Implicit Cooperative Decision-Making for Automated Vehicles

Zur Erlangung des akademischen Grades eines

Doktors der Ingenieurwissenschaften (Dr.-Ing.)

von der KIT-Fakultät für Wirtschaftswissenschaften
des Karlsruher Instituts für Technologie (KIT)

genehmigte

DISSERTATION

von

M.Sc. Karl Kurzer

Tag der mündlichen Prüfung:	08.11.2023
Hauptreferent:	Prof. Dr.-Ing. J. Marius Zöllner
Korreferent:	Prof. Dr.-Ing. Christoph Stiller

Karlsruhe (2023)

Search and Learn

Implicit Cooperative Decision-Making for Automated Vehicles

Karl Kurzer

Kurzfassung

Die Fähigkeiten von automatisierten Fahrzeugen entwickeln sich zwar rasch weiter, doch fehlt ihnen noch immer eine wesentliche Komponente, die sie von ihren menschlichen Gegenstücken unterscheidet: die Fähigkeit zur impliziten Kooperation mit anderen. Die Interaktionen zwischen automatisierten Fahrzeugen und Menschen stellen nach wie vor eine Herausforderung im Bereich des automatisierten Fahrens dar. Um diese zu überwinden, ist es erforderlich, künftige automatisierte Fahrzeuge mit der Fähigkeit auszustatten, implizit Kooperation einzufordern und anbieten zu können, um eine reibungslose Integration in den heutigen heterogenen Verkehr zu ermöglichen.

Diese Form des Vorausschauens kann erreicht werden, indem alle möglichen Aktionen der Verkehrsteilnehmer berücksichtigt werden. Diese Betrachtung schafft jedoch eine Interdependenz zwischen den Aktionen der Verkehrsteilnehmer, die eine Kombination der Prädiktion und der Planung in einem Multiagenten-Markov-Entscheidungsprozess erfordert.

In dieser Arbeit wird ein System vorgeschlagen, das Prädiktion und Planung kombiniert, indem das Problem als ein Multiagenten-Markov-Entscheidungsprozess modelliert wird, der alle möglichen Aktionen der Verkehrsteilnehmer berücksichtigt. Monte Carlo Tree Search wird in Verbindung mit Decoupled Upper Confidence Bound for Trees eingesetzt, um nahezu optimale Trajektorien für alle Verkehrsteilnehmer zu identifizieren, was zu einem Trajektorienplaner führt, welcher implizite Kooperation zwischen Verkehrsteilnehmern ermöglicht.

Verschiedene Verbesserungen des Basisalgorithmus, wie Parallelisierung und Hyperparameter-Optimierung, wurden entwickelt, um die Leistung zu steigern und bessere Lösungen in kürzerer Zeit zu erzeugen. Darüber hinaus wurde der

Planer mit gelernten Belohnungsmodellen auf der Grundlage von Expertentrajektorien erweitert, basierend auf inversem Reinforcement Learning, die es ihm ermöglichen, sich an den gewünschten menschlichen Fahrstil anzupassen, um eine reibungslose Integration in menschenzentrierten Verkehr zu ermöglichen.

Die Wirksamkeit des vorgeschlagenen Ansatzes wurde in 15 anspruchsvollen Multiagentenszenarien demonstriert, wobei sich die Erfolgsquote im Vergleich zu einer Basislösung deutlich verbesserte.

Abstract

While the capabilities of Automated Vehicles (AVs) are evolving rapidly, they still lack an essential component that distinguishes them from their human counterparts: the ability to cooperate implicitly with others. The interactions between AVs and humans continue to pose challenges in the field of Automated Driving (AD). To overcome this, it is required to equip future AVs with the ability to implicitly demand and provide cooperation where necessary to integrate smoothly into today's heterogeneous traffic.

This form of anticipation can be achieved by considering all possible actions of traffic participants. However, this consideration creates an interdependency between traffic participants' actions that requires combining the prediction and the planning into a Multi-agent Markov Decision Process (MMDP).

In this work, a system is proposed that combines prediction and planning by modeling the problem as an MMDP, which considers all possible actions of traffic participants. Monte Carlo Tree Search (MCTS), in conjunction with Decoupled Upper Confidence Bound for Trees (DUCT), is employed to identify near-optimal trajectories for all traffic participants, yielding a trajectory planner that enables implicit cooperation among traffic participants.

Various improvements to the base algorithm, such as parallelization and hyperparameter optimization, were developed to enhance its performance and generate better solutions in a shorter time. In addition, the planner was augmented with learned reward models based on expert trajectories, using Inverse Reinforcement Learning (IRL), enabling it to adapt to a desired human driving style for smooth integration into human-centered traffic.

The effectiveness of the proposed approach was demonstrated in 15 challenging multi-agent scenarios, showing significant improvements in the success rate compared to a baseline.

Preface

As a research scientist at the Karlsruhe Institute of Technology (KIT), my main area of focus was the challenging problem of cooperative trajectory planning. This thesis represents the culmination of my research efforts in this field, which were conducted as part of the priority program Cooperatively Interacting Automobiles of the DFG (German Research Foundation).

I am deeply grateful to the many individuals who have contributed significantly to this project's success.

I sincerely thank the research groups with which I was affiliated at KIT and the FZI Research Center for Information Technology. In addition, my gratitude extends to my professor, J. Marius Zöllner, who provided me with the opportunity to delve into this fascinating subject and offered support throughout my research journey with insightful discussions and valuable suggestions. Further, I thank Christoph Stiller, the spokesperson of the DFG priority program, for his willingness to serve as my co-examiner and Moritz Werling for his mentorship over the last decade, without whom I would not have started this Ph.D.

I also thank all my students I was fortunate enough to supervise, who have contributed immensely through in-depth discussions to shape many aspects of this project.

Finally, I acknowledge the invaluable support of my friends and family, who have always encouraged me to strive for growth and learning and accompanied me every step of this journey.

Contents

Acronyms

AD Automated Driving. iii, 4, 40, 121, 123

AV Automated Vehicle. iii, 1, 34, 35, 61, 84, 123

BO Bayesian Optimization. 80, 81, 85, 116–120, 122

DUCT Decoupled Upper Confidence Bound for Trees. iii, 49, 51, 53, 58, 60, 92, 121

fANOVA functional Analysis of Variance. 117

GCL Guided Cost Learning. 62, 69

GP Gaussian Process. 80

GS Grid Search. 78, 79, 85

HP Hyperparameter. 77–81, 85, 110, 116, 117, 119, 120, 122

HPC Hyperparameter Configuration. *see* HPC

HPO Hyperparameter Optimization. 36, 77–79, 81, 83, 85, 116, 117, 119, 120, 122

IDM Intelligent Driver Model. *see* IDM

IRL Inverse Reinforcement Learning. iii, 17, 18, 22, 62, 65, 66, 69, 70, 84, 105, 121, 141

JSON JavaScript Object Notation. 126

MCS Monte Carlo Search. 12, 13, 49, 52

MCTS Monte Carlo Tree Search. iii, 13–16, 25, 27, 28, 32, 33, 40, 44, 49–53, 55, 57, 59, 60, 62, 65, 66, 69, 72, 73, 75, 83, 84, 92, 94, 113, 116, 121, 122, 125

MDP Markov Decision Process. 7, 8, 10–14, 23, 32

MG Markov Game. 10, 11, 31–33, 36, 59

MMDP Multi-agent Markov Decision Process. iii, 10

MOBIL minimizing overall braking induced by lane change. *see* MOBIL

NE Nash Equilibrium. 11

NN Neural Network. 28, 66, 68, 69, 72, 84

POMDP Partially Observable Markov Decision Process. 23, 25, 26, 83

RF Random Forest. 81, 116, 117

RL Reinforcement Learning. 11, 12, 16, 26, 27, 32, 36, 62, 64–66, 69

RS Random Search. 79, 81, 85, 116, 118, 120

SEM standard error of the mean. 73

SMC sequential Monte Carlo. 25

StVO Road Traffic Regulations in Germany. 2

UCT Upper Confidence Bound for Trees. 14, 49, 52–54, 57, 59, 81, 117, 119

Glossary

action duration [s] duration of an action. 38, 81, 117, 120

action execution fraction [1] fraction of the action duration to be executed. 81, 117

action grouping semantic action classes to distinguish between different successor states. 94, 97–100, 117, 119, 121, 141

blind value heuristic for the exploration of the action space. 52–54, 94, 96, 98, 100, 121

cooperation factor [1] weight to incorporate rewards from other agents. 47

discount factor [1] weight to decay rewards further into the future. 7, 78, 81, 117, 127

HPC configuration instance of hyperparameters. 78–80, 85, 87, 116, 117, 120, 122

IDM single-lane driver model, used for traffic simulation (Treiber et al. 2000). 25–27

iteration one complete execution of the four main phases of the MCTS algorithm: selection, expansion, simulation, and update. 14, 15, 87, 89, 92, 93, 95–97, 99, 100, 106, 112, 114, 115, 119

MOBIL general lane-changing driver model MOBIL for traffic simulation (Kesting et al. 2007). 26, 27

progressive widening increases the number of available actions over time. 52, 53, 55, 60, 78, 81, 83, 98, 99, 111, 117, 119, 121

search depth [1] depth of the search tree. 81, 117, 120

similarity update uses the similarity between actions during the update. 55, 56, 78, 94, 96–98, 100, 111

state value [1] value of a state s. 9, 10, 12, 13, 28, 64

state-action value [1] value of an action a in a given state s. 9, 10, 12–16, 32, 51, 55, 58, 69, 74–77, 85, 89, 96, 97, 110, 111

success rate [1] fraction of evaluated runs without collisions or invalid states/actions, higher values indicate better performance of the algorithm. 87, 88, 93, 95–97, 99, 100, 106, 112, 114, 115, 119

visit count [1] number of times a state s has been visited. 14–16, 51, 52, 56, 58, 75, 76, 96

1 Introduction and Motivation

Despite the fast-evolving capabilities of Automated Vehicles (AVs), they still lack a critical component distinguishing them from human drivers in terms of behavior—the ability to cooperate implicitly. Unlike today's AVs, human drivers incorporate other drivers' actions and intentions into their decisions and can thus demand or offer cooperation even without explicit communication.

Although much research addresses cooperative driving, the focus has been on explicit cooperation, which requires explicit communication between vehicles or between vehicles and infrastructure (Englund et al. 2016, Düring and Pascheka 2014, Frese et al. 2007).

However, in the foreseeable future, neither all vehicles will have the necessary technical equipment to enable communication between vehicles and the infrastructure nor algorithms will be standardized to the extent that communicated environmental information and behavioral decisions are uniformly taken into account. Thus, AVs must be capable of cooperating with other vehicles even without communication.

For this reason, methods that do not explicitly rely on communication are required and thus applicable to a much larger number of traffic scenarios.

1.1 Research Questions

This thesis aims to provide answers to the following two questions:

1. How can cooperative driving without communication be modeled?

2. How can the modeled problem be solved effectively and efficiently?

Since the first question is concerned with the general approach to this problem, the latter builds on the first. The answers given to these two research questions are neither considered conclusive nor complete. However, they provide a solid base for further research on cooperative multi-agent decision-making in heterogeneous environments.

1.2 Motivation

"The FSD [Full Self Driving] improvement will come as a quantum leap, because it's a fundamental architectural rewrite, not an incremental tweak. I drive the bleeding edge alpha build in my car personally. Almost at zero interventions between home & work. Limited public release in 6 to 10 weeks." – Elon Musk, August 2020

While the anticipations concerning the next generation of automated driving software are high (see quote), it is unlikely that the remaining long tail of problems will be solved in the near future. Although the more significant part of the traffic in Germany conforms to rules (Road Traffic Regulations in Germany (StVO)) that define proper behavior on roads, occasionally, situations that require a high degree of interaction between the traffic participants arise, not covered by these rules. In these scenarios, traditional, noninteractive trajectory planning methods frequently fail to find feasible trajectories due to limited longitudinal or lateral clearance between vehicles, as they do not adequately model interactions between the traffic participants (Trautman and Krause 2010, Kurzer et al. 2018b,a, Bae et al. 2020, Saxena et al. 2020), cf. Fig. 1.1.

Cooperation can be an essential aspect of resolving these situations efficiently. In an idealized scenario, vehicles would use vehicle-to-vehicle communication to negotiate their actions (SAE J3216). However, not all traffic participants will be equipped with the required communication systems in the near term, communication signals might be disturbed, or the communicated negotiation

(a) Noninteractive Planning **(b)** Interactive Planning

Figure 1.1: Noninteractive and Interactive Planning: In a, the blue agent conducts traditional non-interactive planning, i.e., finding an optimal single-agent trajectory given the red agent's predicted trajectory (e.g., by assuming constant velocity), and therefore must slow down before the obstacles (i.e., parked vehicles). In b, the blue agent conducts interactive planning, i.e., finding an optimal multi-agent trajectory given that both vehicles have sufficient space to pass the obstacles simultaneously and therefore do not need to slow down.

even be maliciously attacked (Shladover 2021). Thus, implicit approaches (i.e., not relying on explicit communication) are required to ensure robust cooperative decision-making.

1.2.1 Aim and Scope

The purpose of a vehicle is to transfer its occupants from the start position to the desired goal position. It can be assumed that the vehicle's occupants aspire to reach the goal state as comfortably, quickly, and safely as possible. Given an empty road network and a start and goal position, choosing a route that optimizes the criteria above is straightforward. However, vehicles will likely encounter other road users with their own goals, navigating the same network.

Depending on the situation, interactions with these participants are necessary to avoid accidents and minimize time, discomfort, and cognitive strain while traversing the optimal route. This principle applies to all traffic participants. If every participant were to act solely in their interest, without considering the needs of others (a behavior known as "defecting"), it would be improbable to achieve the best possible outcome for everyone involved. Instead, traffic would likely cause significant distress for human drivers.

Cooperative behavior, where individuals work together and consider the needs of others, is the key to achieving a better outcome for all road users. Robert Axelrod's

research on the evolution of cooperation supports this idea. In his work, Axelrod demonstrated that cooperative behavior could be considered collectively stable, meaning that no alternative strategy can outperform or replace cooperation within a population of individuals who already act cooperatively (Axelrod and Hamilton 1981).

By fostering cooperation, traffic participants can work together to achieve a global optimum, ensuring a smoother and safer experience for everyone on the road.

Thus, this thesis aims to plan locally optimal cooperative trajectories for multiple non-holonomic agents in a structured environment, e.g., a multi-lane road with oncoming traffic.

The following assumptions about the upstream and downstream tasks are made. Firstly, the algorithm receives an exact and fully observable environment from the upstream perception module, see Fig. 1.2, as input. Secondly, providing drivable trajectories is sufficient for the downstream control module. While the task of decision-making is most commonly associated with planning, it is essential to note that cooperative decision-making requires planning and prediction to be conducted in parallel (Bahram et al. 2016b). Thus, this work is associated with both the planning and the prediction module, although cooperative trajectory prediction is more of a byproduct of cooperative trajectory planning.

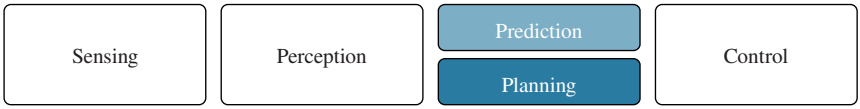


Figure 1.2: Modules of a Typical Automated Driving (AD) Framework

1.2.2 Structure

This thesis is structured into eight chapters, each addressing different aspects of the research conducted.

Chapter 1: Introduction - This chapter provides an overview of the research problem, its significance, and the research questions addressed in the thesis.

Chapter 2: Preliminaries - The fundamental concepts and theories that this thesis builds upon are introduced in this chapter, providing a required basic foundation for the subsequent material.

Chapter 3: State of the Art - A review of existing work in the field of cooperative driving is presented in this chapter, highlighting relevant literature and drawing connections to the research questions posed in this thesis.

Chapter 4: Concept Development - The primary research question, as outlined in Section 1.1, is addressed in this chapter. The basic concept and underlying methodology are introduced and discussed in detail.

Chapter 5: Concept Extensions - Building on the foundation established in Chapter Chapter 4, this chapter presents extensions to the initial concept, providing answers to the secondary research question described in Section 1.1.

Chapter 6: Experiments and Evaluation - The concept and its extensions are evaluated in various scenarios in this chapter, with results analyzed and discussed to determine their effectiveness and potential for improvement.

Chapter 7: Summary and Outlook - The final chapter synthesizes the research findings, discusses their implications, and provides conclusions drawn from the study. Additionally, it highlights potential avenues for future work in the area of cooperative driving.

Chapter A: Appendix - The appendix briefly overviews the developed concepts' source code and implementation details, ensuring transparency and reproducibility for future research.

2 Preliminaries

This chapter briefly introduces the required fundamentals, including the notation used throughout this thesis. As each subject encompasses decades of research, only topics essential to this thesis comprehension are presented. Literature references are given for a comprehensive introduction to each mentioned topic.

2.1 Markov Decision Process

The following section is based on the de facto standard by Sutton et al. (Sutton and Barto 2018). The Markov Decision Process (MDP) formalizes the fundamental problem of sequential decision-making.

An MDP is defined by a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}^a_{ss'}, \mathcal{R}^a_s \rangle$, with:

- $\mathcal{S}$ being the finite state space,
- $\mathcal{A}$ being the finite action space,
- $\mathcal{P}^a_{ss'} = \mathbb{P}[s'|a, s]$ being the model of the probability to transition to s' when taking action a in state s,
- $\mathcal{R}^a_s = \mathbb{E}[r|a, s]$ being the model of the expected reward received when taking action a in state s.

