Traffic Rules for Machine Interpretability". In: *arXiv* (2020). ISSN: 23318422. DOI: 10.1109/cavs51000.2020.9334599. arXiv: 2007.00330.

[31] Klemens Esterle, Tobias Kessler, and Alois Knoll. "Optimal Behavior Planning for Autonomous Driving: A Generic Mixed-Integer Formulation". In: *arXiv* (2020). arXiv: 2003.13312.

[32] Christian Frese and Jürgen Beyerer. "A comparison of motion planning algorithms for cooperative collision avoidance of multiple cognitive automobiles". In: *IEEE Intelligent Vehicles Symposium (IV)*. 2011, pp. 1156–1162. ISBN: 9781457708909. DOI: 10.1109/IVS.2011.5940489.

[33] Scott Fujimoto, Herke Van Hoof, and David Meger. "Addressing Function Approximation Error in Actor-Critic Methods". In: *35th International Conference on Machine Learning, ICML 2018* 4 (2018), pp. 2587–2601. ISSN: 1938-7228. arXiv: 1802.09477.

[34] P. J. Gage, R. D. Braun, and I. M. Kroo. "Interplanetary trajectory optimization using a genetic algorithm". In: *Astrodynamics Conference, 1994* (1994), pp. 538–547. DOI: 10.2514/6.1994-3773.

[35] Claudio Gallicchio and Alessio Micheli. "Graph echo state networks". In: *Proceedings of the International Joint Conference on Neural Networks* (2010). DOI: 10.1109/IJCNN.2010.5596796.

[36] Javier García and Fernando Fernández. "A comprehensive survey on safe reinforcement learning". In: *Journal of Machine Learning Research* 16 (2015), pp. 1437–1480. ISSN: 15337928.

[37] Clement Gehring and Doina Precup. "Smart exploration in reinforcement learning using absolute temporal difference errors". In: *12th International Conference on Autonomous Agents and Multiagent Systems 2013, AAMAS 2013* 2 (2013), pp. 1037–1043.

[38] Philip E. Gill, Walter Murray, and Michael A. Saunders. "SNOPT: An SQP algorithm for large-scale constrained optimization". In: *SIAM Review* 47.1 (2005), pp. 99–131. ISSN: 00361445. DOI: 10.1137/S0036144504446096.

[39] Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. "Neural message passing for quantum chemistry". In: *34th International Conference on Machine Learning, ICML 2017* 3 (2017), pp. 2053–2070. arXiv: 1704.01212.

[40] Marco Gori, Gabriele Monfardini, and Franco Scarselli. "A new model for earning in raph domains". In: *Proceedings of the International Joint Conference on Neural Networks* 2 (2005), pp. 729–734. DOI: 10.1109/IJCNN.2005.1555942.

[41] Stephen Gou and Yuyang Liu. "DQN with model-based exploration: Efficient learning on environments with sparse rewards". In: *arXiv* (2019), pp. 1–10. ISSN: 23318422. arXiv: 1903.09295.

[42] Benjamin Gutjahr, Lutz Gröll, and Moritz Werling. "Lateral Vehicle Trajectory Optimization Using Constrained Linear Time-Varying MPC". In: *IEEE Transactions on Intelligent Transportation Systems* 18.6 (2017), pp. 1586–1595. ISSN: 15249050. DOI: 10.1109/TITS.2016.2614705.

[43] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor". In: *35th International Conference on Machine Learning, ICML 2018*. Ed. by Jennifer G Dy and Andreas Krause. Vol. 5. Proceedings of Machine Learning Research. PMLR, 2018, pp. 2976–2989. ISBN: 9781510867963. DOI: arXiv:1801.01290v2. arXiv: 1801.01290.

[44] Patrick Hart and Alois Knoll. "Counterfactual Policy Evaluation for Decision-Making in Autonomous Driving". In: *arXiv* (2020). arXiv: 2003.11919.

[45] Patrick Hart and Alois Knoll. "Graph Neural Networks and Reinforcement Learning for Behavior Generation in Semantic Environments". In: *IEEE Intelligent Vehicles Symposium, Proceedings*. 2020, pp. 1589–1594. DOI: 10.1109/IV47402.2020.9304738. arXiv: 2006.12576.

[46] Patrick Hart, Leonard Rychly, and Alois Knoll. "Lane-Merging Using Policy-based Reinforcement Learning and Post-Optimization". In: *2019 IEEE Intelligent Transportation Systems Conference, ITSC 2019* (2019), pp. 3176–3181. DOI: 10.1109/ITSC.2019.8917002. arXiv: 2003.03168.

[47] Matthias Heger. "Consideration of Risk in Reinforcement Learning". In: *Machine Learning Proceedings 1994* (1994), pp. 105–111. DOI: 10.1016/b978-1-55860-335-6.50021-0.

[48] Matteo Hessel, Joseph Modayil, Hado Van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, and David Silver. "Rainbow: Combining improvements in deep reinforcement learning". In: *32nd AAAI Conference on Artificial Intelligence, AAAI 2018* (2018), pp. 3215–3222. arXiv: 1710.02298.

[49] Carl Johan Hoel, Katherine Driggs-Campbell, Krister Wolff, Leo Laine, and Mykel J. Kochenderfer. "Combining Planning and Deep Reinforcement Learning in Tactical Decision Making for Autonomous Driving". In: *IEEE Transactions on Intelligent Vehicles* 5.2 (2020), pp. 294–305. ISSN: 23798858. DOI: 10.1109/TIV.2019.2955905. arXiv: 1905.02680.