The reward is a scalar feedback signal, and the central part of each MDP, it describes the goodness of an action in a state. Most MDPs are extended by a discount factor $\gamma \in [0, 1)$, which decays rewards further into the future. This

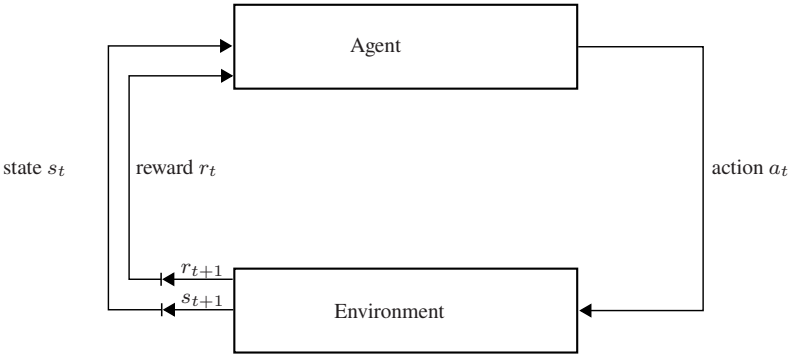


Figure 2.1: Visualization of the Agent Interaction in the Environment: The agent acts in the environment (MDP), transitions to a new state, and receives a reward.

circumvents infinite returns in cyclical MDPs and implies uncertainty surrounding the future (Silver 2015).

Figure 2.1 depicts the interaction of an agent in an MDP. The agent is the decision-making element, that chooses an action a_t at each time step t based on the observed state s_t. The environment transitions to a new state s_{t+1} and emits the reward r_{t+1}. It is the goal to maximize the accumulated discounted return. An MDP is considered to be solved if the optimal policy is discovered.

2.1.1 Trajectory

The trajectory is defined by a sequence of states and actions as a path through an MDP as

$$\tau = (s_0, a_0, s_1, a_1, \ldots, s_T). \tag{2.1}$$

2.1.2 Return

The return G of a trajectory τ equals its accumulated discounted reward at time step t, taking action a_t in state s_t (Sutton and Barto 2018), defined as

$$G(\tau) = \sum_{(s_t, a_t) \in \tau} \gamma^t \mathcal{R}_{s_t}^{a_t}. \tag{2.2}$$

2.1.3 Policy

The policy π is a probability distribution over actions $a \in \mathcal{A}(s)$, defined as

$$\pi = \pi(a \mid s). \tag{2.3}$$

The optimal policy $\pi^*(a \mid s)$[1] is a policy that yields the highest state value and state-action value of all policies.

$$\begin{aligned} V^*(s) &\geq V^\pi(s) \qquad \forall s \in \mathcal{S} \\ Q^*(s,a) &\geq Q^\pi(s,a) \quad \forall s \in \mathcal{S}, \forall a \in \mathcal{A}(s) \end{aligned} \tag{2.4}$$

2.1.4 State Value

The recursive form of the state value $V^\pi(s)$ with an arbitrary policy π, is defined as

$$V^\pi(s) = \sum_{a \in \mathcal{A}} \pi(a \mid s) \left(\mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a V^\pi(s') \right). \tag{2.5}$$

[1] To simplify notation the optimal policy $\pi^*(a \mid s)$ is abbreviated with $*$.

The recursive form of the state value $V^*(s)$ with the optimal policy, is defined as

$$V^*(s) = \max_{a \in \mathcal{A}} \left(\mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a V^*(s') \right). \tag{2.6}$$

2.1.5 State-Action Value

The recursive form of the state-action value $Q^\pi(s, a)$ with an arbitrary policy π, is defined as

$$Q^\pi(s, a) = \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a \sum_{a' \in \mathcal{A}} \pi(a' \mid s') Q^\pi(s', a'). \tag{2.7}$$

The recursive form of the state-action value $Q^*(s, a)$ with the optimal policy, is defined as

$$Q^*(s, a) = \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a \max_{a' \in \mathcal{A}} Q^*(s', a'). \tag{2.8}$$

2.2 Multi-agent Markov Decision Process

The extension of an MDP to a Markov Game (MG)/Multi-agent Markov Decision Process (MMDP) is required if sequential decision-making problems with multiple interacting agents are to be modeled (Littman 1994, Boutilier 1999, Zhang et al. 2021). With MMDPs usually referring to the fully cooperative setting (Boutilier 1999), with identical reward models and MGs denoting competitive and mixed settings (Zhang et al. 2021).

An MG is defined by a tuple $\langle \Upsilon, \mathcal{S}, \mathcal{A}, \mathcal{P}_{ss'}^a, \mathcal{R}_s^a \rangle$, with:

- Υ being the finite agent space of all agents, υ, indexed by $i \in 1, 2, \ldots n$
- $\mathcal{S}$ being the finite state space observed by all agents

- $\mathcal{A}$ being the finite joint action space; $\mathcal{A} = \prod_{\upsilon \in \Upsilon} \mathcal{A}^{\upsilon}$, $\mathcal{A}^{\upsilon}$ denoting the action space of agent υ
- $\mathcal{P}^{\boldsymbol{a}}_{ss'} = \mathbb{P}[s' | \boldsymbol{a}, s]$ being the model of the probability to transition to s' when taking joint action $\boldsymbol{a}$ in state s
- $\mathcal{R}^{\boldsymbol{a}}_{s} = \mathbb{E}[r | \boldsymbol{a}, s]$ being the model of the expected reward that is received when taking joint action $\boldsymbol{a}$ in state s

In an MG each agent independently selects its action a_i. The resulting joint action $\boldsymbol{a}$ then determines the state transition and the rewards of each agent. Thus each agent tries to maximize its return separately by searching for an optimal policy π^*_{υ} that is conditioned on the policies of all other agents.

The joint policy of an MG is defined as

$$\pi(\boldsymbol{a} \mid s) := \prod_{\upsilon \in \Upsilon} \pi_{\upsilon}\left(a_{\upsilon} \mid s\right). \tag{2.9}$$

The optimal joint policy $\pi^*(\boldsymbol{a} \mid s)$ is commonly one that results in a Nash Equilibrium (NE). A NE is defined as an optimal joint policy for each agent, and thus there is no benefit to deviate from it. That means there is no policy π_{υ} that would yield a higher expected return in response to the optimal policies $\pi^*_{\neg\upsilon}$ of all other agents (Zhang et al. 2021),

$$V_{\upsilon}^{\pi^*_{\upsilon}, \pi^*_{\neg\upsilon}}(s) \geq V_{\upsilon}^{\pi_{\upsilon}, \pi^*_{\neg\upsilon}}(s) \quad \forall \pi^*_{\upsilon}. \tag{2.10}$$

2.3 Reinforcement Learning

Reinforcement Learning (RL) is a subset of machine learning (Silver 2015). It can be used to find optimal policies (2.4) in an MDP. The learning is explicitly guided by the reward signal of the MDP, resulting from the interaction of the agent with its environment, cf. Fig. 2.1.

2.3.1 Planning and Searching

Planning and searching are two closely related yet different paradigms (Silver 2009). Both approaches produce an optimal action given a state. However, planning can be seen as global optimization, trying to determine an optimal policy (2.4).

While planning usually encompasses an extended period of computation based on a model to generate a new policy, searching can be seen as local optimization, determining an optimal action a^* from a specific state s. Searching merely computes a partial policy for the current state, making finding an optimal action significantly more efficient. In order to use searching for planning, the search algorithm is restarted for each trajectory step.

2.3.2 Model-Based Reinforcement Learning

While the definitions for model-based reinforcement learning differ, they always include a model, (Kaiser et al. 2020). Some see model-based RL as the process of learning a model of the MDP (i.e., the transition model) and using this model to derive an optimal policy. Others include cases where the model is already given and solely used to determine the optimal policy or action. In this work, the transition model and the reward model are known, and the goal is to find an optimal action a^* for any given state s of the environment.

2.3.2.1 Monte Carlo Search

Monte Carlo Search (MCS) is frequently used to generate a Monte Carlo estimate of the state value or the state-action value in MDPs with large branching factor, randomness or partial observability (Coulom 2007).

The mean of the returns over all trajectories $\mathcal{T}$ sampled from policy π, starting in state s is the Monte Carlo estimate of the state value,

$$\widehat{V}^{\pi}(s) = \frac{1}{|\mathcal{T}|} \sum_{\tau \in \mathcal{T} \sim \pi(s)} G(\tau), \tag{2.11}$$

and when considering action a the state-action value

$$\widehat{Q}^{\pi}(s,a) = \frac{1}{|\mathcal{T}|} \sum_{\tau \in \mathcal{T} \sim \pi(a|s)} G(\tau), \tag{2.12}$$

respectively (Sutton and Barto 2018).

Once an estimate for the state-action value has been determined, the optimal action is found by maximizing over it,

$$a^* = \arg\max_{a \in \mathcal{A}} \widehat{Q}^{\pi}(s,a). \tag{2.13}$$

While MCS generates estimates, it is inefficient, as it only conducts a single step of policy optimization (Brunskill 2019). That means previous trajectory returns do not influence the sampling process, potentially wasting many samples in unpromising regions of the search space.

2.3.2.2 Monte Carlo Tree Search

An exhaustive tree search can find the optimal trajectory through any MDP with a finite set of states and actions (Browne et al. 2012). However, as the action space grows, searching for the optimal trajectory through the entire tree quickly becomes intractable.

Tree Search combined with Monte Carlo sampling addresses this issue by approximating the optimal solution asymptotically through sampling. Monte Carlo Tree Search (MCTS) is a computationally efficient, highly selective best-first search, (Kocsis and Szepesvári 2006, Coulom 2007) that explores different trajectories

through the MDP, to discover the trajectory that maximizes the return G from the root state. In the past, MCTS has demonstrated its capabilities on highly challenging problems such as Go (Silver et al. 2016).

Given an initial root state of the MDP, MCTS approximates the state-action value in four sequential steps during each iteration until a terminal condition is met (e.g., until a time budget or computational budget is exceeded). Since MCTS is an anytime algorithm (Kocsis and Szepesvári 2006), it returns an estimate after the first iteration.

2.3.2.2.1 Selection The most popular form of MCTS uses an Upper Confidence Bound for Trees (UCT), to control the selection of successor states. The UCT value (Kocsis and Szepesvári 2006) for all explored actions from the current state is calculated during the selection phase, see (2.14), and the state action tuple with the maximum UCT value is selected. This process repeats until a state is selected that has not been fully explored (i.e., not all available actions in the state have been expanded), see Fig. 2.2.

Using UCT, MCTS solves the exploration-exploitation dilemma (Kocsis and Szepesvári 2006), being an upper confidence bound for the estimation of the true state-action value. The first term in (2.14) fosters exploitation of previously explored actions with high state-action values. The second term guarantees that all actions for a given state are being explored at least once, with $N(s)$ being the visit count for state s and $N(s,a)$ the number of times action a has been chosen in that state. To balance the exploration-exploitation trade-off, a constant factor c is used.

$$\mathrm{UCT}(s,a) = \widehat{Q}^{\pi}(s,a) + c\sqrt{\frac{2\log N(s)}{N(s,a)}} \tag{2.14}$$

2.3.2.2.2 Expansion Once the selection policy encounters a state with untried actions left, it expands that state by randomly sampling an action from a

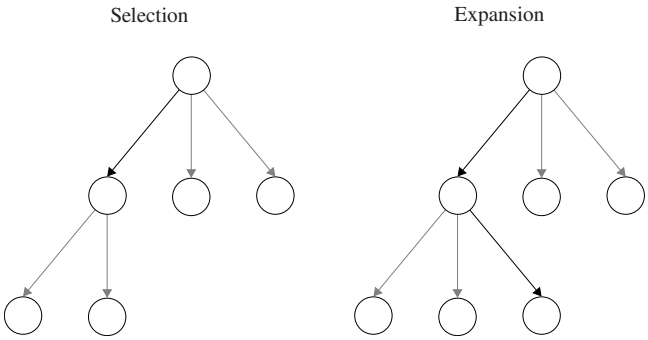


Figure 2.2: Selection and Expansion in MCTS: Circles denote states and edges denote actions.

uniform distribution over the action space, see (2.15), and executing the action reaching a successor state, see Fig. 2.2.

$$a \sim U[\min(\mathcal{A}), \max(\mathcal{A})] \tag{2.15}$$

2.3.2.2.3 Simulation After the expansion of an action completes, a simulation of subsequent random actions is conducted until a terminal condition is met (i.e., the planning horizon is reached or an action is sampled, resulting in a terminal state). This generates an estimate of the state-action value for the previous expansion, see Fig. 2.3.

2.3.2.2.4 Update Lastly, the return G of the trajectory τ generated by the iteration is backpropagated to all states along the trajectory, see Fig. 2.3, and the state-action values and visit counts for all actions of the trajectory are updated, see (2.16) and (2.17), respectively.

$$N(s,a) \leftarrow N(s,a) + 1 \tag{2.16}$$

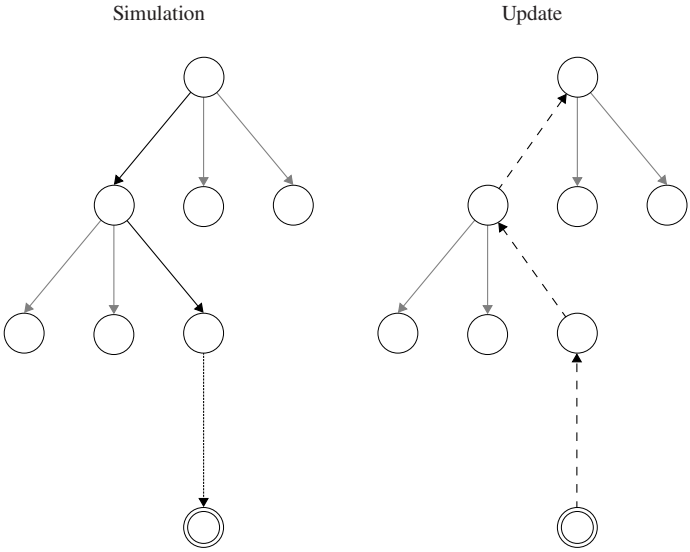


Figure 2.3: Simulation and Update in MCTS: Circles denote states and edges denote actions.

$$Q^{\pi}(s,a) \leftarrow Q^{\pi}(s,a) + \frac{1}{N(s,a)}(G(\tau) - Q^{\pi}(s,a)) \tag{2.17}$$

2.3.2.2.5 Final Selection Once a predefined computational budget is depleted, the best-performing action from the root state is selected. Best-performing can mean different things depending on the implementation, e.g., maximum state-action value or maximum visit count.

2.4 Inverse Reinforcement Learning

While the task in RL is the deduction of an optimal policy from interactions of an agent with the environment based on a reward model (Sutton and Barto 2018), see

(5.2), the opposite is the case for Inverse Reinforcement Learning (IRL) (Arora and Doshi 2021). Here, the task is to infer the underlying reward model that the optimal policy aims to maximize (Ng and Russell 2000).

Since the reward model is the most succinct and transferable description of an agent's behavior (Abbeel and Ng 2004), a close approximation of the underlying reward model will yield a behavior that is similar to the behavior that results from the optimal policy, i.e., the expert behavior. A reward model generalizes better than a policy, as it does not directly capture the agents' behavior, but its goals (Silver 2015).

Thus IRL is particularly effective in conducting behavior cloning or discovering the underlying motivations of an agent (Ng and Russell 2000).

IRL learns the parameters $\boldsymbol{\theta}$ of a parameterized reward model $\mathcal{R}_{\boldsymbol{\theta}}$ so that the expert policy π_{E} becomes the optimal policy given the reward model (Ng and Russell 2000).

Instead of requiring access to the expert policy π_{E} itself, it is sufficient to observe trajectories $\mathcal{T}_{\mathrm{E}}$ that originate from that policy in order to learn the parameters $\boldsymbol{\theta}$ (Abbeel and Ng 2004),

$$\tau_{\mathrm{E}} = (s_0, a_0, s_1, a_1, \ldots, s_T) \quad a_t \sim \pi_{\mathrm{E}}(a_t|s_t, \boldsymbol{\theta}). \tag{2.18}$$

For linear reward models, the reward model $\mathcal{R}_{\boldsymbol{\theta}}$ can be represented as a linear combination of the features $\phi(s, a)$ and the weights $\boldsymbol{\theta}$ (Abbeel and Ng 2004),

$$\mathcal{R}_{\boldsymbol{\theta}}(s_t, a_t) = \boldsymbol{\theta}^\top \phi(s_t, a_t). \tag{2.19}$$

The expected feature count μ is the expectation over the sum of the discounted features of a policy π (Abbeel and Ng 2004),

$$\mu(\pi) := \mathbb{E}\left[\sum_{t=0}^{T} \gamma^t \phi(s_t, a_t) \mid \pi, T, d_0\right]. \tag{2.20}$$

With d_0 being the start state distribution, T being the length of the trajectories, and π the policy from which the features originate.

2.4.1 Maximum Entropy Inverse Reinforcement Learning

Similarly to the policy π as a distribution over actions (2.3), a policy ρ as a distribution over trajectories can be defined,

$$\rho(\tau) = \rho(s_0, a_0, s_1, a_1, \ldots, s_T) = \prod_{t=0}^{T-1} \pi(a_t \mid s_t), \tag{2.21}$$

(assuming a constant start state s_0 and a deterministic transition model $\mathcal{P}^a_{ss'}$).