[50] Brian Ichter, James Harrison, and Marco Pavone. "Learning Sampling Distributions for Robot Motion Planning". In: *Proceedings - IEEE International Conference on Robotics and Automation* (2018), pp. 7087–7094. ISSN: 10504729. DOI: 10.1109/ICRA.2018.8460730. arXiv: 1709.05448.

[51] International Organization for Standardization. *Road vehicles - Safety of the intended functionality*. Norm. 2019.

[52] Iso 26262. *Road vehicles - Functional Safety Standard*. Norm. 2009.

[53] Ashesh Jain, Amir R. Zamir, Silvio Savarese, and Ashutosh Saxena. "Structural-RNN: Deep learning on spatio-temporal graphs". In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition* 2016-Decem (2016), pp. 5308–5317. ISSN: 10636919. DOI: 10.1109/CVPR.2016.573. arXiv: 1511.05298.

[54] Sham Kakade and John Langford. "Approximately Optimal Approximate Reinforcement Learning". In: *Proceedings of the 19th International Conference on Machine Learning* (2002), pp. 267–274.

[55] Sertac Karaman and Emilio Frazzoli. "Sampling-based algorithms for optimal motion planning". In: *International Journal of Robotics Research* 30.7 (2011), pp. 846–894. ISSN: 02783649. DOI: 10.1177/0278364911406761. arXiv: 1105.1186.

[56] Matthew Kelly. "An introduction to trajectory optimization: How to do your own direct collocation*". In: *SIAM Review* 59.4 (2017), pp. 849–904. ISSN: 00361445. DOI: 10.1137/16M1062569.

[57] A Kesting, M Treiber, and D Helbing. "General lane-changing model MOBIL for car-following models". In: *Transportation Research Record* 1999.1 (2007).

[58] Jeong Jung Kim and Ju Jang Lee. "Trajectory optimization with particle swarm optimization for manipulator motion planning". In: *IEEE Transactions on Industrial Informatics* 11.3 (2015), pp. 620–631. ISSN: 15513203. DOI: 10.1109/TII.2015.2416435.

[59] Thomas N. Kipf and Max Welling. "Semi-supervised classification with graph convolutional networks". In: *5th International Conference on Learning Representations, ICLR 2017 - Conference Track Proceedings* (2017), pp. 1–14. arXiv: 1609.02907.

[60] Mykel J Kochenderfer. *Decision Making Under Uncertainty: Theory and Application*. 2015.

[61] Torsten Koller, Felix Berkenkamp, Matteo Turchetta, and Andreas Krause. "Learning-based model predictive control for safe exploration". In: *arXiv* (2018). ISSN: 23318422.

[62] Kraftfahrt Bundesamt (KBA). *Entwicklung der Fahrleistungen nach Fahrzeugarten seit 2015*. Tech. rep. 2019, pp. 1–9.

[63] Karl Kurzer, Chenyang Zhou, and J Marius Zöllner. "Decentralized Cooperative Planning for Automated Vehicles with Hierarchical Monte Carlo Tree Search". In: *IEEE Intelligent Vehicles Symposium (IV)*. 2018, pp. 529–536. ISBN: 9781538644522. DOI: 10.1109/IVS.2018.8500712. arXiv: 1809.03200.

[64] Benjamin James Lansdell, Konrad Paul Kording, and Prashanth Ravi Prakash. "Learning to solve the credit assignment problem". In: *arXiv* (2019), pp. 1–19. ISSN: 23318422. arXiv: 1906.00889.

[65] Martin Lauer. *Learning in Multi Agent Environments Dr.* 2019.

[66] Christopher Lazarus, James G. Lopez, and Mykel J. Kochenderfer. "Runtime safety assurance using reinforcement learning". In: *AIAA/IEEE Digital Avionics Systems Conference - Proceedings* 2020-Octob (2020). ISSN: 21557209. DOI: 10.1109/DASC50938.2020.9256446. arXiv: 2010.10618.

[67] Donsuk Lee, Yiming Gu, Jerrick Hoang, and Micol Marchetti-Bowick. "Joint interaction and trajectory prediction for autonomous driving using graph neural networks". In: *arXiv* NeurIPS (2019). ISSN: 23318422. arXiv: 1912.07882.

[68] David Lenz, Tobias Kessler, and Alois Knoll. "Tactical Cooperative Planning for Autonomous Vehicles using MCTS". In: *IEEE Intelligent Vehicles Symposium (IV)*. 2016, pp. 447–453. ISBN: 9781509018208. DOI: 10.1109/IVS.2016.7535424.

[69] Yaguang Li, Rose Yu, Cyrus Shahabi, and Yan Liu. "Diffusion convolutional recurrent neural network: Data-driven traffic forecasting". In: *arXiv* (2017), pp. 1–16. arXiv: 1707.01926.

[70] Yunzhu Li, Jiaming Song, and Stefano Ermon. "InfoGAIL: Interpretable imitation learning from visual demonstrations". In: *Advances in Neural Information Processing Systems* 2017-Decem.Nips (2017), pp. 3813–3823. ISSN: 10495258. arXiv: 1703.08840.

[71] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. "Continuous control with deep reinforcement learning". In: *4th International Conference on Learning Representations, ICLR 2016 - Conference Track Proceedings* (2016). arXiv: 1509.02971.

[72] Maja J Mataric. "Reward functions for Acclerated Learning". In: *In Proceedings of the Eleventh International Conference on Machine Learning* (1194).

[73] Usama Mehmood, Stanley Bak, Scott A. Smolka, and Scott D. Stoller. "Safe CPS from Unsafe Controllers". In: *arXiv* (2021). arXiv: 2102.12981.

[74] Kunal Menda, Yi Chun Chen, Justin Grana, James W. Bono, Brendan D. Tracey, Mykel J. Kochenderfer, and David Wolpert. "Deep reinforcement learning for event-driven multi-agent decision processes". In: *arXiv* 20.4 (2017), pp. 1259–1268. ISSN: 23318422.

[75] Alessio Micheli. "Neural network for graphs: A contextual constructive approach". In: *IEEE Transactions on Neural Networks* 20.3 (2009), pp. 498–511. ISSN: 10459227. DOI: 10.1109/TNN.2008.2010350.

[76] Alessio Micheli, Diego Sona, and Alessandro Sperduti. "Contextual processing of structured data by recursive cascade correlation". In: *IEEE Transactions on Neural Networks* 15.6 (2004), pp. 1396–1410. ISSN: 10459227. DOI: 10.1109/TNN.2004.837783.

[77] Oliver Mihatsch and Ralph Neuneier. "Risk-sensitive reinforcement learning". In: *Machine Learning* 49.2-3 (2002), pp. 267–290. ISSN: 08856125. DOI: 10.1023/A:1017940631555.

[78] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. "Playing Atari with Deep Reinforcement Learning". In: *arXiv* (2013), pp. 1–9. arXiv: 1312.5602.

[79] Teodor Mihai Moldovan and Pieter Abbeel. "Safe exploration in Markov decision processes". In: *Proceedings of the 29th International Conference on Machine Learning, ICML 2012* 2 (2012), pp. 1711–1718. arXiv: 1205.4810.

[80] Tetsuro Morimura, Masashi Sugiyama, Hisashi Kashima, Hirotaka Hachiya, and Toshiyuki Tanaka. "Parametric return density estimation for Reinforcement Learning". In: *Proceedings of the 26th Conference on Uncertainty in Artificial Intelligence, UAI 2010* (2010), pp. 368–375. arXiv: 1203.3497.

[81] Andrew Y. Ng, Daishi Harada, and Stuart Russell. "Policy invariance under reward transformations : Theory and application to reward shaping". In: *Sixteenth International Conference on Machine Learning* 3 (1999), pp. 278–287. ISSN: 1098-6596. arXiv: arXiv:1011.1669v3.

[82] Julia Nilsson, Mattias Brännström, Jonas Fredriksson, and Erik Coelingh. "Longitudinal and Lateral Control for Automated Yielding Maneuvers". In: *IEEE Transactions on Intelligent Transportation Systems* 17.5 (2016), pp. 1404–1414. ISSN: 15249050. DOI: 10.1109/TITS.2015.2504718.

[83] Brian Paden, Michal Čáp, Sze Zheng Yong, Dmitry Yershov, and Emilio Frazzoli. "A survey of motion planning and control techniques for self-driving urban vehicles". In: *IEEE Transactions on Intelligent Vehicles* 1.1 (2016), pp. 33–55. ISSN: 23798858. DOI: 10.1109/TIV.2016.2578706. arXiv: 1604.07446.

[84] Jia Pan, Zhuo Chen, and Pieter Abbeel. "Predicting initialization effectiveness for trajectory optimization". In: *Proceedings - IEEE International Conference on Robotics and Automation* (2014), pp. 5183–5190. ISSN: 10504729. DOI: 10.1109/ICRA.2014.6907620.

[85] Christian Pek and Matthias Althoff. "High-level Decision Making for Safe and Reasonable Autonomous Lane Changing using Reinforcement Learning". In: *International Conference on Intelligent Transportation Systems* (2018).

[86] 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. ISSN: 25225839. DOI: 10.1038/s42256-020-0225-y.

[87] Romain Pepy, Alain Lambert, and Hugues Mounier. "Path planning using a dynamic vehicle model". In: *2006 2nd International Conference on Information & Communication Technologies*. Vol. 1. IEEE. 2006, pp. 781–786.

[88] Mihail Pivtoraiko, Ross A. Knepper, and Alonzo Kelly. "Differentially constrained mobile robot motion planning in state lattices". In: *Journal of Field Robotics* 26.3 (2009), pp. 308–333. ISSN: 15564959. DOI: 10.1002/rob.20285. arXiv: 10.1.1.91.5767.

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

[90] B. Ravi Kiran, Ibrahim Sobh, Victor Talpaert, Patrick Mannion, Ahmad A. Al Sallab, Senthil Yogamani, and Patrick Pérez. "Deep reinforcement learning for autonomous driving: A survey". In: *arXiv* (2020), pp. 1–18. ISSN: 23318422. DOI: 10.1109/tits.2021.3054625. arXiv: 2002.00444.

[91] Michael T. Rosenstein and Andrew G. Barto. "Supervised actor-critic reinforcement learning". In: *Handbook of Learning and Approximate Dynamic Programming* (2004), pp. 359–380. DOI: 10.1109/9780470544785.ch14.