A prominent method for IRL is Maximum Entropy Inverse Reinforcement Learning (Ziebart et al. 2008), which assumes a probabilistic model for expert behavior. Using the definition of a policy over trajectories (2.21), maximum entropy IRL specifies the distribution over expert trajectories conditioned on the weights of the reward model

$$\rho_{\mathrm{E}}(\tau) = \frac{e^{G_{\boldsymbol{\theta}}(\tau)}}{Z_{\boldsymbol{\theta}}}. \tag{2.22}$$

This model implies that the probability of an expert trajectory increases exponentially with its return. With the numerator being the exponentiated return of a trajectory (2.2) and the denominator the partition function (2.23), the integral of the exponentiated return of all trajectories.

$$Z_{\boldsymbol{\theta}} = \int e^{G_{\boldsymbol{\theta}}(\tau)} \, \mathrm{d}\tau \tag{2.23}$$

The likelihood of the parameters $\boldsymbol{\theta}$ given the expert trajectories $\mathcal{T}_{\mathrm{E}}$ is defined with

$$L(\boldsymbol{\theta}|\mathcal{T}_{\mathrm{E}}) = \prod_{\tau \in \mathcal{T}_{\mathrm{E}}} \frac{e^{G_{\boldsymbol{\theta}}(\tau)}}{Z_{\boldsymbol{\theta}}}. \tag{2.24}$$

Applying the logarithm to (2.24), yields the Log-likelihood

$$l(\boldsymbol{\theta}|\mathcal{T}_{\mathrm{E}}) = \sum_{\tau \in \mathcal{T}_{\mathrm{E}}} \left(G_{\boldsymbol{\theta}}(\tau) - \log Z_{\boldsymbol{\theta}} \right), \tag{2.25}$$

which is proportional to

$$\frac{1}{|\mathcal{T}_{\mathrm{E}}|} \sum_{\tau \in \mathcal{T}_{\mathrm{E}}} G_{\boldsymbol{\theta}}(\tau) - \log Z_{\boldsymbol{\theta}}. \tag{2.26}$$

The maximization of the Log-likelihood[2] (2.26) through the parameters $\boldsymbol{\theta}$ will result in the parameters that best explain the expert trajectories,

$$\max_{\boldsymbol{\theta} \in \boldsymbol{\Theta}} \frac{1}{|\mathcal{T}_{\mathrm{E}}|} \sum_{\tau \in \mathcal{T}_{\mathrm{E}}} G_{\boldsymbol{\theta}}(\tau) - \log Z_{\boldsymbol{\theta}}. \tag{2.27}$$

Using the gradient of the Log-likelihood in a gradient ascent step, locally optimal parameters can be found,

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \nabla_{\boldsymbol{\theta}} l(\boldsymbol{\theta}). \tag{2.28}$$

[2] in the following, the Log-likelihood refers to the proportional Log-likelihood

Given the equalities $\frac{d}{dx} \ln f(x) = \frac{f'(x)}{f(x)}$[3] and $\frac{d}{dx} e^{f(x)} = f'(x) e^{f}$[4] the gradient of the Log-Likelihood (2.29) can be formulated as an expectation (2.30).

$$\begin{aligned}
\nabla_{\boldsymbol{\theta}} l(\boldsymbol{\theta}) &= \frac{1}{|\mathcal{T}_{\mathrm{E}}|} \sum_{\tau \in \mathcal{T}_{\mathrm{E}}} \nabla_{\boldsymbol{\theta}} G_{\boldsymbol{\theta}}(\tau) - \nabla_{\boldsymbol{\theta}} \log Z_{\boldsymbol{\theta}} \\
&= \frac{1}{|\mathcal{T}_{\mathrm{E}}|} \sum_{\tau \in \mathcal{T}_{\mathrm{E}}} \nabla_{\boldsymbol{\theta}} G_{\boldsymbol{\theta}}(\tau) - \frac{\nabla_{\boldsymbol{\theta}} Z_{\boldsymbol{\theta}}}{Z_{\boldsymbol{\theta}}} \\
&= \frac{1}{|\mathcal{T}_{\mathrm{E}}|} \sum_{\tau \in \mathcal{T}_{\mathrm{E}}} \nabla_{\boldsymbol{\theta}} G_{\boldsymbol{\theta}}(\tau) - \frac{\nabla_{\boldsymbol{\theta}} \int e^{G_{\boldsymbol{\theta}}(\tau)} \, \mathrm{d}\tau}{Z_{\boldsymbol{\theta}}} \\
&= \frac{1}{|\mathcal{T}_{\mathrm{E}}|} \sum_{\tau \in \mathcal{T}_{\mathrm{E}}} \nabla_{\boldsymbol{\theta}} G_{\boldsymbol{\theta}}(\tau) - \frac{\int e^{G_{\boldsymbol{\theta}}(\tau)}}{Z_{\boldsymbol{\theta}}} \nabla_{\boldsymbol{\theta}} G_{\boldsymbol{\theta}}(\tau) \, \mathrm{d}\tau \\
&= \frac{1}{|\mathcal{T}_{\mathrm{E}}|} \sum_{\tau \in \mathcal{T}_{\mathrm{E}}} \nabla_{\boldsymbol{\theta}} G_{\boldsymbol{\theta}}(\tau) - \int \rho_{\mathrm{E}}(\tau) \nabla_{\boldsymbol{\theta}} G_{\boldsymbol{\theta}}(\tau) \, \mathrm{d}\tau
\end{aligned} \tag{2.29}$$

$$\nabla_{\boldsymbol{\theta}} l(\boldsymbol{\theta}) = \frac{1}{|\mathcal{T}_{\mathrm{E}}|} \sum_{\tau \in \mathcal{T}_{\mathrm{E}}} \nabla_{\boldsymbol{\theta}} G_{\boldsymbol{\theta}}(\tau) - \mathbb{E}_{\tau \sim \rho_{\mathrm{E}}(\tau)} \left[\nabla_{\boldsymbol{\theta}} G_{\boldsymbol{\theta}}(\tau) \right] \tag{2.30}$$

Applying importance sampling (see Section 2.4.2) the expectation in (2.30) can be calculated using the policy $\rho_{\mathrm{S}}(\tau)$,

$$\begin{aligned}
\mathbb{E}_{\tau \sim \rho_{\mathrm{E}}(\tau)} \left[\nabla_{\boldsymbol{\theta}} G_{\boldsymbol{\theta}}(\tau) \right] &= \int \nabla_{\boldsymbol{\theta}} G_{\boldsymbol{\theta}}(\tau) \rho_{\mathrm{E}}(\tau) \, \mathrm{d}\tau \\
&= \int \nabla_{\boldsymbol{\theta}} G_{\boldsymbol{\theta}}(\tau) \frac{e^{G_{\boldsymbol{\theta}}(\tau)}}{Z_{\boldsymbol{\theta}}} \, \mathrm{d}\tau \\
&= \int \nabla_{\boldsymbol{\theta}} G_{\boldsymbol{\theta}}(\tau) \frac{e^{G_{\boldsymbol{\theta}}(\tau)}}{Z_{\boldsymbol{\theta}}} \frac{\rho_{\mathrm{S}}(\tau)}{\rho_{\mathrm{S}}(\tau)} \, \mathrm{d}\tau \\
&= \mathbb{E}_{\tau \sim \rho_{\mathrm{S}}(\tau)} \left[\frac{e^{G_{\boldsymbol{\theta}}(\tau)}}{\rho_{\mathrm{S}}(\tau) Z_{\boldsymbol{\theta}}} \nabla_{\boldsymbol{\theta}} G_{\boldsymbol{\theta}}(\tau) \right]
\end{aligned} \tag{2.31}$$

[3] Logarithmic derivative
[4] Exponential derivative

Further, the partition function can be approximated using importance sampling as well, see (2.32) and (2.33).

$$\begin{aligned} Z_{\boldsymbol{\theta}} &= \int e^{G_{\boldsymbol{\theta}}(\tau)} \, \mathrm{d}\tau \\ &= \int e^{G_{\boldsymbol{\theta}}(\tau)} \frac{\rho_{\mathrm{S}}(\tau)}{\rho_{\mathrm{S}}(\tau)} \, \mathrm{d}\tau \\ &= \mathbb{E}_{\tau \sim \rho_{\mathrm{S}}(\tau)} \left[\frac{e^{G_{\boldsymbol{\theta}}(\tau)}}{\rho_{\mathrm{S}}(\tau)} \right] \\ &\approx \frac{1}{|\mathcal{T}_{\mathrm{S}}|} \sum_{\tau \in \mathcal{T}_{\mathrm{S}}} \frac{e^{G_{\boldsymbol{\theta}}(\tau)}}{\rho_{\mathrm{S}}(\tau)} \end{aligned} \tag{2.32}$$

$$\widehat{Z}_{\boldsymbol{\theta}} := \frac{1}{|\mathcal{T}|} \sum_{\tau \in \mathcal{T}} \frac{e^{G_{\boldsymbol{\theta}}(\tau)}}{\rho(\tau)} \tag{2.33}$$

Substituting the expectation in (2.30) with (2.31) as well as (2.23) with (2.33), the final form of the gradient estimate can be obtained (2.35).

$$\begin{aligned} \nabla_{\boldsymbol{\theta}} l(\boldsymbol{\theta}) &= \frac{1}{|\mathcal{T}_{\mathrm{E}}|} \sum_{\tau \in \mathcal{T}_{\mathrm{E}}} \nabla_{\boldsymbol{\theta}} G_{\boldsymbol{\theta}}(\tau) - \mathbb{E}_{\tau \sim \rho_{\mathrm{E}}(\tau)} \left[\nabla_{\boldsymbol{\theta}} G_{\boldsymbol{\theta}}(\tau) \right] \\ &= \frac{1}{|\mathcal{T}_{\mathrm{E}}|} \sum_{\tau \in \mathcal{T}_{\mathrm{E}}} \nabla_{\boldsymbol{\theta}} G_{\boldsymbol{\theta}}(\tau) - \mathbb{E}_{\tau \sim \rho(\tau)} \left[\frac{1}{\rho_{\mathrm{S}}(\tau)} \frac{e^{G_{\boldsymbol{\theta}}(\tau)}}{Z_{\boldsymbol{\theta}}} \nabla_{\boldsymbol{\theta}} G_{\boldsymbol{\theta}}(\tau) \right] \\ &= \frac{1}{|\mathcal{T}_{\mathrm{E}}|} \sum_{\tau \in \mathcal{T}_{\mathrm{E}}} \nabla_{\boldsymbol{\theta}} G_{\boldsymbol{\theta}}(\tau) - \frac{1}{|\mathcal{T}_{\mathrm{S}}|} \sum_{\tau \in \mathcal{T}_{\mathrm{S}}} \frac{1}{\rho_{\mathrm{S}}(\tau)} \frac{e^{G_{\boldsymbol{\theta}}(\tau)}}{Z_{\boldsymbol{\theta}}} \nabla_{\boldsymbol{\theta}} G_{\boldsymbol{\theta}}(\tau) \\ &\approx \frac{1}{|\mathcal{T}_{\mathrm{E}}|} \sum_{\tau \in \mathcal{T}_{\mathrm{E}}} \nabla_{\boldsymbol{\theta}} G_{\boldsymbol{\theta}}(\tau) - \frac{1}{|\mathcal{T}_{\mathrm{S}}|} \sum_{\tau \in \mathcal{T}_{\mathrm{S}}} \frac{1}{\rho_{\mathrm{S}}(\tau)} \frac{e^{G_{\boldsymbol{\theta}}(\tau)}}{\widehat{Z}_{\boldsymbol{\theta}}} \nabla_{\boldsymbol{\theta}} G_{\boldsymbol{\theta}}(\tau) \end{aligned} \tag{2.34}$$

$$\widehat{\nabla_{\boldsymbol{\theta}} l(\boldsymbol{\theta})} = \frac{1}{|\mathcal{T}_{\mathrm{E}}|} \sum_{\tau \in \mathcal{T}_{\mathrm{E}}} \nabla_{\boldsymbol{\theta}} G_{\boldsymbol{\theta}}(\tau) - \frac{1}{|\mathcal{T}_{\mathrm{S}}|} \sum_{\tau \in \mathcal{T}_{\mathrm{S}}} \frac{1}{\rho_{\mathrm{S}}(\tau)} \frac{e^{G_{\boldsymbol{\theta}}(\tau)}}{\frac{1}{|\mathcal{T}_{\mathrm{S}}|} \sum_{\tau \in \mathcal{T}_{\mathrm{S}}} \frac{e^{G_{\boldsymbol{\theta}}(\tau)}}{\rho_{\mathrm{S}}(\tau)}} \nabla_{\boldsymbol{\theta}} G_{\boldsymbol{\theta}}(\tau) \tag{2.35}$$

2.4.2 Importance Sampling

Importance sampling facilitates the estimation of a random variable x that adheres to a distribution $p(x)$ by employing an alternative, more tractable proposal distribution $q(x)$ for sampling purposes (Owen 2013). The proposal distribution should encompass regions where the target distribution exhibits substantial density. This technique is particularly beneficial when obtaining direct samples from the target distribution is challenging or entails considerable computational costs.

In this work, it is used to approximate the partition function within maximum entropy IRL, see (2.32) and (2.33).

$$\begin{aligned}\mathbb{E}_{X\sim p(x)}[X] &= \int_{\mathcal{D}} f(x)p(x)\mathrm{d}x \\ &= \int_{\mathcal{D}} \frac{f(x)p(x)}{q(x)} q(x)\,\mathrm{d}x \\ &= \mathbb{E}_{X\sim q(x)}\left[X\frac{p(x)}{q(x)}\right]\end{aligned} \tag{2.36}$$

$$\widehat{\mathbb{E}}_{X\sim p(x)}[X] = \frac{1}{|\mathcal{X}|}\sum_{x\in\mathcal{X}} x \quad \mathcal{X}\sim p(x) \tag{2.37}$$

$$\widehat{\mathbb{E}}_{X\sim q(x)}[X] = \frac{1}{|\mathcal{X}|}\sum_{x\in\mathcal{X}} x\frac{p(x)}{q(x)} \quad \mathcal{X}\sim q(x) \tag{2.38}$$

3 State of the Art in Cooperative Trajectory Planning

This chapter provides an overview of the state-of-the-art cooperative trajectory planning methods, focusing on different approaches for cooperative driving. Specific topics and related work relevant to each chapter's content will be presented in the upcoming chapters. This overview emphasizes the need for further research to develop more comprehensive and effective solutions for interactive trajectory planning in automated driving.

Most methods in the field model cooperative trajectory planning model the problem as an MDP or a Partially Observable Markov Decision Process (POMDP). The reward models used in these methods may vary depending on the desired behavior of the system. These methods aim to find an optimal trajectory or policy by searching, planning, or learning.

Further, the following criteria have been established to compare cooperative planning methods. These criteria can be utilized to assess the methods' characteristics, applicability, and performance in different traffic scenarios. Various cooperative trajectory planning methods are summarized and compared based on these crieria in Table 3.1.

- Application: Describing the specific scenario for which the method is designed helps identify its suitability and potential limitations in different contexts.
- Method: Identifying the primary technique used provides insight into its underlying principles and theoretical foundations.

- Implicit: Determining whether the method implicitly models interactions helps assess its ability to capture complex interactions among traffic participants without explicit communication.
- Parallel Prediction/Planning: Concurrent prediction and planning are required for simultaneous decision-making and anticipation of other agents' actions.
- Explicit Interactive Planning: Explicitly modeling interactions are required to appropriately model the interdependency of dynamic traffic situations, allowing for safer and more cooperative behaviors.
- Partial Observability: Considering partial observability acknowledges that not all states are fully known, ensuring that the method responds appropriately even under uncertainty.
- Continuous Action Space (ego): A continuous ego vehicle action space allows arbitrary maneuvers, making it applicable to a broader range of scenarios.
- Continuous Action Space (other): Similar to the ego vehicle, a continuous action space for other traffic participants indicates the method's capacity to model complex behaviors and interactions.
- Identical Action Spaces: Understanding if all traffic participants have identical action spaces provides insight into the method's assumptions and simplifications regarding different vehicle types.
- Longitudinal Actions: Considering acceleration and deceleration is essential for understanding the method's ability to control vehicle speed and maintain safe distances.
- Lateral Actions: Considering steering enables the method to navigate complex traffic situations, enhancing its maneuvering capabilities.
- End-To-End: An end-to-end approach streamlines the overall planning process and potentially reduces errors introduced by intermediate steps.
- Continuous Cooperation: Modeling continuous cooperation is essential for evaluating the method's ability to adapt to dynamic traffic situations and collaborate with other traffic participants.
- # Steps: The number of steps in the method's planning horizon provides insight into its look-ahead capability and ability to plan for future events.

- # Agents: The number of agents considered in the method indicates its scalability and applicability to scenarios with varying levels of traffic density.
- Output: The method's output, such as acceleration or steering angle, determines its usability and compatibility with specific sensors or control systems.

3.1 Planning

MCTS has been proposed as a solution for cooperative trajectory planning in dense traffic by Isele (Isele 2019). This method models the interactions between traffic participants on a high level based on the probability of yielding to the ego vehicle's intention. When no merge occurs, the simulator within MCTS uses the IDM driver models. The model can be further parameterized by factors such as personality, gap size, and distance during a merge. This approach reduces complexity by breaking down the problem into smaller steps, i.e., gap selection, prediction, and planning. Sequentially solving subproblems significantly reduces the method's potential as the interactions are limited.

Tian et al. propose an interactive lane change method (Tian et al. 2021) that also utilizes MCTS. This method explicitly models the interactions between traffic participants, considering the rationality and intelligence of the human driver as latent state dynamics, which are modeled using mixed policies and iterative level-k (Stahl and Wilson 1994) reasoning. This approach allows learning the latent dynamics through observation during the planning phase, resulting in more robust actions.

Hubmann et al. propose an interactive intersection navigation method modeled as a POMDP (Hubmann et al. 2018a) and solved via the Adaptive Belief Tree algorithm (Seiler et al. 2015, Klimenko et al. 2014). In addition, future trajectories of other vehicles are partially observable and tracked using a sequential Monte Carlo (SMC) method.