[92] Stéphane Ross, Geoffrey J. Gordon, and J. Andrew Bagnell. "A reduction of imitation learning and structured prediction to no-regret online learning". In: *Journal of Machine Learning Research* 15 (2011), pp. 627–635. ISSN: 15324435. arXiv: 1011.0686.

[93] G A Rummery and M Niranjan. "On-line Q-learning using connectionist systems". In: *On-line Q-learning using connectionist systems* 37.September (1994), p. 20.

[94] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. "Prioritized experience replay". In: *4th International Conference on Learning Representations, ICLR 2016 - Conference Track Proceedings* (2016), pp. 1–21. arXiv: 1511.05952.

[95] John Schulman, Sergey Levine, Philipp Moritz, Michael Jordan, and Pieter Abbeel. "Trust region policy optimization". In: *32nd International Conference on Machine Learning, ICML 2015* 3 (2015), pp. 1889–1897. arXiv: 1502.05477.

[96] John Schulman, Philipp Moritz, Sergey Levine, Michael I. Jordan, and Pieter Abbeel. "High-dimensional continuous control using generalized advantage estimation". In: *4th International Conference on Learning Representations, ICLR 2016 - Conference Track Proceedings* (2016), pp. 1–14. arXiv: 1506.02438.

[97] Lui Sha. "Using simplicity to control complexity". In: *IEEE Software* 18.4 (2001), pp. 20–28. ISSN: 07407459. DOI: 10.1109/MS.2001.936213.

[98] Shai Shalev-Shwartz, Shaked Shammah, and Amnon Shashua. "On a Formal Model of Safe and Scalable Self-driving Cars". In: *arXiv* (2017), pp. 1–37. ISSN: 23318422. arXiv: 1708.06374.

[99] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. "Mastering the game of Go with deep neural networks and tree search". In: *Nature* 529.7587 (2016), pp. 484–489. ISSN: 14764687. DOI: 10.1038/nature16961.

[100] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy Lillicrap, Karen Simonyan, and Demis Hassabis. "Mastering chess and shogi by self-play with a general reinforcement learning algorithm". In: *arXiv* (2017), pp. 1–19. arXiv: 1712.01815.

[101] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. "Deterministic policy gradient algorithms". In: *31st International Conference on Machine Learning, ICML 2014* 1 (2014), pp. 605–619.

[102] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, Yutian Chen, Timothy Lillicrap, Fan Hui, Laurent Sifre, George Van Den Driessche, Thore Graepel, and Demis Hassabis. "Mastering the game of Go without human knowledge". In: *Nature* 550.7676 (2017), pp. 354–359. ISSN: 14764687. DOI: 10.1038/nature24270.

[103] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, Yutian Chen, Timothy Lillicrap, Fan Hui, Laurent Sifre, George Van Den Driessche, Thore Graepel, and Demis Hassabis. "Mastering the game of Go without human knowledge". In: *Nature* 550.7676 (2017), pp. 354–359. ISSN: 14764687. DOI: 10.1038/nature24270.

[104] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. "Deep inside convolutional networks: Visualising image classification models and saliency maps". In: *2nd International Conference on Learning Representations, ICLR 2014 - Workshop Track Proceedings* (2014), pp. 1–8. arXiv: 1312.6034.

[105] Alessandro Sperduti and Antonina Starita. "Supervised neural networks for the classification of structures". In: *IEEE Transactions on Neural Networks* 8.3 (1997), pp. 714–735. ISSN: 10459227. DOI: 10.1109/72.572108.

[106] 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* 2017-Decem.Nips 2017 (2017), pp. 3309–3319. ISSN: 10495258. arXiv: 1705.07761.

[107] Statistisches Bundesamt (Destatis). "Fehlverhalten der Fahrzeugfuehrer bzw. Pkw-Fahrer bei Unfaellen mit Personenschaden 1991 – 2019". In: *Statistisches Bundesamt (Destatis)* 49.0 (2020).

[108] Bartolomeo Stellato, Goran Banjac, Paul Goulart, Alberto Bemporad, and Stephen Boyd. "OSQP: an operator splitting solver for quadratic programs". In: *Mathematical Programming Computation* 12.4 (2020), pp. 637–672. ISSN: 18672957. DOI: 10.1007/s12532-020-00179-2. arXiv: 1711.08013.

[109] Petr Stodola, Jan Drozd, Jan Nohel, Jan Hodický, and Dalibor Procházka. "Trajectory optimization in a cooperative aerial reconnaissance model". In: *Sensors (Switzerland)* 19.12 (2019), pp. 1–18. ISSN: 14248220. DOI: 10.3390/s19122823.

[110] Etsuko Sugawara and Hiroshi Nikaido. *Properties of AdeABC and AdeIJK efflux systems of Acinetobacter baumannii compared with those of the AcrAB-TolC system of Escherichia coli*. Tech. rep. 12. 2014, pp. 7250–7257. DOI: 10.1128/AAC.03728-14.

[111] Richard S Sutton and Andrew G Barto. *Reinforcement Learning: An Introduction, Second Edition*. Vol. 258. 2018. ISBN: 0262193981.

[112] Aviv Tamar, Dotan Di Castro, and Shie Mannor. "Policy gradients with variance related risk criteria". In: *Proceedings of the 29th International Conference on Machine Learning, ICML 2012* 1 (2012), pp. 935–942.

[113] Martin Treiber, Ansgar Hennecke, and Dirk Helbing. "Congested Traffic States in Empirical Observations and Microscopic Simulations". In: (2008). arXiv: 0002177v2 [arXiv:cond-mat].

[114] Hado Van Hasselt. "Double Q-learning". In: *Advances in Neural Information Processing Systems 23: 24th Annual Conference on Neural Information Processing Systems 2010, NIPS 2010* (2010), pp. 1–9.

[115] Hado Van Hasselt, Arthur Guez, and David Silver. "Deep reinforcement learning with double Q-Learning". In: *30th AAAI Conference on Artificial Intelligence, AAAI 2016* (2016), pp. 2094–2100. arXiv: 1509.06461.

[116] Robert J. Vanderbei. "LOQO: An interior point code for quadratic programming". In: *Optimization Methods and Software* 11.1 (1999), pp. 451–484. ISSN: 10556788. DOI: 10.1080/10556789908805759.

[117] Oriol Vinyals, Timo Ewalds, Sergey Bartunov, Petko Georgiev, Alexander Sasha Vezhnevets, Michelle Yeo, Alireza Makhzani, Heinrich Küttler, John Agapiou, Julian Schrittwieser, John Quan, Stephen Gaffney, Stig Petersen, Karen Simonyan, Tom Schaul, Hado van Hasselt, David Silver, Timothy Lillicrap, Kevin Calderone, Paul Keet, Anthony Brunasso, David Lawrence, Anders Ekermo, Jacob Repp, and Rodney Tsing. "StarCraft II: A new challenge for reinforcement learning". In: *arXiv* (2017). ISSN: 23318422. arXiv: 1708.04782.

[118] Riccardo Volpi and Vittorio Murino. "Addressing model vulnerability to distributional shifts over image transformation sets". In: *arXiv* (2019). ISSN: 23318422.

[119] Andreas Wächter. "Short Tutorial: Getting Started With Ipopt in 90 Minutes". In: *Dagstuhl Seminar Proceedings* (2009), pp. 1–17. ISSN: 1862-4405.

[120] Jiankun Wang, Wenzheng Chi, Chenming Li, Chaoqun Wang, and Max Q.H. Meng. "Neural RRT*: Learning-Based Optimal Path Planning". In: *IEEE Transactions on Automation Science and Engineering* 17.4 (2020), pp. 1748–1758. ISSN: 15583783. DOI: 10.1109/TASE.2020.2976560.

[121] Tingwu Wang, Renjie Liao, Jimmy Ba, and Sanja Fidler. "Nervenet: Learning structured policy with graph neural networks". In: *6th International Conference on Learning Representations, ICLR 2018 - Conference Track Proceedings* (2018).

[122] Xinyi Wang, Hieu Pham, Paul Michel, Antonios Anastasopoulos, Graham Neubig, and Jaime Carbonell. "Optimizing data usage via differentiable rewards". In: *arXiv* (2019), pp. 1–15. arXiv: 1911.10088.

[123] Christopher J C H Watkins. *Learning from delayed rewards*. 1989.

[124] Ronald J. Williams. "Simple statistical gradient-following algorithms for connectionist reinforcement learning". In: *Machine Learning* 8.3-4 (1992), pp. 229–256. ISSN: 0885-6125. DOI: 10.1007/bf00992696.

[125] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S. Yu. "A comprehensive survey on graph neural networks". In: *arXiv* XX.Xx (2019), pp. 1–22. ISSN: 2162-237X. DOI: 10.1109/tnnls.2020.2978386. arXiv: 1901.00596.

[126] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. "How transferable are features in deep neural networks?" In: *Advances in Neural Information Processing Systems* 4.January (2014), pp. 3320–3328. ISSN: 10495258. arXiv: 1411.1792.

[127] Changxi You, Jianbo Lu, Dimitar Filev, and Panagiotis Tsiotras. "Advanced planning for autonomous vehicles using reinforcement learning and deep inverse reinforcement learning". In: *Robotics and Autonomous Systems* 114 (2019), pp. 1–18. ISSN: 09218890. DOI: 10.1016/j.robot.2019.01.003.

[128] Manzil Zaheer, Satwik Kottur, Siamak Ravanbhakhsh, Barnabás Póczos, Ruslan Salakhutdinov, and Alexander J. Smola. "Deep sets". In: *Advances in Neural Information Processing Systems* 2017-Decem.ii (2017), pp. 3392–3402. ISSN: 10495258. arXiv: 1703.06114.

[129] Wei Zhan, Liting Sun, Di Wang, Haojie Shi, Aubrey Clausse, Maximilian Naumann, Julius Kümmerle, Hendrik Königshof, Christoph Stiller, Arnaud de la Fortelle, and Masayoshi Tomizuka. "Interaction dataset: An international, adversarial and cooperative motion dataset in interactive driving scenarios with semantic maps". In: *arXiv* (2019). ISSN: 23318422. arXiv: 1910.03088.

[130] Weichao Zhou, Ruihan Gao, Baek Gyu Kim, Eunsuk Kang, and Wenchao Li. "Runtime-Safety-Guided Policy Repair". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 12399 LNCS (2020), pp. 131–150. ISSN: 16113349. DOI: 10.1007/978-3-030-60508-7_7. arXiv: 2008.07667.