Further, Hubmann et al. propose an interactive lane change method also modeled as a POMDP (Hubmann et al. 2018b) and solved via the Adaptive Belief Tree algorithm (Seiler et al. 2015, Klimenko et al. 2014). The longitudinal driving behavior of traffic participants is governed by the IDM. It is extended to behave either cooperatively or uncooperatively. The willingness to cooperate is estimated through observations.

3.2 Game Theory

Schwarting et al. introduce an interactive lane-changing and intersection navigation approach that leverages game theory to model interactions between traffic participants (Schwarting et al. 2019). This method considers each traffic participant with its whole action space. The cooperative aspect is included by estimating the agent's social value orientation. The problem is formulated as a nonlinear program and solved using a state-of-the-art nonlinear optimizer.

Bahram et al. also propose a game-theoretic approach for interactive lane changing based on a complete search (Bahram et al. 2016b). They model the problem as a sequential game to account for interactions between maneuvers performed by traffic participants. To reduce complexity, the number of actions for each agent is limited to 15, and the maneuver with the lowest risk is determined by calculating collision risks for all possible options.

3.3 Learning

Saxena et al. propose an interactive lane change method in dense traffic utilizing model-free RL (Saxena et al. 2020). In this work, the interactions between traffic participants are not explicitly modeled. Instead, they are learned implicitly from data during training generated in a simulator based on IDM and MOBIL driver models, similar to the Social Generative Adversarial Network in (Bae et al. 2020).

Bouton et al. present an intersection navigation method that can handle occlusions using model-free RL in conjunction with a probabilistic model checker to ensure safety (Bouton et al. 2019b). In this approach, the other two traffic participants (i.e., a car and a pedestrian) are modeled using rule-based models, and their interactions are learned from data during training. This method can be extended to handle additional participants by decomposing the scene, which, however, may lead to conservative behavior since worst-case assumptions are made.

In another study, Bouton et al. propose an interactive single-lane lane change method in dense traffic using model-free RL (Bouton et al. 2019a). In this work, the interactions between traffic participants are learned implicitly from data generated in a simulator based on IDM driver models and hence do not need to be explicitly modeled. The cooperativeness of other drivers is estimated using a recursive Bayesian filter. The results suggest that an MCTS approach with full observability performs the best.

3.4 Combined Learning and Planning

Bae et al. propose an interactive lane change method in dense traffic leveraging the predictions of a Social Generative Adversarial Network (SGAN) (Gupta et al. 2018) as a basis for planning controls using Model Predictive Control (MPC) (Bae et al. 2020). The SGAN is trained on simulations of dense traffic utilizing the IDM and MOBIL driver models. During the planning phase, the controller generates a set of trajectories with Monte Carlo rollouts to be evaluated in conjunction with the SGAN predictions. The trajectory with the lowest cumulative cost is selected, provided it satisfies the constraints. This approach is limited to lane changes and uses simple driver models for other traffic participants. Further, the planning horizon of 2 s reduces the interactions with other vehicles.

Hoel et al. presented an interactive lane change method that combines MCTS and RL (Hoel et al. 2020). The method considers other traffic participants modeled using the IDM and the MOBIL driver models. In this approach, the driver's

state is partially observable and estimated using a particle filter. Actions yielding collisions are excluded from the action space. In addition, the MCTS is guided by a Neural Network (NN) that generates state value targets, enhancing the sampling process's efficiency and accuracy.

3.5 Summary

Although there is a significant body of research on interactive trajectory planning for intersection and lane change scenarios, most approaches reduce the complexity of interactive trajectory planning through assumptions that violate completeness guarantees or limit the problem to lateral or longitudinal planning.

Only one approach (Schwarting et al. 2019) meets all of the following criteria. Firstly, it does not rely on communication between vehicles; secondly, prediction and planning are performed concurrently and interactively with a sufficiently large planning horizon; and finally, all interacting agents are modeled with their complete and continuous action spaces.

This highlights the need for further research to address existing methods' limitations and develop more comprehensive and effective solutions for interactive trajectory planning in automated driving.

Publication	Application	Method	Implicit	Parallel Prediction/Planning	Explicit Interactive Planning	Partial Observability	Continuous Action Space (ego)	Continuous Action Space (other)	Identical Action Spaces	Longitudinal Actions	Lateral Actions	End-To-End	Continuous Cooperation	# Steps	# Agents	Output
(Bae et al. 2020)	lane change	MPC/SL	+	+	-	+	+	+	-	+	+	+	-	1	∞	$\ddot{x}, \delta$
(Saxena et al. 2020)	lane change	RL	+	+	-	-	+	+	-	+	+	+	+	1	∞	$\dddot{x}, \dot{\delta}$
(Bouton et al. 2019b)	intersection	RL	+	+	-	+	-	-	-	+	-	+	-	1	≤ 10	$\ddot{x}$
(Bouton et al. 2019a)	lane change	RL	+	+	-	+	-	-	-	+	-	+	+	1	4	$\ddot{x}$
(Bouton et al. 2020)	lane change	RL	+	+	-	-	-	-	-	+	+	-	-	1	8	high level actions
(Isele 2019)	lane change	MCTS	+	-	+	+	-	-	-	+	+	-	+	2	2	τ to intention
(Kurzer et al. 2018a)	driving	MCTS	+	+	+	-	+	+	+	+	+	+	+	4	∞	τ
(Tian et al. 2021)	lane change	MCTS	+	+	+	+	-	-	+	+	+	-	-	8	2	control commands
(Hoel et al. 2020)	lane change	MCTS/RL	+	+	-	+	-	+	-	+	+	-	-	20	8	high level actions
(Hubmann et al. 2018b)	lane change	ABT	+	+	+	+	-	-	-	+	+	+	-	1	∞	$\ddot{x}_{\mathrm{lon}}, \dot{x}_{\mathrm{lat}}$
(Hubmann et al. 2018a)	intersection	ABT	+	+	+	+	+	+	-	+	-	+	-	1	∞	$\ddot{x}$
(Schwarting et al. 2019)	driving	NLP	+	+	+	-	+	+	+	+	+	+	+	20	∞	$\ddot{x}, \delta$
(Bahram et al. 2016a)	lane change	Search	+	+	+	-	-	-	+	+	+	+	-	1	∞	τ

Table 3.1: Overview of Different Interactive Trajectory Planning Algorithms (State of the Art)

4 Implicit Cooperative Decision-Making

The following chapter describes the central contribution of this work: the multi-agent trajectory planner. Contemporary methods often simplify interactive trajectory planning by making assumptions that compromise solution completeness or restrict the planning to the lateral or longitudinal dimension. In contrast, the proposed cooperative multi-agent trajectory planning algorithm considers all physically feasible actions for every traffic participant and synchronously performs prediction and planning. This allows for generating cooperative trajectories across a multi-step planning horizon, addressing many limitations of existing approaches.

In this chapter, some of the material presented has been previously published in the following papers Kurzer et al. 2018a and Kurzer et al. 2018b. Additionally, parts of the research described in this chapter are based on the work of the following supervised theses Engelhorn 2018 and Zhou 2018.

4.1 Problem Formulation

The problem of cooperative trajectory planning is formulated as an MG consisting of multiple non-communicating agents, see Section 2.2. This formulation captures the aspect of real-world scenarios where traffic participants (human or automated agents) often need to make decisions independently without direct communication with others. Each agent independently chooses an action at each time step,

unaware of the decisions made by others, which promotes decentralized decision-making that could foster more scalable and robust solutions.

In this cooperative multi-agent system, both the state transition and the reward are contingent on the collective actions of all agents. This interdependency prompts agents to align their actions with the common objective, potentially generating more efficient and safer trajectories that contribute to an overall optimized outcome.

In the context of an MG, every agent aspires to maximize its expected cumulative reward (2.2). Several ways to achieve this goal include learning-based methods, such as value-based RL. Here the agent aims to approximate the state-action value function for all states of the MDP to derive later a policy that maximizes the state-action value in each state, yielding the optimal policy $\pi^*(a \mid s)$ (5.2).

However, learning-based methods have their limitations. For example, a comprehensive global optimization that explores and evaluates all potential states in the MDP to formulate the optimal policy is computationally intensive and, thus, time-consuming. Furthermore, learning is highly dependent on the training data distribution and can struggle with "out-of-distribution" scenarios, i.e., situations not encountered or adequately represented during training. The optimal policy derived from learning-based methods may perform poorly when confronted with these novel situations, potentially leading to suboptimal or unsafe decisions.

On the other hand, planning-based methods, such as MCTS, are more adaptable to such out-of-distribution scenarios. Rather than relying on a learned policy, they compute actions for the current state of the environment, allowing for greater flexibility when facing previously unseen situations. In addition, planning-based methods search for the optimal action for a specific state instead of focusing on the global optimization of the optimal policy for all possible states, reframing the problem as local optimization. This change in perspective is especially beneficial for automated driving, where immediate, context-specific decisions are typically more valuable than optimizing long-term strategies.

Therefore, the problem is formulated as the search for the optimal action given a state in an MG. For this search, MCTS, a planning-based method that balances exploration and exploitation through a combination of tree search and random sampling, is employed.

4.2 Requirements and Assumptions

In developing any system, it is critical to identify the fundamental requirements and assumptions underpinning that system's design and operation. In the context of the multi-agent cooperative trajectory planning problem, these requirements and assumptions shape the formulation of the algorithm and influence its performance, robustness, and adaptability.

The requirements are the criteria or capabilities the system must satisfy or possess to address the problem effectively. In this case, they include the algorithm's rationality, implicit communication, intuitiveness, fastness, and scalability. These requirements are dictated by the inherent characteristics and constraints of real-world driving scenarios, where multiple traffic participants interact dynamically and make independent decisions.

Conversely, the assumptions are the premises or conditions taken to be valid for system design. They form the basis upon which the requirements are formulated and the problem is solved. Here, the assumptions include the rational behavior of traffic participants, the absence of explicit communication, the co-existence of automated and non-automated traffic participants, and the computational complexity associated with the number of traffic participants.

Based on the assumptions, it is important to recall the research questions to formulate the algorithm's requirements.

1. How can cooperative driving without communication be modeled?
2. How can this problem be solved efficiently?

4.2.1 Rational

This requirement assumes that humans and AVs will act rationally within the driving environment. In this context, being rational means that individuals will select actions that maximize their return and that the reward model is aligned with traffic regulations and a safe driving style. Requiring rational behavior is crucial for the system's proper functioning because it allows the algorithm to make accurate predictions about the actions of other drivers. If other participants were to behave irrationally, predicting their actions would become significantly more complex, leading to possible miscalculations and unsafe driving situations.

4.2.2 Implicit

This requirement stems from the assumption that no explicit form of communication, such as vehicle-to-vehicle (V2V) or vehicle-to-infrastructure (V2I) communication, is necessary to exchange intentions or information. This reflects the current state of traffic systems, where most decisions are made based on observed behavior rather than explicit communication. Operating without explicit communication is vital as it allows the algorithm to apply to a broader range of scenarios and environments. Further, reducing dependence on possibly unreliable communication channels enhances the system's resilience.

4.2.3 Intuitive

The requirement for intuitive behavior arises from the assumption that AVs and human traffic participants will co-exist. Thus, the algorithm should generate predictable and intuitive trajectories for human drivers, even if those drivers may not fully understand the mechanics of autonomous decision-making. This is important for overall traffic safety and efficiency since unpredictable maneuvers by autonomous vehicles could confuse or surprise human drivers, potentially leading to accidents or traffic disruptions.

4.2.4 Scalable and Fast

The scalable and fast requirement assumes that the number of traffic participants can be large and variable and that the environment can change quickly. As such, the algorithm's computational complexity must be manageable even as the number of agents increases or the state space becomes more complex. This requirement is essential to ensure the algorithm can make timely decisions in real-world driving scenarios since delays in decision-making can lead to inefficiencies or even safety risks.

4.2.5 Further Research Questions

Based on the previous requirements, new research questions arise.

1. How can a continuous action space be incorporated so that trajectory planning in constricted scenarios is possible?
   The rationality requirement necessitates optimal decision-making at each step. Incorporating a continuous action space broadens the possible decisions, enabling precise maneuvers crucial for maximizing rewards, especially in constricted scenarios requiring safe and efficient navigation.

2. How can a reward function be designed so that the resulting behavior represents human driving?
   This question arises from the requirement for intuitive behavior. To ensure that AVs behave in a predictable manner for humans, the algorithm needs to learn behaviors that closely resemble human driving. Designing an appropriate reward function is a crucial step toward achieving this.

3. How can parallel computation be leveraged to accelerate convergence speed?
   This question concerns the requirement for fast, scalable computation. By leveraging parallel computation, the algorithm could be sped up, enabling

the algorithm to handle larger state spaces or more complex scenarios more efficiently.

4. How can Hyperparameter Optimization (HPO) be employed to improve the solution quality?
   This question stems from the requirement for high-quality solutions. HPO techniques can be used to automatically tune the parameters of the algorithm, potentially leading to better performance and more efficient solutions.

4.3 Multi-Agent Driving Simulator

RL-based methods require access to an environment such as a simulator or the real world (see Fig. 2.1) to simulate an agent's experience. However, since RL is based on the trial-and-error principle, ensuring safety during learning in the real world is hard. Thus, the following section describes the multi-agent driving simulator developed for and used throughout this work that models agents' interactions in the MG.

4.3.1 State Space

The state of a traffic participant is defined with its

- longitudinal position x_{lon},
- lateral position x_{lat},
- longitudinal velocity $\dot{x}_{\mathrm{lon}}$,
- lateral velocity $\dot{x}_{\mathrm{lat}}$,
- longitudinal acceleration $\ddot{x}_{\mathrm{lon}}$,
- lateral acceleration $\ddot{x}_{\mathrm{lat}}$, and
- heading ϕ.

Further, each traffic participant is a vehicle denoted by its

- width w,
- length l,
- wheelbase l_{wb},
- maximum acceleration $a_{\max}$, and
- maximum steering angle $\delta_{\max}$.

An agent v controls a traffic participant. Generally, this could be any participant, such as a car, truck and trailer, motorcycle, bicycle, or human. If desired, these traffic participants can easily be integrated once an appropriate model is defined. This thesis uses the kinematics of a single-track model (Section 4.3.4.1); thus, only car-like vehicles are considered.

4.3.2 Action Space

The action space of an agent is two-dimensional. The two dimensions are the longitudinal velocity change $\Delta\dot{x}_{\mathrm{lon}}$ and the lateral change in position Δx_{lat}. The tuple describes the desired state change over the action duration $\Delta T = t_1 - t_0$. Based on the current state and the chosen action, a jerk-optimal trajectory with continuous velocity and curvature is calculated using quintic polynomials (Takahashi et al. 1989). A jerk-optimal trajectory ensures the desired state change is achieved while optimizing comfort, safety, and energy efficiency. Two trajectories are calculated, one for the longitudinal and one for the lateral direction, respectively, defined with the coefficients $\boldsymbol{a}$ as

$$\begin{aligned} x(t) &= a_0 + a_1 t + a_2 t^2 + a_3 t^3 + a_4 t^4 + a_5 t^5 \\ \dot{x}(t) &= a_1 + 2a_2 t + 3a_3 t^2 + 4a_4 t^3 + 5a_5 t^4 \\ \ddot{x}(t) &= 2a_2 + 6a_3 t + 12a_4 t^2 + 20a_5 t^3 \end{aligned} \tag{4.1}$$

,as well as in matrix form

$$\boldsymbol{Ma} = \begin{bmatrix} 1 & t_0 & t_0^2 & t_0^3 & t_0^4 & t_0^5 \\ 0 & 1 & 2t_0 & 3t_0^2 & 4t_0^3 & 5t_0^4 \\ 0 & 0 & 2 & 6t_0 & 12t_0^2 & 20t_0^3 \\ 1 & t_1 & t_1^2 & t_1^3 & t_1^4 & t_1^5 \\ 0 & 1 & 2t_1 & 3t_1^2 & 4t_1^3 & 5t_1^4 \\ 0 & 0 & 2 & 6t_1 & 12t_1^2 & 20t_1^3 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \end{bmatrix}. \tag{4.2}$$

With each polynomial requiring six coefficients, six constraints need to be defined. The constraints are denoted by $\boldsymbol{c}$, defined as

$$\boldsymbol{c} = \begin{bmatrix} x(t_0) \\ \dot{x}(t_0) \\ \ddot{x}(t_0) \\ x(t_1) \\ \dot{x}(t_1) \\ \ddot{x}(t_1) \end{bmatrix}. \tag{4.3}$$

The current state of the vehicle determines the start constraints. The end constraints for $\dot{x}_{\mathrm{lon}}$ and x_{lat} are based on the selected action. The end position is given by the mean of the start and end velocity and the action duration ΔT. Further, the acceleration in the longitudinal and lateral directions and the velocity in the lateral

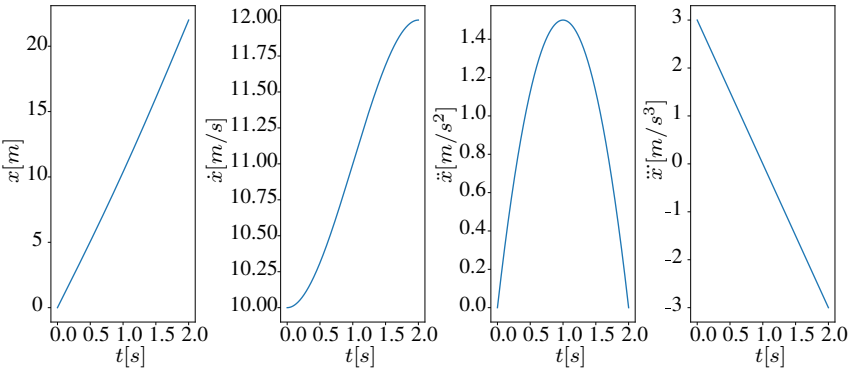


Figure 4.1: Trajectory of a Quintic Polynomial: The resulting position, velocity, acceleration, and jerk profile (from left to right) for the constraints $\boldsymbol{c} = [x(t_0) = 0, \dot{x}(t_0) = 10, \ddot{x}(t_0) = 0, x(t_1) = 22, \dot{x}(t_1) = 12, \ddot{x}(t_1) = 0]\,, \Delta T = 2$.

direction are set to zero, see (4.4), which significantly reduces the dimensionality of the action space, making the search more computationally manageable.

$$\begin{aligned} x_{\text{lon}}(t_1) &= x_{\text{lon}}(t_0) + \frac{\dot{x}_{\text{lon}}(t_0) + \dot{x}_{\text{lon}}(t_1)}{2}\Delta T \\ \dot{x}_{\text{lon}}(t_1) &= \dot{x}_{\text{lon}}(t_0) + \Delta\dot{x}_{\text{lon}} \\ \ddot{x}_{\text{lon}}(t_1) &= 0 \\ x_{\text{lat}}(t_1) &= x_{\text{lat}}(t_1) + \Delta x_{\text{lat}} \\ \dot{x}_{\text{lat}}(t_1) &= 0 \\ \ddot{x}_{\text{lat}}(t_1) &= 0 \end{aligned} \tag{4.4}$$

Using the constraints, (4.1) can be solved for its coefficients $\boldsymbol{a}$, see (4.5) (assuming $\boldsymbol{M}$ is invertible). An exemplary trajectory is depicted in Fig. 4.1.

$$\begin{aligned} \boldsymbol{M}\boldsymbol{a} &= \boldsymbol{c} \\ \boldsymbol{M}^{-1}\boldsymbol{M}\boldsymbol{a} &= \boldsymbol{M}^{-1}\boldsymbol{c} \\ \boldsymbol{a} &= \boldsymbol{M}^{-1}\boldsymbol{c} \end{aligned} \tag{4.5}$$

Instead of conducting the planning using Cartesian coordinates, trajectories are generated using the Frenet frame, a dynamic reference frame that aligns with the centerline of the road (Werling et al. 2010). This transformation allows the separation of longitudinal and lateral trajectory planning.