[131] Z. J. Zhou and N. N. Yan. "A survey of numerical methods for convection-diffusion optimal control problems". In: *Journal of Numerical Mathematics* 22.1 (2014), pp. 61–85. ISSN: 15702820. DOI: 10.1515/jnum-2014-0003.

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

[133] Julius Ziegler, Philipp Bender, Markus Schreiber, Henning Lategahn, Tobias Strauss, Christoph Stiller, Thao Dang, Uwe Franke, Nils Appenrodt, Christoph G. Keller, Eberhard Kaus, Ralf G. Herrtwich, Clemens Rabe, David Pfeiffer, Frank Lindner, Fridtjof Stein, Friedrich Erbs, Markus Enzweiler, Carsten Knoppel, Jochen Hipp, Martin Haueis, Maximilian Trepte, Carsten Brenk, Andreas Tamke, Mohammad Ghanaat, Markus Braun, Armin Joos, Hans Fritz, Horst Mock, Martin Hein, and Eberhard Zeeb. "Making bertha drive-an autonomous journey on a historic route". In: *IEEE Intelligent Transportation Systems Magazine* 6.2 (2014), pp. 8–20. ISSN: 19391390. DOI: 10.1109/MITS.2014.2306552.

A Appendix

A.1 Architectures and Hyperparameters

Used hyperparameters for the conventional *DNN large* large agent:

```
{
  "BehaviorSACAgent": {
    "ActorFcLayerParams": [512, 512, 512],
    "CriticJointFcLayerParams": [512, 512, 512],
    "ActorLearningRate": 0.0003,
    "CriticLearningRate": 0.0003,
    "AlphaLearningRate": 0.0003,
    "TargetUpdateTau": 0.05,
    "TargetUpdatePeriod": 3,
    "Gamma": 0.995,
    "RewardScaleFactor": 1.0,
    "AgentName": "sac_agent",
    "DebugSummaries": false,
    "ReplayBufferCapacity": 10000,
    "ParallelBufferCalls": 1,
    "BatchSize": 512,
    "BufferNumSteps": 2,
    "BufferPrefetch": 3
  }
}
```

Used hyperparameters for the *GNN large* soft actor critic (SAC) agent:

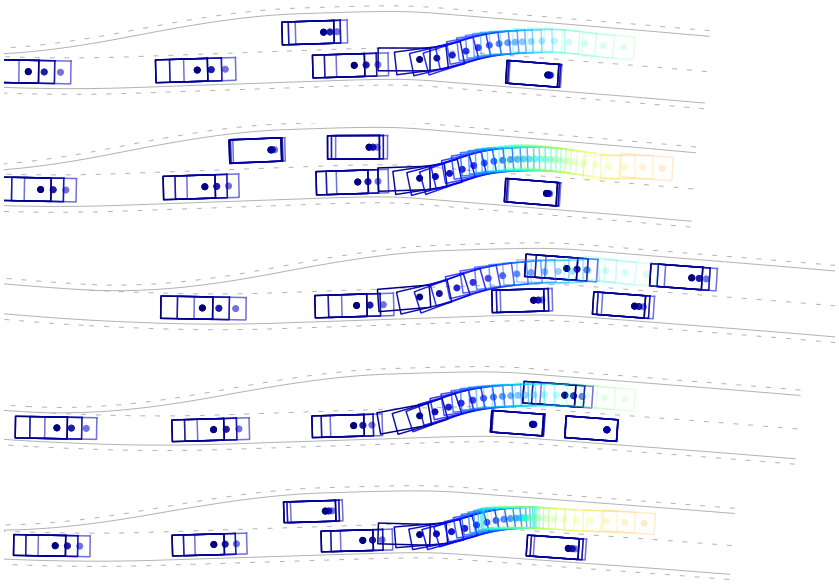
```
{
  "BehaviorGraphSACAgent": {
    "ActorFcLayerParams": [512, 512],
    "CriticObservationFcLayerParams": null,
    "CriticActionFcLayerParams": null,
    "CriticJointFcLayerParams": [512, 512],
    "ActorLearningRate": 0.0003,
    "CriticLearningRate": 0.0003,
    "AlphaLearningRate": 0.0003,
    "TargetUpdateTau": 0.05,
```

```
        "TargetUpdatePeriod": 3,
        "Gamma": 0.995,
        "RewardScaleFactor": 1.0,
        "AgentName": "gnn_sac_agent",
        "DebugSummaries": false,
        "ReplayBufferCapacity": 10000,
        "ParallelBufferCalls": 1,
        "BatchSize": 512,
        "BufferNumSteps": 2,
        "BufferPrefetch": 3
    },
}
```

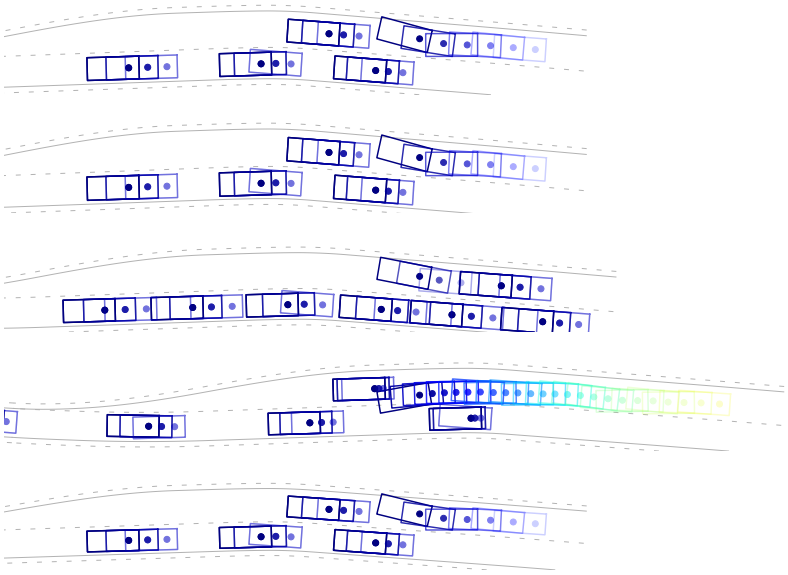
Default intelligent driver model (IDM) and minimizing overall braking induced by lane changes (MOBIL) parameters:

```
{
    "IDM": {
        "BrakeForLaneEnd": true,
        "BrakeForLaneEndEnabledDistance": 100.0,
        "BrakeForLaneEndDistanceOffset": 30.0,
        "DesiredVelocity": 10.0,
        "MinimumSpacing": 2.0,
        "DesiredTimeHeadway": 1.5,
        "MaxAcceleration": 1.7,
        "ComfortableBrakingAcceleration": 1.66,
        "MinVelocity": 0.0,
        "MaxVelocity": 50.0,
        "Exponent": 4,
        "NumTrajectoryTimePoints": 11,
        "CoolnessFactor": 0.0,
        "AccelerationUpperBound": 8.0,
        "AccelerationLowerBound": -5.0
    },
    "Mobil": {
        "AThr": 0.2,
        "Politeness": 0.2,
        "SafeDeceleration": 4.0
    }
}
```

A.2 Successful and Colliding Scenarios



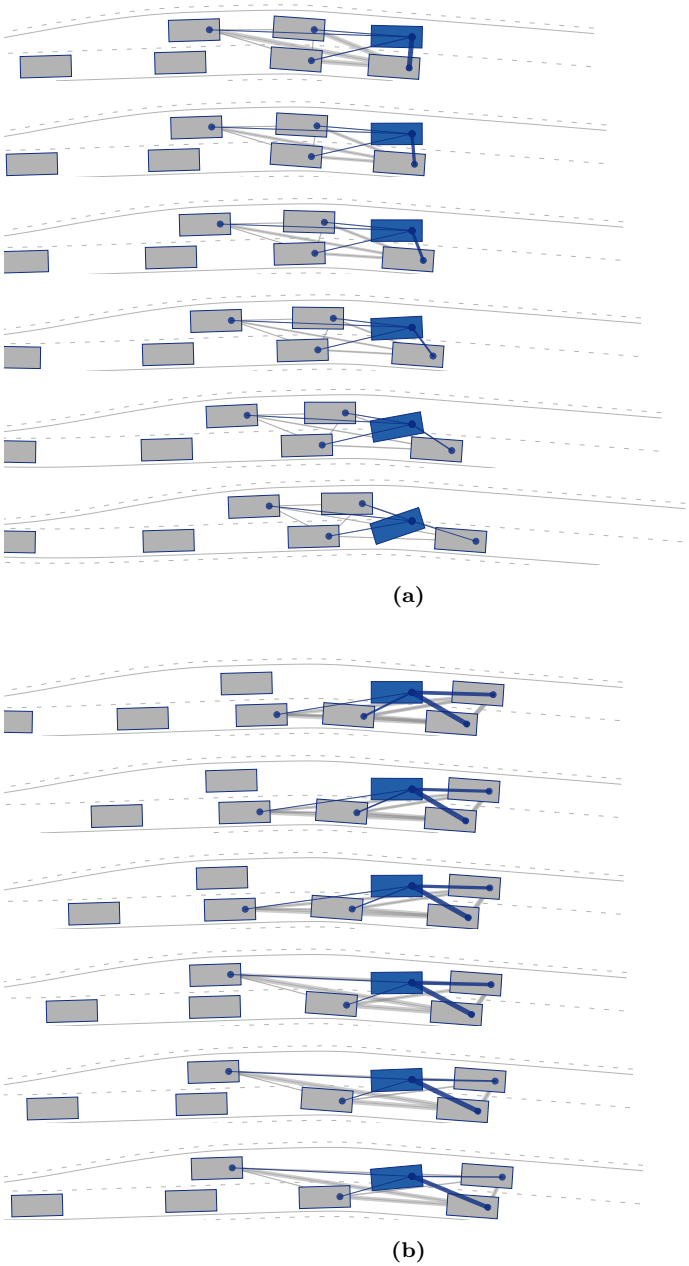
(a) Successful episodes with the *GNN large* architecture.



(b) Colliding episodes with the *GNN large* architecture.

Figure A.1: Successful and colliding scenarios using the *GNN large* architecture.