4.3.3 Semantic Action Classes

Sample-based planning methods, such as MCTS, often face challenges due to large search spaces. To address this issue, strategies that reduce or structure the search space can substantially enhance performance. In this work's context of AD, incorporating domain knowledge by subdividing the action space into semantic action classes offers a promising approach (Kurzer et al. 2018a).

Semantic action classes systematically categorize actions into meaningful groups based on their impact on the vehicle's state. These classes are action-state-dependent areas within the action space of an agent and are subdivided based on the following nine semantic definitions, as illustrated in Fig. 4.2 and Fig. 4.3 (Kurzer et al. 2018a):

- Maintain (0): small effect on the longitudinal velocity as well as the lateral position; lane does not change
- Accelerate (+): large positive effect on the longitudinal velocity and a small effect on the lateral position; lane does not change
- Decelerate (−): large negative effect on the longitudinal velocity and a small effect on the lateral position; lane does not change
- Accelerated Lane Change Left (L+): large positive effect on the longitudinal velocity and a large positive effect on the lateral position; lane change to the left
- Accelerated Lane Change Right (R+): large positive effect on the longitudinal velocity and a large negative effect on the lateral position; lane change to the right

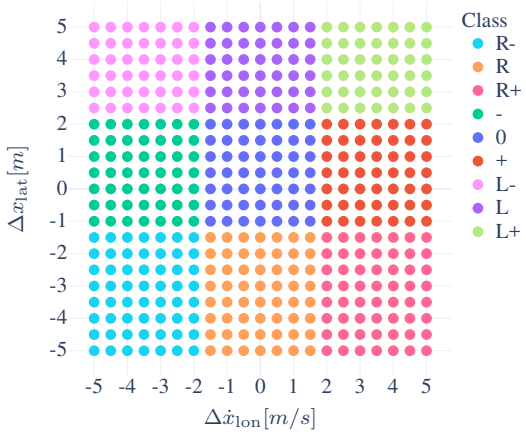


Figure 4.2: Semantic Action Classes in the Action Space: The boundaries of the action classes are action-state-dependent. In the lateral direction, they represent the relative position of the vehicle to its current lane; in the longitudinal direction, the acceleration of the action. The vehicle is driving in the middle lane of a three-lane road, with a slight offset to the right; hence the semantic action classes are not symmetrical.

- Decelerate Lane Change Left (L−): large negative effect on the longitudinal velocity and a large positive effect on the lateral position; lane change to the left
- Decelerate Lane Change Right (R−): large negative effect on the longitudinal velocity and a large negative effect on the lateral position; lane change to the right
- Lane Change Left (L): small effect on the longitudinal velocity and a large positive effect on the lateral position; lane change to the left
- Lane Change Right (R): small effect on the longitudinal velocity and a large negative effect on the lateral position; lane change to the right

This structured exploration enables a more targeted search and has the potential to improve overall performance. Furthermore, semantic action classes condense the search space by focusing on specific, meaningful action groups, allowing the algorithm to allocate computational resources more efficiently.

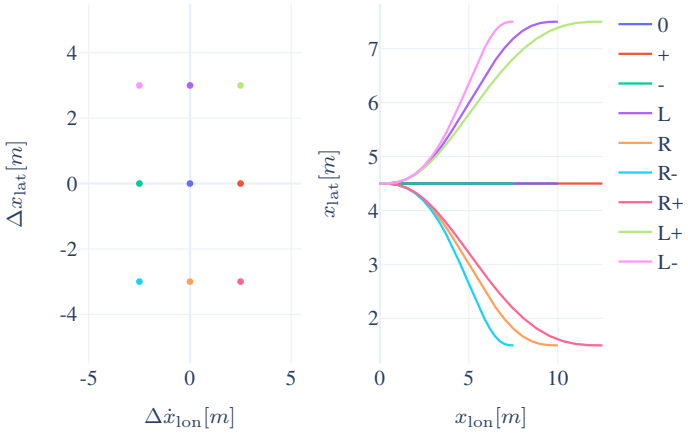


Figure 4.3: Relationship between Action Space and Trajectories: Nine different actions are sampled from the action space (left), resulting in nine different trajectories (right).

Additionally, semantic action classes enhance the interpretability of the resulting trajectories by relating selected actions to high-level maneuver classes, facilitating debugging, analysis, and communication with human operators. Lastly, by first sampling from the centers of the semantic action classes, the algorithm can prioritize meaningful actions, resulting in generally safer and more desirable trajectories.

4.3.4 Transition Function

Since this work does not focus on trajectory planning close to physical limits (e.g., required by evasive maneuvers), trajectories are evaluated using a single track model (Schramm et al. 2018) as it has been shown to perform sufficiently well for trajectory planning tasks (Kong et al. 2015). The execution of trajectories derived from the selected action is deterministic.

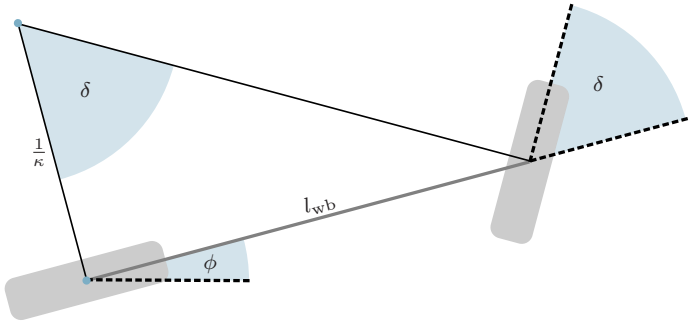


Figure 4.4: Kinematic of a Single Track Model

4.3.4.1 Vehicle Model

A single-track model is a simplified vehicle model of a front-wheel steered vehicle that models the respective tire pairs on the front and rear axles as a single wheel (Schramm et al. 2018), see Fig. 4.4.

4.3.4.2 Physical Constraints

To ensure that a chosen trajectory is drivable for a single-track model, the differential and kinematic constraints, i.e., the maximal acceleration and the minimum curve radius must be accounted for (LaValle 2006).

Based on the polynomials that describe the trajectories in longitudinal and lateral directions, the heading

$$\phi = \arctan\left(\frac{\dot{x}_{\text{lat}}}{\dot{x}_{\text{lon}}}\right), \tag{4.6}$$

curvature

$$\kappa = \frac{\dot{x}_{\text{lon}}\ddot{x}_{\text{lat}} - \dot{x}_{\text{lat}}\ddot{x}_{\text{lon}}}{\left(\dot{x}_{\text{lon}}^2 + \dot{x}_{\text{lat}}^2\right)^{\frac{3}{2}}}, \tag{4.7}$$

steering angle

$$\delta = \arctan\left(l_{\text{wb}}\kappa\right), \tag{4.8}$$

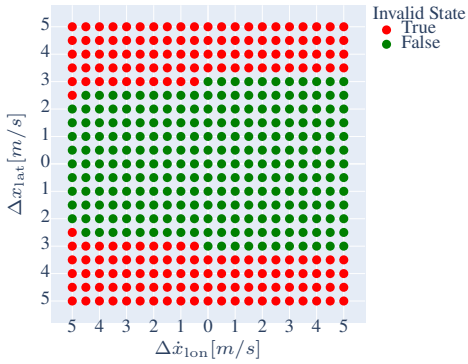


Figure 4.5: Invalid States: Invalid successor states resulting from different actions

acceleration

$$\ddot{x} = \sqrt{\ddot{x}_{\text{lon}}^2 + \ddot{x}_{\text{lat}}^2}, \tag{4.9}$$

and velocity

$$\dot{x} = \sqrt{\dot{x}_{\text{lon}}^2 + \dot{x}_{\text{lat}}^2} \tag{4.10}$$

are calculated.

An initially sampled action might fail to satisfy certain constraints or criteria in sampled-based trajectory planning. During the simulation and expansion phase of MCTS, candidate trajectories are sampled from the action space. Some of these sampled trajectories may violate constraints, such as physical limits (e.g., maximum steering angle or acceleration) or safety conditions (e.g., staying on the road and avoiding collisions with static obstacles, see Fig. 4.5), similar to Hoel et al. 2020. In such cases, resampling aims to generate new candidate actions that comply with these constraints. By incorporating resampling into MCTS, the search process becomes more effective in identifying valid and feasible trajectories, improving the overall performance of the planning process.

The physical constraints are considered in the validation reward of the reward function, see Section 4.3.5.3.

4.3.5 Reward Function

The model of the expected reward is the basis for the agent's behavior. It considers the state s and the action a of an agent, $\mathcal{R}_s^a$. As the transition model in this context is deterministic, the reward function can be simplified by eliminating the expectation term, resulting in the following:

$$r = r_s + r_a + r_{\text{validation}}. \tag{4.11}$$

The importance of each of the terms mentioned below is adjusted with a corresponding weight.

4.3.5.1 State Reward

The state reward r_s is based on the divergence of the current and the desired state. The desired state is defined by a longitudinal velocity v_{des} and a lane index i_{des}. A separate module is responsible for estimating these desired values for all agents. In the context of this work, it is assumed that the desired longitudinal velocity and lane index (i.e., the lane the agent is driving in, where the rightmost lane is indexed as 0) are equal to their respective values before any interaction occurs in each scenario.

Additionally, the agent is encouraged to drive close to the center of a lane by rewarding deviations from the center less.

4.3.5.2 Action Reward

Agents select actions to reduce the deviation from the desired state. While effectiveness in minimizing the deviation is necessary, the most effective action might yield undesirable accelerations. Thus, efficiency (i.e., balancing action and state rewards) is considered by penalizing all actions, and the action reward r_a is hence negative, i.e., a cost, see (4.14). Currently, r_a considers longitudinal

(4.12) and lateral acceleration (4.13) as well as lane changes, (4.14), with w being the weights and Δl the number of lane changes an action results in. If desired, the state and action reward can easily be extended to capture additional safety, efficiency, and comfort-related aspects of the generated trajectories.

$$C_{\ddot{x}} = w_{\ddot{x}} \int_{t_0}^{t_1} (\ddot{x}(t))^2 dt \tag{4.12}$$

$$C_{\ddot{y}} = w_{\ddot{y}} \int_{t_0}^{t_1} (\ddot{y}(t))^2 dt \tag{4.13}$$

$$C_l = w_l \Delta l \tag{4.14}$$

$$r_a = C_{\ddot{x}} + C_{\ddot{y}} + C_l \tag{4.15}$$

4.3.5.3 Validation Reward

The last term is the action validation reward, see (4.16). It evaluates whether a state and action are valid, i.e., being inside the drivable environment and adhering to the physical constraints, and whether a state action combination is collision-free.

$$\begin{aligned} r_{\text{validation}} &= w_{\text{invalid state}} \mathbb{1}_{\text{invalid state}} \\ &+ w_{\text{invalid action}} \mathbb{1}_{\text{invalid action}} \\ &+ w_{\text{collision}} \mathbb{1}_{\text{collision}} \end{aligned} \tag{4.16}$$

The symbol $\mathbb{1}_{\text{condition}}$ represents an indicator function, where the subscript denotes the evaluated condition. The function takes the value of 1 when the specified condition is true and 0 otherwise, serving as a concise notation to express conditional relationships in equations.

4.3.5.4 Cooperative Reward

To achieve cooperative behavior, a cooperative reward r^i_{coop} is defined. The cooperative reward of an agent i is the sum of its rewards, see (4.11), as well as the sum of all other rewards of all other agents multiplied by a cooperation factor λ^i, see (4.17), ((Lenz et al. 2016, Kurzer et al. 2018a)). The cooperation factor determines the agent's willingness to cooperate with other agents, with $\lambda^i = 0$ being purely interactive and $\lambda^i = 1$ being fully cooperative.

$$r^i_{\text{coop}} = r + \lambda^i \sum_{j=0, j\neq i}^{n} r^j \tag{4.17}$$

The cooperation factor can be used to represent different driver types. For example, an offensive driver weighs his own goals more than the goals of other road users. The joint reward function (4.17) is agent-individual and does not represent a global cost function. Therefore, the cooperative rewards of the individual agents cannot be compared since they have different values depending on the respective cooperation factor.

4.3.6 Collision Detection

Appropriate collision detection along the planned trajectory is a critical part of any trajectory planner. Since collision detection is a major bottleneck of trajectory planning methods, it must be conducted efficiently (Ziegler and Stiller 2010).

Further, in the case of a search-based approach, efficiency is critical since each trajectory combination has to be checked for collisions. Hence, a decomposition of rectangles into circles is employed (Ziegler and Stiller 2010). A hierarchical rectangle decomposition is implemented to reduce the computational cost further (Ericson 2004), see Fig. 4.6.

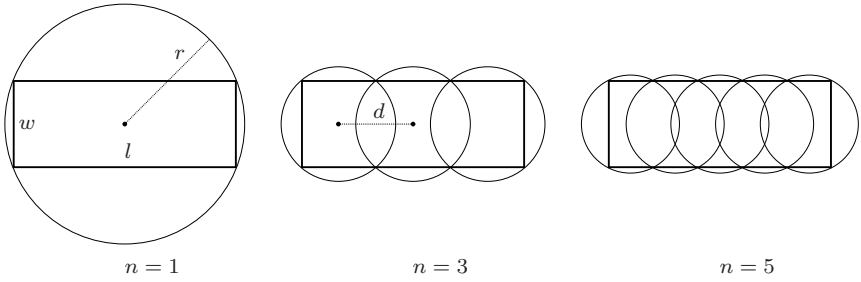


Figure 4.6: Hierarchical Rectangle Decomposition: Rectangles are decomposed using a hierarchy of circles. Initially, a single circle approximates the rectangle. In case of a collision, the approximation of the rectangle is improved by increasing the number of circles n.

At first, a single circle approximates the rectangle. However, if the circle collides with another object, the approximation is improved by increasing the number of circles to approximate the rectangle. This reduces the amount of over-approximation while also reducing computational costs, where a coarse collision detection suffices.

When approximating a rectangle, the goodness of the approximation varies depending on the number of circles n used. While the approximation in the lateral direction improves with an increasing number of circles, it decreases in the longitudinal direction. Hence, a hierarchical rectangle decomposition achieves higher accuracy and a computational benefit.

The radius for the decomposition of a rectangle with length l and width w using n circles is

$$r = \sqrt{\frac{l^2}{4n^2} + \frac{w^2}{4}}. \tag{4.18}$$

The distance between the circles is

$$d = 2\sqrt{r^2 - \frac{w^2}{4}}. \tag{4.19}$$

The collision of the circles to approximate an object with is used as a proxy for a collision. Two circles collide as soon as the distance between their centers d is less than the sum of their radii,

$$\mathbb{1}_{\text{collision}} = d < r_0 + r_1. \tag{4.20}$$

4.4 Decentralized Continuous MCTS

An exemplary application of MCTS to a traffic scenario requiring cooperation is shown in Fig. 4.7. However, specific extensions to it are required to solve this and many other cooperative trajectory planning problems with MCTS.

The original MCTS algorithm developed by (Kocsis and Szepesvári 2006) uses UCT, designed for sequential decision-making games with a finite set of states and actions. However, if traffic participants interact without communication, the actions of other traffic participants are not known until they are observed. Thus the basic MCTS used in turn-based games is not applicable and needs to be extended to simultaneous move games. In addition, trajectory planning in a continuous state and action space requires alterations to the standard selection procedure of actions. This is because UCT would degenerate MCTS to MCS, as each action in each state needs to be selected at least once, see (2.14). Lastly, the output of MCTS is an action a given a state s. In order to plan continuous trajectories, e.g., for driving, it is thus required to compute actions repeatedly for successor states.

4.4.1 Decoupled UCT

To address the problem of simultaneous decision-making Decoupled Upper Confidence Bound for Trees (DUCT) is employed (Tak et al. 2014). While the decoupled version of UCT does not guarantee to converge to an optimal policy (Schaeffer et al. 2009), it has shown to perform best when compared to other variants (Tak et al. 2014).

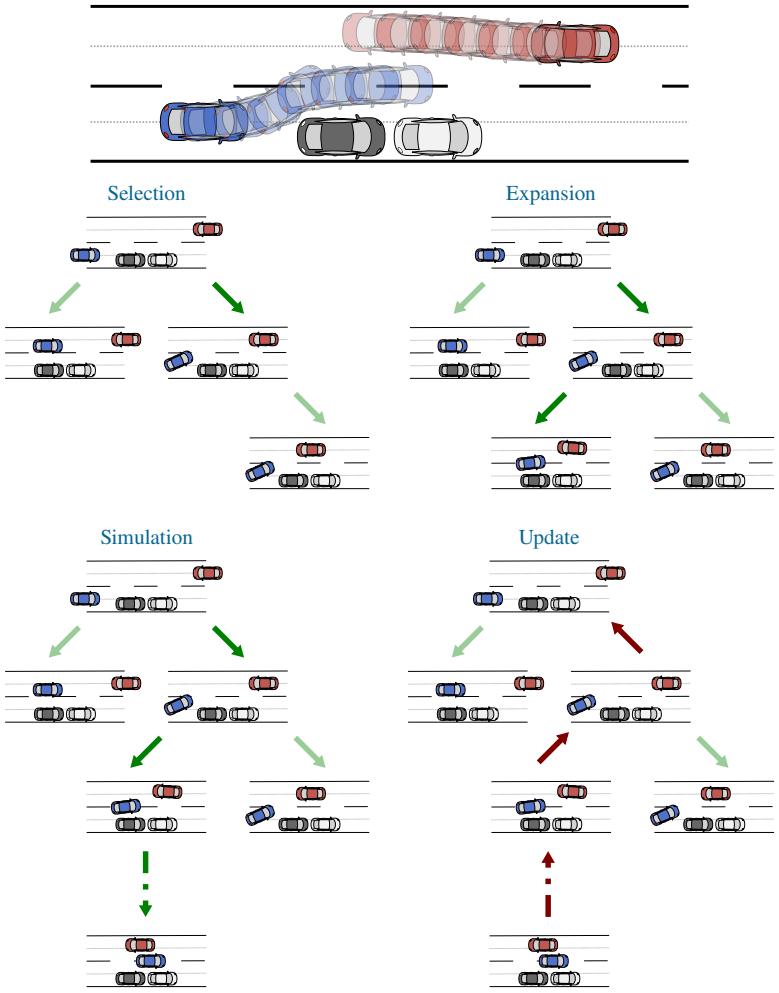


Figure 4.7: Phases of MCTS for a Scenario with a Narrow Passage: The gray vehicles are parked, and the red and the blue vehicle can pass the narrowing at the same time if the red vehicle deviates from its optimal trajectory. During the selection, MCTS traverses the tree by selecting auspicious future states until a state is encountered that has untried actions left. After the expansion of the state, a simulation of subsequent actions is run until the planning horizon is reached. Next, the result is backpropagated to all states along the selected path. Ultimately this process converges to the optimal policy.

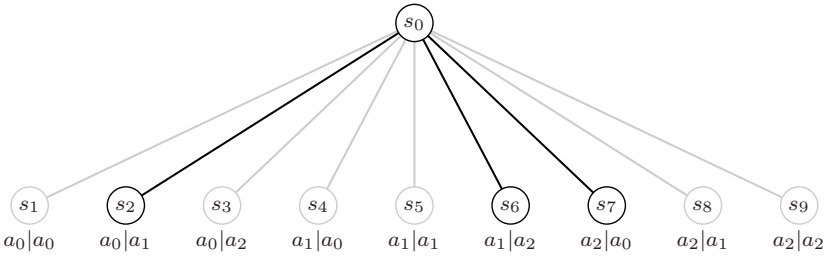


Figure 4.8: Action Combinations in DUCT: The resulting successor states are based on an identical action space $[a_0, a_1, a_2] \in \mathcal{A}(s_0)$ for two agents, with $\cdot|\cdot$ denoting the joint action $\boldsymbol{a}$ that led to a state.

The complexity of simultaneous decision-making results from incomplete information regarding the decision of other agents. Thus, the state-action value for an action a_i from agent i can only be approximated by averaging over all possible actions of the other agents. Based on the description of MCTS in Section 2.3.2.2, DUCT hence tracks the state-action value and visit count on a per agent basis, and the dependency between different agents is not considered when calculating the DUCT value,

$$\mathrm{DUCT}(s, a_i) = \widehat{Q}^{\pi}(s, a_i) + c\sqrt{\frac{2\log N(s)}{N(s, a_i)}} \quad a_i \in \mathcal{A}^i. \tag{4.21}$$

Each agent selects the action that maximizes its DUCT value during the selection step. The resulting joint action $\boldsymbol{a}$ leads to the successor node if it exists or expands a new node.

For two agents with identical action spaces of size, three Fig. 4.8 depicts all possible successor states. Since DUCT tracks the state-action value and visit count separately for each agent, s_0 is considered fully expanded once each agent has executed each of its available actions at least once (s_2, s_6, s_7), rather than all possible combinations resulting from the joint action space. Hence, an action can only be selected again if all actions of an agent have been selected at least once.

Only by randomly selecting a joint action $\boldsymbol{a}$ that has not previously been selected the remaining combinations $(s_1, s_3, s_4, s_5, s_8, s_9)$ are added to the search tree.

Similarly, the final selection is conducted independently of other agents.

4.4.2 Progressive Widening

The use of UCT, see (2.14), in MCTS requires the exploration of all possible actions from a given state (Kocsis and Szepesvári 2006). Actions of the successor states are only explored (Kocsis and Szepesvári 2006, Browne et al. 2012) once all actions of the predecessor have been explored. Thus, if the action space is continuous, the application of UCT within MCTS degenerates to MCS.

Progressive widening, sometimes also called progressive unpruning, aims to address the issue of large action spaces (Coulom 2007, Chaslot et al. 2008a), with additional work considering infinitely many actions within UCT (Wang et al. 2008).

Progressive widening gradually expands the existing action space of a node by adding additional actions. The number of actions for a state follows a sublinear function of the visit count, see (4.22), with c and $\alpha \in (0, 1)$ being determined empirically. The expansion of the action space can be random or follow a heuristic.

$$|\mathcal{A}(s)| = \lfloor c \cdot N(s)^{\alpha} \rfloor \tag{4.22}$$

The simplest way to add new actions is to select a random action from the theoretical action space. Another option is to use a heuristic, such as blind value, see Section 4.4.4.1.

The application of progressive widening is limited to an empirically determined search depth within the tree since the visit count for individual actions decreases with increasing search depth in this work. Hence the limitation ensures that the available actions at greater depths are visited sufficiently.

4.4.3 Selection Strategies

As mentioned previously, to account for the interaction and uncertainty (i.e., simultaneous action selection) between multiple agents DUCT is employed. In DUCT, separate UCT values are calculated for each agent, considering their rewards and visit counts.

4.4.4 Expansion Strategies

The expansion strategy is either random or guided. The random expansion strategy draws samples uniformly from the entire action space. In order to be able to use a continuous action space, progressive widening is employed to decide whether the action space is expanded. The guided expansion strategy uses a heuristic to find a promising node for expansion. One such heuristic is blind value.

4.4.4.1 Blind Value

Whenever a node is to be expanded, MCTS needs to add an action to the action space of the node. The standard strategy for discrete and small continuous action spaces is to employ uniform sampling over the entire action space (Browne et al. 2012, Couëtoux et al. 2012). However, as the action space grows, the sampling will be less likely to be in promising regions.

A heuristic that aims to guide the sampling process so that promising regions are more likely to be sampled is called blind value (Couëtoux et al. 2012). Blind value uses the previously explored actions of a node to guide the next expansion. The blind value first focuses on regions away from previously explored actions and then shifts towards regions with many high-valued actions.

The blind value for an action a_i of a set of randomly sampled actions $\mathcal{A}_{\text{rnd}}$ is calculated using the set of explored actions $\mathcal{A}_{\text{exp}}$ as well as an adaptation

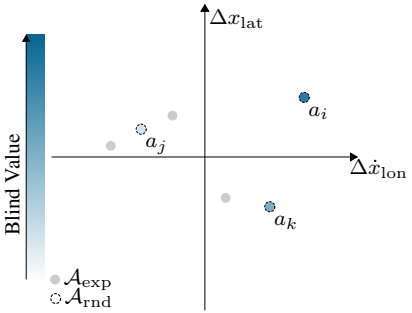


Figure 4.9: Blind value of Actions in Continuous Space: Assuming that the previously explored actions have identical UCT values, $a_i \in \mathcal{A}_{\text{rnd}}$ has the highest blind value, since its distance to other actions is largest, cf. (4.23) and (4.24).

coefficient ρ, see (4.23) and (4.25), respectively, with σ denoting the standard deviation.

$$\text{BV}(a_i, \mathcal{A}_{\text{exp}}, \rho) = \min_{a_j \in \mathcal{A}_{\text{exp}}} \text{UCT}(s, a_j) + \rho d(a_i, a_j) \tag{4.23}$$

$$d(a_i, a_j) = \sqrt{\left(a_i^{\Delta \dot{x}_{\text{lon}}} - a_j^{\Delta \dot{x}_{\text{lon}}}\right)^2 + \left(a_i^{\Delta x_{\text{lat}}} - a_j^{\Delta x_{\text{lat}}}\right)^2} \tag{4.24}$$

$$\rho(\mathcal{A}_{\text{exp}}, \mathcal{A}_{\text{rnd}}) = \frac{\sigma\left(\{\text{UCT}\left(s, a_j\right) \mid \forall a_j \in \mathcal{A}_{\text{exp}}\}\right)}{\sigma\left(\{d\left(0, a_i\right) \mid \forall a_i \in \mathcal{A}_{\text{rnd}}\}\right)} \tag{4.25}$$

The action with the highest blind value is finally selected, defined as

$$a^* = \underset{a_i \in \mathcal{A}_{\text{rnd}}}{\arg\max}\ \text{BV}(a_i, \mathcal{A}_{\text{exp}}, \rho). \tag{4.26}$$

4.4.5 Simulation Strategies

There are two types of simulation strategies, random and semantic. The random simulation strategy draws samples uniformly from the entire action space. The semantic simulation strategy draws the first samples from the centers of the semantic action classes, cf. Section 4.3.3, and continues to sample uniformly within the semantic action classes. Sampling from the centers is advantageous because the resulting lateral position will be centered on neighboring lanes if a lane change is selected.

4.4.6 Update Strategies

The update strategy is either standard or employs a similarity update. The standard update strategy updates only traversed nodes with the return, cf. Section 2.3.2.2.4. The similarity update strategy aims to aggregate the knowledge of similar actions.

4.4.6.1 Similarity Update

Progressive widening is an effective strategy to address a continuous action space during selection. However, as the action space of a node grows, it becomes less likely that an action is selected repeatedly, which is required to accurately estimate the state-action value. Due to the continuous nature of the action space, actions with a high degree of similarity might be sampled, cf. Fig. 4.10. Similar actions likely yield similar trajectories and returns; however, they are treated as unrelated actions in MCTS.

To share information between actions, the similarity of two actions $K(a_i, a_j) \in (0, 1]$ in the action space can be determined by a distance measure based on a radial basis function (kernel), see (4.27). Lower values for $\gamma \in \mathbb{R}^+$ increase the

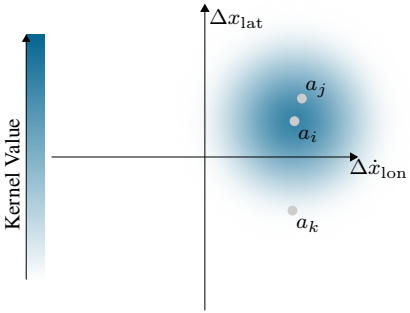


Figure 4.10: Similarity update Action a_j Given Action a_i in Continuous Space: In order to share information between similar actions, action a_j is updated with statistics gathered from action a_i weighted by the kernel value of $K(a_i, a_j)$, (4.28).

influence of other actions, and higher values decrease it (Kurzer et al. 2018a), with the value being determined empirically.

$$K(a_i, a_j) = \exp\left(-\gamma \left\|a_i - a_j\right\|^2\right) \tag{4.27}$$

During the update, the similarity update calculates the pairwise distance between the taken action a_i from a state and all other previously explored actions a_j of that state. The visit count and the return of the taken action are weighted by this distance to update the previously explored actions,

$$\begin{aligned} N(s, a_j) &\leftarrow N(s, a_j) + K(a_i, a_j), \\ Q(s, a_j) &\leftarrow Q(s, a_j) + \frac{K(a_i, a_j)}{N(s, a_j)}(G(\tau) - Q(s, a_j)). \end{aligned} \tag{4.28}$$

4.4.7 Final Selection Strategies

While UCT defines a clear selection criterion within the search tree of MCTS, different strategies for the final selection exist. The two most common strategies choose either the child node with the highest visit count,

$$\mathrm{MaxVisitCount}(s) = \underset{a \in \mathcal{A}(s)}{\arg\max}\, , N(s,a). \tag{4.29}$$

or the child node with the highest average reward (Browne et al. 2012, Chaslot et al. 2008a),

$$\mathrm{MaxActionValue}(s) = \underset{a \in \mathcal{A}(s)}{\arg\max}\, \widehat{Q}^{\pi}(s,a). \tag{4.30}$$

The $\mathrm{MaxVisitCount}$ strategy aims to choose the most promising and reliable action, and the $\mathrm{MaxActionValue}$ strategy aims to choose the action with the highest expected reward.

While it has been empirically shown that both strategies often perform similarly (Chaslot et al. 2008a), this work's experiments discovered that $\mathrm{MaxActionValue}$ outperformed $\mathrm{MaxVisitCount}$.

4.4.8 Action Grouping

The following section introduces the concept of action grouping. It is closely related to existing move grouping approaches (Saito et al. 2007, Childs et al. 2008, Bouzy 2006). The grouping of similar actions is intended to reduce the computational complexity on the one hand and increase the algorithm's robustness on the other hand.

In the context of this work, an action group m is a subset of the action space $\mathcal{A}$, which contains actions that cause similar changes in the state of the agent.

Formally, an action group m is a subset of the action space, where each action in that subset is mapped to m by the function f that maps actions to action groups:

$$m \subseteq \mathcal{A}, \quad \forall a \in m : f(a) = m, \quad f(a) : \mathcal{A} \to \mathcal{M}. \tag{4.31}$$

This mapping function f partitions the action space $\mathcal{A}$ into distinct subsets $m \in \mathcal{M}$, where each subset m represents a distinct action group. Each agent groups the actions available to it into different action-state-dependent action groups. When a new action is sampled, it is assigned to the corresponding action group.

To give action groups a semantic meaning, the mapping function f can be influenced by the features of the actions or their related states. This work uses the definition of semantic action classes (see Section 4.3.3) for action grouping based on domain knowledge. The result is a unique mapping from the action space to the action group space, with no overlap between different action groups.

During the update phase, the groups' statistics are calculated based on the statistics of their members. In the selection phase, the group statistics are first used to select the best action group according to DUCT before an action is determined within the group, effectively focusing on relevant areas of the action space.

Action groups carry statistics analogous to actions. These are the total visit count of the action group $N(s, m)$ and the estimate of the action groups' state-action value $\hat{Q}(s, m)$.

The visit count of an action group is merely the visit count of the actions in that group,

$$N(s, m) = \sum_{\forall a \in m} N(s, a). \tag{4.32}$$

Similarily, the state-action value is the mean of the state-action values over that group,

$$\hat{Q}(s, m) = \frac{1}{|m|} \sum_{\forall a \in m} Q(s, a). \tag{4.33}$$

The following changes occur when action grouping is applied to MCTS. During selection, UCT is first used to determine the best action group,

$$\mathrm{UCT}(s,m) = \widehat{Q}(s,m) + c\sqrt{\frac{2\log N(s)}{N(s,m)}} \quad m \in \mathcal{M}. \tag{4.34}$$

Then the best action is selected within this group using (2.14).

Analogously the final selection first selects the best action group and later determines the best action in that group, according to the specific criteria, see Section 4.4.7.

4.5 Summary

This chapter presents a cooperative multi-agent trajectory planning algorithm that considers all possible actions of traffic participants while prioritizing physical feasibility. The algorithm operates within the MG framework where each agent makes independent but simultaneous decisions. While learning-based methods have limitations such as computational intensity and struggle with novel scenarios, a planning-based method, MCTS, is utilized for its adaptability, local optimization ability, and efficiency in real-time, context-specific decision-making.

The requirements and assumptions for the multi-agent cooperative trajectory planning algorithm include rationality, implicit communication, intuitiveness, and scalability, mirroring the realities of dynamic real-world driving scenarios. Assumptions like rational traffic behavior, the absence of explicit communication, a mix of automated and non-automated traffic, and computational complexity are outlined in the chapter. Primary research questions focus on modeling cooperative driving, efficient problem-solving, incorporating a continuous action space, designing a reward function that mirrors human driving, leveraging parallel computation for faster convergence, and using Hyperparameter Optimization (HPO) for better solution quality.

Further, a multi-agent driving simulator is developed for reinforcement learning (RL)-based methods. The simulator uses a single-track model focusing on car-like vehicles. It calculates jerk-optimal trajectories using quintic polynomials, optimizing comfort, safety, and energy efficiency. Also, the simulator implements an efficient collision detection mechanism and designs a reward model for the agent's behavior.