A.3 Graph Neural Network Visualizations

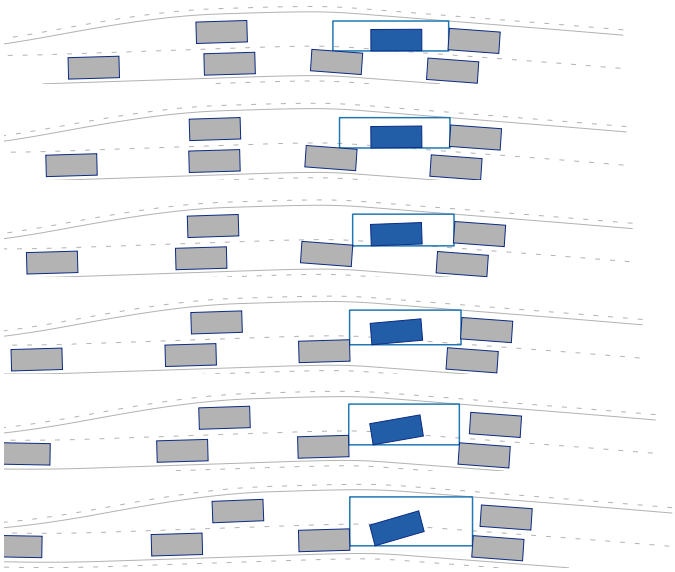


(a)

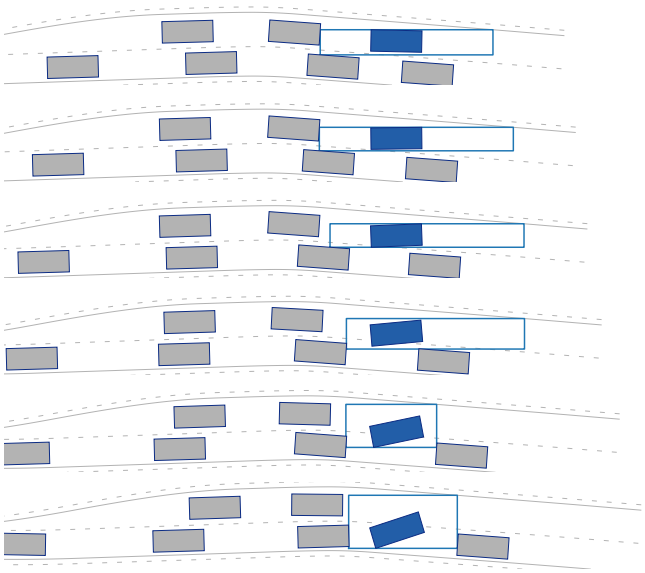
(b)

Figure A.2: Graph edges visualized for two episodes using the *GNN large* architecture.

A.4 Extracted Post-Optimization Constraints



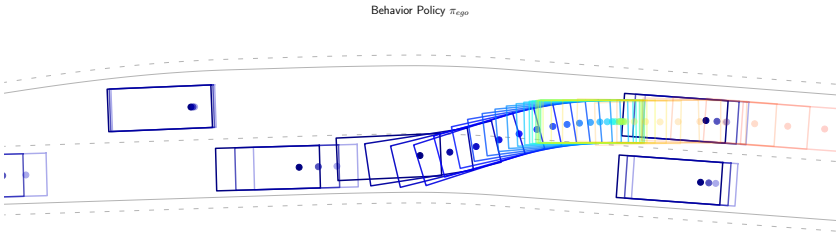
(a)

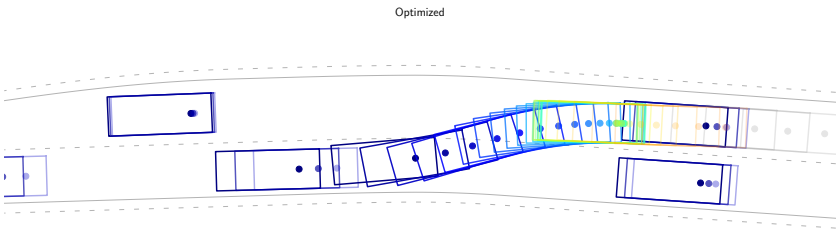


(b)

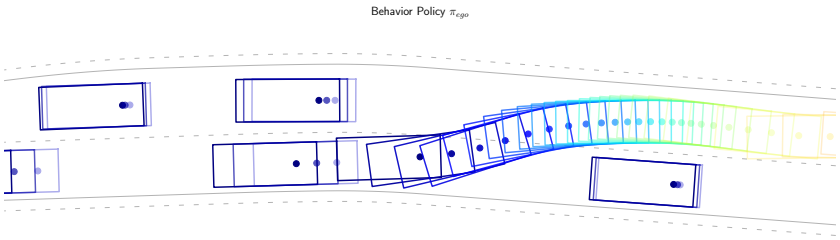
Figure A.3: Extracted constraint bounding boxes for the post-optimization.

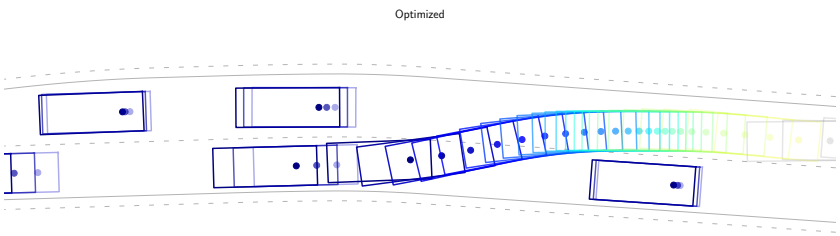
A.5 Qualitative Results of the Post-Optimization





(a)





(b)

Figure A.4: Trajectories produced by the *GNN large* architecture and the respective post-optimized trajectories.