Lastly, the chapter proposes a modified MCTS version tailored to cooperative trajectory planning, referred to as decentralized continuous MCTS. This method addresses limitations regarding simultaneous move games and continuous action spaces. It utilizes DUCT and progressive widening for simultaneous decision-making and planning in continuous spaces, respectively. Lastly, various methods are introduced, including action grouping, which combines similar actions to reduce computational complexity and increase robustness.

5 Intuitive, Scalable, and Fast Decision-Making

While the proposed concept in Chapter 4 is generally capable of solving a diverse set of scenarios, new questions arise: What is a suitable parameterization of the reward model? Can the planning algorithm be accelerated through parallelization? How can the parameters for the cooperative trajectory planning algorithm be automatically tuned?

In this chapter, some of the material presented has been previously published in the following papers Kurzer et al. 2020b and Kurzer et al. 2022. Additionally, parts of the research described in this chapter are based on the work of the following supervised theses Hörtnagl 2020, Bitzer 2020 and Reboud 2020.

5.1 Inverse Reinforcement Learning

Predictable trajectories for AVs are crucial to avoid confusing or surprising human drivers and prevent accidents or traffic disruptions. In line with the requirement for intuitive behavior arising from the co-existence of AVs and human traffic participants, this section aims to answer how a reward function can be learned from expert demonstrations so that the behavior sampled from the optimal policy based on this reward function resembles the expert demonstrations.

5.1.1 Introduction

RL based approaches frequently make use of manually specified reward models (Wolf et al. 2018, Kurzer et al. 2021). In environments where systems need to interact with humans, their decisions must be comprehensible and predictable (Carroll et al. 2019). As the complexity of the reward model rises, the manual parametrization of the same to generate the desired behavior becomes quickly infeasible.

In driving, numerous features impact the reward of any given trajectory as quantifiable inputs to a reward model. A feature can represent any aspect of a vehicle's state or environment, such as vehicle jerk, acceleration, other vehicles or obstacles, and adherence to traffic rules. The reward model steers the trajectory selection process by attributing higher rewards to trajectories resulting in favorable outcomes while imposing penalties for undesirable ones.

While tuning the weighting of features to create the desired behavior in a diverse set of scenarios is tedious and error-prone, IRL has proven to be able to recover the underlying reward model from recorded trajectories that demonstrate expert behavior in areas such as robotics and automated driving (Ng and Russell 2000, Abbeel and Ng 2004, Ziebart et al. 2008, Kuderer et al. 2015, Wulfmeier et al. 2016b).

This work builds on an existing cooperative trajectory planning algorithm (Kurzer et al. 2018a) to generate expert trajectories. Its contribution is twofold. The first is a system that conducts Guided Cost Learning (GCL), a sampling-based Maximum Entropy IRL method with MCTS to efficiently solve the forward RL problem in a cooperative multi-agent setting. The second is an evaluation that compares a linear and nonlinear reward model to a manually designed one. It is shown that the performance of the learned models is similar to or better than the manually tuned baseline. An overview of the system is depicted in Fig. 5.1.

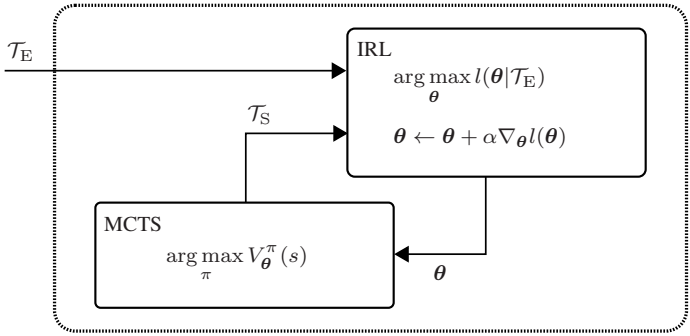


Figure 5.1: Overview of the System: At first, an initial set of expert trajectories $\mathcal{T}_\mathrm{E}$ is generated. Then the cooperative trajectory planning algorithm computes a set of sample trajectories $\mathcal{T}_\mathrm{S}$ using the randomly initialized reward model. Next, using the $\mathcal{T}_\mathrm{E}$ and $\mathcal{T}_\mathrm{S}$, the likelihood of the parameters $\boldsymbol{\theta}$ given the expert trajectories is increased using gradient ascent. Finally, the process repeats with the cooperative trajectory planning algorithm, sampling new trajectory samples until convergence.

5.1.2 Related Work

Early work in IRL performed feature matching rather than estimating the true underlying reward function (Abbeel and Ng 2004) to learn driving styles in a discrete driving simulator. More recently, driving styles are learned using continuous trajectories and action spaces (Kuderer et al. 2015), including additional features that impact driver preference (Naumann et al. 2020).

Wulfmeier et al. demonstrate the effectiveness of learning nonlinear reward models building on Maximum Entropy IRL (Ziebart et al. 2008) using Deep Neural Networks (Wulfmeier et al. 2016a), which they extended to learning cost maps for path planning from raw sensor measurements (Wulfmeier et al. 2016b).

Further improvements in the approximation of the partition function and the efficiency of IRL in combination with RL have been proposed by Guided Cost Learning (Finn et al. 2016). By adapting the IRL procedure, the method yields both a cost function and policy given expert demonstrations using sampling-based methods. In addition, even more, efficient sampling-based methods have been proposed (Wu et al. 2020).

In contrast, the following work learns a linear and nonlinear reward model integrated into a cooperative multi-agent trajectory planning algorithm using a continuous action and state space.

5.1.3 Problem Statement

Using the definitions of a policy (2.3) and a trajectory (2.1), a policy over trajectories can be defined assuming a deterministic start state distribution and transition model. The return of a trajectory τ equals its accumulated discounted reward at time step t, taking action a_t in state s_t (Sutton and Barto 2018), see (2.2). The value function of a policy π for an MDP with a reward model parameterized by $\boldsymbol{\theta}$ is the expectation of the return of trajectories sampled from that policy,

$$V_{\boldsymbol{\theta}}^{\pi}(s) = \mathbb{E}_{\tau\sim\rho}\left[G_{\boldsymbol{\theta}}(\tau)\right]. \tag{5.1}$$

While the forward RL problem is solved by finding the optimal policy

$$\pi^*(a \mid s) = \arg\max_{\pi} V_{\boldsymbol{\theta}}^{\pi}(s), \tag{5.2}$$

the inverse RL problem is solved by finding the parameters $\boldsymbol{\theta}$ so that,

$$V_{\boldsymbol{\theta}}^{\pi_{\mathrm{E}}}(s) \geq V_{\boldsymbol{\theta}}^{\pi}(s) \quad \forall \pi \in \Pi, \tag{5.3}$$

with π_{E} being the expert policy as part of the policy space Π.

This work aims to learn the parameters $\boldsymbol{\theta}$ of a reward model for cooperative trajectory planning so that the optimal trajectories of the planning algorithm are similar to the demonstrated expert trajectories, i.e., that the expert policy yields the highest state value of all policies given the parametrization of the reward model (5.3).

5.1.4 Approach

IRL is used to learn a reward model from expert demonstrations so that the behavior sampled from the optimal policy based on this reward model resembles the expert demonstrations. Concisely, this work makes use of a cooperative trajectory planning algorithm based on MCTS (Kurzer et al. 2018a) and Maximum Entropy IRL (Ziebart et al. 2008), yielding a system that is similar to Guided Cost Learning (Finn et al. 2016).

The MCTS is used to solve the (forward) RL problem, i.e., finding the optimal policy/action given a reward model and generating near-optimal trajectory samples $\mathcal{T}_\mathrm{S}$ for that policy. Using these trajectories in combination with the expert trajectories $\mathcal{T}_\mathrm{E}$ Maximum Entropy IRL is used to conduct a gradient ascent step increasing the likelihood of the parameters $\boldsymbol{\theta}$ given the expert trajectories, (2.27), (2.28), see Fig. 5.1.

While the trajectory planning algorithm explicitly encodes interaction between agents, the IRL procedure treats the resulting trajectories as if they would stem from a single agent MDP, with other agents being part of the environment. On the one hand, this has the advantage that the resulting reward model is more robust towards changes in the number of agents; on the other hand, it has the drawback of a non-stationary environment, as changes in the reward model (after each gradient ascent step of the IRL procedure) change the behavior of all agents, potentially destabilizing the training process (Mnih et al. 2015). However, this was not found to be the case in this work.

5.1.4.1 Solving the Forward RL Problem

Most IRL algorithms require a method that evaluates the current parameters of the reward model within the algorithm (i.e., finding an optimal policy given the current reward model). For complex tasks where finding a solution to the forward RL problem is hard, IRL can quickly become impractical (Finn et al. 2016, Wu et al. 2020). The task of finding an optimal policy for any given MDP is

usually much harder than finding an optimal action (or trajectory) for the same MDP given a specific state (learning vs. planning)(Sutton and Barto 2018). The MCTS-based cooperative trajectory planning algorithm (Kurzer et al. 2018a) is thus instrumental in the IRL setting, as it generates near-optimal trajectories for arbitrary reward models quickly. Thus, this work employs this algorithm to solve the forward RL problem.

5.1.4.2 Reward Model

The reward model is a central part of an RL system, as the goal of RL is to maximize the cumulative discounted reward by finding the optimal policy (Sutton and Barto 2018).

Initially IRL applied solely linear reward models, that are represented as a linear combination of features $\phi(s, a)$ and parameters $\boldsymbol{\theta}$ (5.13)(Abbeel and Ng 2004). However, especially for larger RL problems, linear reward models have been outperformed by nonlinear reward models such as NNs (Wulfmeier et al. 2016a, Finn et al. 2016). This work uses a linear reward model and a nonlinear reward model in the form of a NN.

5.1.4.2.1 Features Similar to many other planning methods (Naumann et al. 2020), the cooperative trajectory planner assumes the desired lane index i_{des} as well as the desired velocity v_{des} for each agent.

The desired lane index represents the lane an agent prefers to occupy, with the rightmost lane denoted by an index of 0. At a particular time step t, i_t denotes the current lane index of the agent. Further, the variables l_{center} and l_{width} represent the agent's current lane's lateral position and width, respectively.

State and action-dependent features $\phi(s, a)$ are scalar values that consider specific characteristics of a state and action. Each feature is evaluated for each time step t of the trajectory.

$$\phi(\tau) = \frac{1}{T} \sum_{(s_t, a_t) \in \tau}^{T} \phi(s_t, a_t). \tag{5.4}$$

The parameters $\boldsymbol{\theta}$ are identical for all agents; however, features are not. All features are normalized to lie between $[-1, 1]$ for the length T of a trajectory τ. The feature for the desired lane is defined as

$$\phi_{\text{desLane}}(\tau) = \frac{1}{T} \sum_{(s_t, a_t) \in \tau}^{T} \max\left(1 - \left|i_t - i_{\text{des}}\right|, -1\right), \tag{5.5}$$

encouraging the agent to drive on the desired lane. A deviation from the desired velocity v_{des} larger than $10\,\%$ results in a negative feature value,

$$\phi_{\text{desVelocity}}(\tau) = \frac{1}{T} \sum_{(s_t, a_t) \in \tau}^{T} \max\left(1 - 10\left|\frac{v_t}{v_{\text{des}}} - 1\right|, -1\right). \tag{5.6}$$

Similarly, deviating more than a quarter of the lane width l_{width} from the lane center l_{center} yields a negative feature value,

$$\phi_{\text{laneCenter}}(\tau) = \frac{1}{T} \sum_{(s_t, a_t) \in \tau}^{T} \max\left(1 - \frac{|l_{\text{center}} - x_{\text{lat}}|}{l_{\text{width}}/4}, -1\right). \tag{5.7}$$

To prevent excessive accelerations, a proxy value scaled by gravity for the acceleration a of an action is computed using the Root Mean Square (RMS) approach,

$$\text{RMS}_{\text{acceleration}} = \frac{1}{g} \sqrt{\frac{\int_t^{t+\Delta T} a_t^2 \, \mathrm{d}t}{\Delta T}}. \tag{5.8}$$

The RMS is an advantageous method of quantifying acceleration because it effectively encompasses both the direction and magnitude of acceleration. In addition, incorporating the square operation within the RMS computation means it assigns

greater significance to large accelerations. If the computed RMS acceleration value surpasses a quarter of the gravity g, the feature turns negative,

$$\phi_{\text{acceleration}}(\tau) = \frac{1}{T} \sum_{(s_t, a_t) \in \tau}^{T} \max\left(1 - \frac{\text{RMS}_{\text{acceleration}})}{g/4}, -1\right). \tag{5.9}$$

In addition, the following binary features are defined for trajectories that either result in collisions (5.10), invalid states (i.e., an agent drives off the road) (5.11) or invalid actions (5.12) (i.e., an agent executes a physically impossible action). Each of these binary features marks a terminal state.

$$\phi_{\text{collision}}(\tau) = \mathbb{1}_{\text{collision}}(\tau) \tag{5.10}$$

$$\phi_{\text{invalid state}}(\tau) = \mathbb{1}_{\text{invalid state}}(\tau) \tag{5.11}$$

$$\phi_{\text{invalid action}}(\tau) = \mathbb{1}_{\text{invalid action}}(\tau) \tag{5.12}$$

5.1.4.2.2 Linear Reward Model The linear reward model is a linear combination of the parameters $\boldsymbol{\theta}$ and their respective features $\phi(s, a)$,

$$\mathcal{R}_{\boldsymbol{\theta}}(s_t, a_t) = \boldsymbol{\theta}^{\top} \phi(s_t, a_t). \tag{5.13}$$

The feature count is normalized using the length of the trajectory. Since each feature is bounded between $[-1, 1]$ the return of a trajectory is bounded between $[-||\boldsymbol{\theta}||, ||\boldsymbol{\theta}||]$.

5.1.4.2.3 Nonlinear Reward Model To allow for a more complex reward structure, a nonlinear reward model in the form of a fully connected NN is introduced,

$$\mathcal{R}_{\boldsymbol{\theta}}(s_t, a_t, s_{t-1}) = W_2 \Gamma(W_1 \phi(s_t, a_t, s_{t-1})). \tag{5.14}$$

It consists of two layers, with parameters W_1 and W_2, respectively. The first layer is followed by a ReLU activation function Γ. The inputs to the NN are the features for the linear model in addition to the values of ϕ_{desLane}, $\phi_{\text{desVelocity}}$ and $\phi_{\text{laneCenter}}$ at the previous time step.

5.1.4.3 Guided Cost Learning

GCL is an algorithm that combines sampling-based Maximum Entropy IRL (see Section 2.4.1) with RL (Finn et al. 2016).

Since the partition function (2.23) can only be calculated for small and discrete MDPs, it cannot be computed for the cooperative trajectory planning problem. Instead, GCL circumvents this problem by sampling to approximate it.

It estimates the partition function (2.23) using a distribution over trajectories generated by a sampling-based method; in this work, the MCTS-based cooperative trajectory planner (Kurzer et al. 2018a) (5.17). This sampling-based method overcomes the computational limitations of directly calculating the partition function. The ideal proposal density for importance sampling, which minimizes the variance of the estimate (Finn et al. 2016), is given by

$$\rho_{\text{S}}^{*}(\tau) \propto e^{G_{\boldsymbol{\theta}}(\tau)}. \tag{5.15}$$

The key concept of GCL is the adjustment of this sampling distribution to the distribution that follows from the current reward model (2.22). In order to achieve this within the MCTS, this work introduces a probabilistic final selection policy named *Softmax Q-Proposal*,

$$\pi_{\text{MCTS}}(a|s_0) = \frac{e^{cQ^{\pi}(s_0,a)}}{\sum_{a\in\mathcal{A}(s_0)} e^{cQ^{\pi}(s_0,a)}}. \tag{5.16}$$

The numerator is the exponentiated state-action value $Q^{\pi}(s_0, a)$ (i.e., the expected return (2.2)) of taking action a in root state s_0 over the sum of the state-action values of all explored actions $\mathcal{A}$ in the root state s_0. The coefficient c can be

used to scale the variance of the distribution, its value is determined empirically. Based on (2.21), this results in the following distribution over trajectories

$$\rho(\tau) = \rho_{\mathrm{MCTS}}(s_0, a_0, s_1, a_1, \ldots, s_T) = \prod_{t=0}^{T-1} \pi_{\mathrm{MCTS}}(a_t|s_t). \tag{5.17}$$

Applying importance sampling (see Section 2.4.2) the expectation in (2.30) can be calculated using the policy $\rho_{\mathrm{S}}(\tau)$ (2.31),

$$\mathbb{E}_{\tau\sim\rho_{\mathrm{E}}(\tau)}\left[\nabla_{\boldsymbol{\theta}} G_{\boldsymbol{\theta}}(\tau)\right] = \mathbb{E}_{\tau\sim\rho_{\mathrm{S}}(\tau)}\left[\frac{e^{G_{\boldsymbol{\theta}}(\tau)}}{\rho_{\mathrm{S}}(\tau) Z_{\boldsymbol{\theta}}} \nabla_{\boldsymbol{\theta}} G_{\boldsymbol{\theta}}(\tau)\right]. \tag{5.18}$$

Further, the partition function (2.32) can be approximated using importance sampling as well (2.33). Substituting the expectation in (2.30) with (5.18) as well as the partition function (2.23) with (2.33), the final approximation of the gradient can be obtained (2.34). Given this form of the gradient, the proposed Softmax Q-IRL algorithm (Alg. 1) performs gradient ascent, converging towards the expert behavior. The necessary data sampling routine (Alg. 1 Line 5) is depicted in

Algorithm 1: Softmax Q-IRL

Input: $\mathcal{T}_{\mathrm{E}}$
Output: $\boldsymbol{\theta}$

1 $\boldsymbol{\theta}_0 \sim U[-1, 1]$;
2 **for** $i \leftarrow 0$ **to** M **do**
3 $\quad \mathcal{T}_{\mathrm{S}} \leftarrow \varnothing$;
4 $\quad$ **for** $j \leftarrow 0$ **to** N **do**
5 $\quad\quad \mathcal{T}_{\mathrm{S}}, \Pi_{\mathrm{S}} \leftarrow (\mathcal{T}_{\mathrm{S}}, \Pi_{\mathrm{S}}) \cup$ `generateSamples(`$\boldsymbol{\theta}$`)`;
6 $\quad$ **end**
7 $\quad \widehat{\nabla_{\boldsymbol{\theta}} l(\boldsymbol{\theta})} = \frac{1}{|\mathcal{T}_{\mathrm{E}}|} \sum_{\tau\in\mathcal{T}_{\mathrm{E}}} \nabla_{\boldsymbol{\theta}} G_{\boldsymbol{\theta}}(\tau) - \frac{1}{|\mathcal{T}_{\mathrm{S}}|} \sum_{\tau\in\mathcal{T}_{\mathrm{S}}} \frac{e^{G_{\boldsymbol{\theta}}(\tau)}}{\rho_{\mathrm{S}}(\tau)\widehat{Z}_{\boldsymbol{\theta}}} \nabla_{\boldsymbol{\theta}} G_{\boldsymbol{\theta}}(\tau)$;
8 $\quad \boldsymbol{\theta}_{i+1} \leftarrow \boldsymbol{\theta}_i + \alpha \widehat{\nabla_{\boldsymbol{\theta}} l(\boldsymbol{\theta}_i)}$;
9 **end**
10 **return** $\boldsymbol{\theta}$

Alg. 2. It generates the sample trajectories $\mathcal{T}_\mathrm{S}$ as well as their policies Π. Here, Υ denotes the number of agents in the respective scenario.

Algorithm 2: Sampling of Trajectories and Policies

Function `generateSamples(`$\boldsymbol{\theta}$`)`
  $\mathcal{T} \leftarrow \varnothing$; $\Pi \leftarrow \varnothing$; $\rho(\cdot) \leftarrow 1$;
  `// sample from the start state distribution`
  $s_0 \sim d$;
  for $t \leftarrow 0$ **to** $T-1$ **do**
    `// estimate for each action explored at the root state`
    $\widehat{Q}(s_t, a_0), \ldots, \widehat{Q}(s_t, a_m) \leftarrow$ `MCTSQEstimate(`$\boldsymbol{\theta}, s_t$`)`;
    $\pi_\mathrm{MCTS}(a|s_t) \leftarrow \frac{e^{c\widehat{Q}(s_t,a)}}{\sum_{a \in \mathcal{A}(s_t)} e^{c\widehat{Q}(s_t,a)}}$;
    `// for each agent in the scenario`
    for $i \leftarrow 0$ **to** $|\Upsilon|$ **do**
      $a_i \sim \pi_\mathrm{MCTS}(a|s_t)$;
      $\rho(\tau_i) \leftarrow \rho(\tau_i)\pi_\mathrm{MCTS}(a_i|s_t)$;
      $\tau_i \leftarrow \tau_i \cup (s_t, a_i)$;
      if $t = T-1$ **then**
        $\mathcal{T} \leftarrow \mathcal{T} \cup \tau_i$;
        $\Pi \leftarrow \Pi \cup \rho(\tau_i)$;
      end
    end
    $s_t \leftarrow$ `EnvironmentStep(`$s_t, a_0, \ldots, a_m$`)`;
  end
  return $\mathcal{T}, \Pi$
end

5.2 Parallelization

In autonomous driving, real-time decision-making is essential to ensure safety and efficiency. Therefore, expediting the process of finding optimal policies by leveraging computational resources is paramount. This section delves into the potential of parallelizing the algorithm to accelerate policy optimization, contributing to the overarching goal of efficient problem-solving.

5.2.1 Introduction

With an ever-increasing availability of powerful and affordable parallel-computing infrastructure on-premise and in the cloud, parallelization is more crucial than ever. In the past, the parallelization of algorithms, e.g., for NNs (Seiffert 2004), increased their capabilities tremendously.

First, two parallelization strategies are applied to the cooperative trajectory planning algorithm introduced in Section 4.4. While leaf parallelization is directly applicable to MCTS with continuous action spaces (Cazenave and Jouandeau 2007, Chaslot et al. 2008b), this is not the case for root parallelization. Hence, a mechanism to merge unrelated search trees is devised.

5.2.2 Related Work

Similarly, prior research on the parallelization of MCTS has demonstrated that it can benefit significantly from today's multiprocessor architectures to speed up the search (Cazenave and Jouandeau 2007, Chaslot et al. 2008b, Soejima et al. 2010, Enzenberger and Müller 2010, Bourki et al. 2011, Rocki and Suda 2011).

While the previous approaches have studied the parallelization of MCTS in environments with discrete action spaces (mainly using the game of Go), this work requires an extension to the continuous domain. It builds on research of MCTS in continuous domains (Couëtoux et al. 2011, Rolet et al. 2009, Yee et al. 2016,

Kurzer et al. 2018b, Moerland et al. 2018). The three predominant types of parallelization are leaf parallelization, root parallelization, and tree parallelization (Browne et al. 2012, Chaslot et al. 2008b). Of these, leaf parallelization and root parallelization have been adapted in this work to continuous action spaces (Kurzer et al. 2020b).

5.2.3 Problem Statement

Based on well-known parallelization techniques successfully applied to MCTS in discrete domains, the cooperative trajectory planning algorithm ought to be parallelized. Here, aggregating information from differently explored trees is a crucial challenge for root parallelization.

5.2.4 Approach

In the following, leaf and root parallelization techniques are applied to MCTS with a continuous action space.

5.2.4.1 Leaf Parallelization

Leaf parallelization is the simplest form of parallelization. Here only the simulation policy needs to be adapted (Kurzer et al. 2020b). Then, a separate simulation is run for each thread starting from the previously expanded node. Finally, once all simulations have terminated (i.e., reached a terminal state or the planning horizon), the results of the simulations are aggregated, and the traversed branch is updated. Due to the aggregation of results, this method requires the synchronization of threads, and thus slower or longer simulations block further execution. Leaf parallelization primarily achieves a reduction of the standard error of the mean (SEM).

5.2.4.1.1 Mean The simplest form of leaf parallelization uses the mean simulation return, defined as

$$\overline{G}_{\text{sim}} = \frac{1}{|\Xi|} \sum_{\xi \in \Xi} G_{\text{sim}}(\xi), \tag{5.19}$$

i.e., the cumulated sum of returns over all threads $\xi \in \Xi$ divided by the number of threads. An issue with using the mean is that environments that require a precise selection of actions become harder to solve (Soemers et al. 2016). This issue arises as a large part of simulations will lead to undesirable states; the small part that does reach the desired state will be outweighed and thus result in an overly pessimistic estimate for the state-action value.

5.2.4.1.2 Maximum The maximum return can be used to circumvent the issue of taking the mean of the simulation. While this is possible in cooperative environments, it cannot be generalized to adversarial multiplayer environments since it might lead to traps (Soemers et al. 2016). The maximum simulation return is the value from the thread ξ that generated the maximum cumulated sum of rewards over all time steps t during simulation,

$$G^*_{\text{sim}} = \max_{\xi} G_{\text{sim}}(\xi). \tag{5.20}$$

5.2.4.2 Root Parallelization

Root parallelization grows multiple trees from a single root instead of leaf parallelization (Kurzer et al. 2020b). This parallelization technique fosters exploration. Each thread holds a copy of the root and grows a unique tree. When the computational budget has been reached, the trees of all threads are merged. The merging can be conducted in several different ways. One possible solution is a voting mechanism that allows each tree to submit its best action, and the majority vote then selects the final action (Soejima et al. 2010).

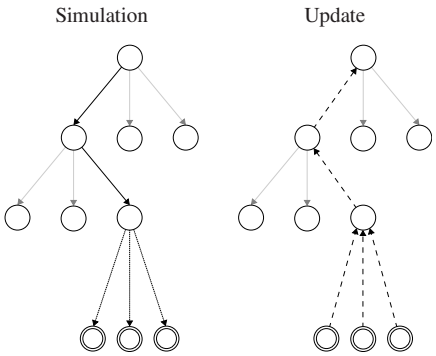


Figure 5.2: Leaf Parallelization of MCTS: The results of multiple simulations are aggregated and used to update all nodes along the traversed branch of the tree.

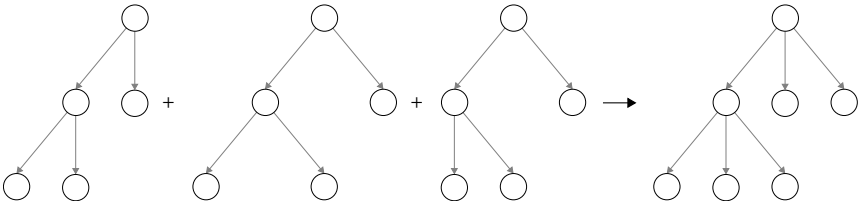


Figure 5.3: Root Parallelization in MCTS: The results of multiple trees are aggregated and used to form one final tree.

5.2.4.2.1 Similarity Merge Using the similarity update depicted in Fig. 4.10, multiple trees can be merged into a single tree, from which a final action can be chosen.

The process is described in Algorithm 3. First, all actions of all trees are added to the final tree, line 4. Second, a pairwise similarity update is conducted between all actions of the final tree and the simulated trees. For this, the similarity matrix is determined, line 10, and the similarity state-action value is calculated based on the weighted visit count and state-action value, line 11. Last, the action with the maximum state-action value from the final tree is returned.

The computational complexity of this method is predominantly determined by the formation of the similarity matrix and the calculation of the similarity state-action

value, both of which have to be conducted pairwise for actions, yielding a time complexity of $O\left(|\mathcal{A}|^2\right)$, where $|\mathcal{A}|$ is the cardinality of the set of actions. In the context of parallel processing with multiple threads, if each thread corresponds to a separate tree and can generate a distinct set of actions, the total number of actions for the similarity calculation can grow with the number of threads. This increase leads to an overall complexity of $O\left(\left(|\mathcal{A}||\Xi|\right)^2\right)$, where $|\mathcal{A}||\Xi|$ represents the product of the cardinalities of the set of actions and the set of threads respectively.

Algorithm 3: Similarity Merge

1 **Function** `similarityMerge`(Ψ: *Tree[]*)
2 $\quad\mathcal{A}_{\text{final}} \leftarrow \varnothing$; $Q_{\text{sim}}(s_0, \cdot) \leftarrow 0$;
3 $\quad$**for** $\psi \in \Psi$ **do**
4 $\quad\quad\mathcal{A}_{\text{final}} \leftarrow \mathcal{A}_{\text{final}} \cup \mathcal{A}_{\psi}$;
5 $\quad$**end**
6 $\quad\boldsymbol{K} \leftarrow \mathbf{0}_{|\mathcal{A}_{\text{final}}| \times |\mathcal{A}_{\text{final}}|}$;
7 $\quad$**for** $i \leftarrow 0$ **to** $|\mathcal{A}_{\text{final}}|$ **do**
8 $\quad\quad$**for** $j \leftarrow 0$ **to** $|\mathcal{A}_{\text{final}}|$ **do**
9 $\quad\quad\quad$**if** $i \neq j$ **then**
10 $\quad\quad\quad\quad\boldsymbol{K}_{ij} \leftarrow \exp\left(-\gamma \left\|a_i - a_j\right\|^2\right)$;
11 $\quad\quad\quad\quad Q_{\text{sim}}(s_0, a_i) \leftarrow \frac{N(s_0,a_i)Q(s_0,a_i)+\boldsymbol{K}_{ij}N(s_0,a_j)Q(s_0,a_j)}{N(s_0,a_i)+\boldsymbol{K}_{ij}N(s_0,a_j)}$;
12 $\quad\quad\quad$**end**
13 $\quad\quad$**end**
14 $\quad$**end**
15 $\quad$**return** Q_{sim};
16 **end**

5.2.4.2.2 Similarity Vote Inspired by a voting scheme for root parallelization, an extension for continuous domains is developed, see Algorithm 4. It does not merely choose the action with the overall highest visit count or state-action value but instead lets each tree vote for an action (Soejima et al. 2010). Analogous to (Soejima et al. 2010), each tree submits its best action, line 4. Then a similarity matrix $\boldsymbol{K}$ is calculated to store the similarity of a chosen action from one tree

to all actions from all trees, line 9. Weighted by the state-action values of the submitted actions, the final action is the one that maximizes the similarity vote, line 12.

Algorithm 4: Similarity Vote

1 **Function** `similarityVote`(Ψ: *Tree[]*)
2 $\quad \mathcal{A}_{\text{final}} \leftarrow \varnothing$; $Q_{\text{sim}}(s_0, \cdot) \leftarrow 0$;
3 $\quad$ **for** $\psi \in \Psi$ **do**
4 $\quad\quad \mathcal{A}_{\text{final}} \leftarrow \mathcal{A}_{\text{final}} \cup \arg\max_a Q_\psi(s_0, a)$;
5 $\quad$ **end**
6 $\quad \boldsymbol{K} \leftarrow \mathbf{0}_{|\mathcal{A}_{\text{final}}| \times |\mathcal{A}_{\text{final}}|}$;
7 $\quad$ **for** $i \leftarrow 0$ **to** $|\mathcal{A}_{\text{final}}|$ **do**
8 $\quad\quad$ **for** $j \leftarrow 0$ **to** $|\mathcal{A}_{\text{final}}|$ **do**
9 $\quad\quad\quad \boldsymbol{K}_{ij} \leftarrow \exp\left(-\gamma \left\|a_i - a_j\right\|^2\right)$;
10 $\quad\quad\quad Q_{\text{sim}}(s_0, a_i) \leftarrow Q_{\text{sim}}(s_0, a_i) + \boldsymbol{K}_{ij} Q(s_0, a_j)$;
11 $\quad\quad$ **end**
12 $\quad$ **end**
13 $\quad$ **return** $\arg\max_{a \in \mathcal{A}_{\text{final}}} Q_{\text{sim}}(s_0, a)$;
14 **end**

5.3 Hyperparameter Optimization

The performance and robustness of algorithms in autonomous driving are critically dependent on the fine-tuning of Hyperparameters (HPs). However, selecting optimal values for these HPs can be non-trivial and computationally expensive. This section explores the utility of HPO in streamlining the process and enhancing the efficiency of finding optimal policies, aligning with the imperative of solving the problem efficiently.

5.3.1 Introduction

Similar to many systems, the performance of the cooperative trajectory planning algorithm introduced in Section 4.4 is dependent on a large variety of exogenously specified parameters (i.e., HPs). Such parameters include but are not limited to the planning horizon, the discount factor, coefficients for progressive widening and similarity update, as well as the choice of the respective policies.

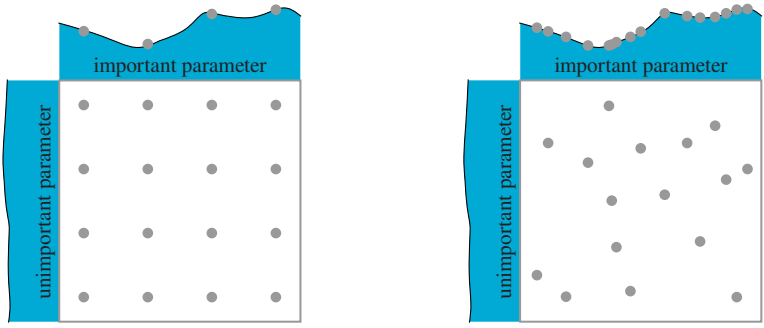
While care has been taken in choosing the "right" HPC, the choices are frequently convoluted and high-dimensional, with interactions that make them difficult to reason about for humans. Thus, to improve upon the manually-tuned baseline HPC, the following describes an approach that jointly optimizes these HPs, resulting in significant improvements.

5.3.2 Preliminaries

The following introduces common algorithms used for optimizing unknown objective functions.

5.3.2.1 Model-Free Optimization

One of the simplest forms of HPO is Grid Search (GS) (Bergstra and Bengio 2012). Since its implementation is trivial and can easily be parallelized, it usually outperforms manual tuning of HPs. GS examines the Cartesian product of finite sets of values for each HP, cf. Fig. 5.4a. Since the required function evaluations increase exponentially with the dimensionality of the hyperparameter space, denser evaluations of the hyperparameter space quickly become intractable (Feurer and Hutter 2019). Thus, GS is well suited for very low dimensional hyperparameter spaces (1D, 2D). Further, it is unfitting for higher-dimensional ones as large amounts of samples explore irrelevant HPs without insufficient exploration of the relevant ones, cf. Fig. 5.4a.



(a) Grid Search **(b)** Random Search

Figure 5.4: Comparison of GS and RS: Using GS and RS for HPO of an unknown objective function can lead to vastly different results. Since the HPs have differing importance (one important and one unimportant), 16 samples in the case of GS effectively generate only four relevant observations (Bergstra and Bengio 2012, Feurer and Hutter 2019).

Random Search (RS) is an alternative to GS, which outperforms GS in most cases given the same computational budget (Bergstra and Bengio 2012). As the name suggests, RS selects HPCs at random, see Fig. 5.4b. For a number n of function evaluations and k HPs RS is likely to produce n distinct observations for each dimension, where GS generates only $n^{1/k}$ observations (Feurer and Hutter 2019), cf. Fig. 5.4. Another advantage of RS is that it does not make assumptions about the underlying unknown objective function and converges towards the global optimum.

5.3.2.2 Sequential Model-Based Optimization

Unlike model-free optimization, model-based optimization uses an intermediary model to guide the search towards promising areas, making informed choices essential for optimizing unknown objective functions that are expensive to evaluate (Feurer and Hutter 2019).

5.3.2.2.1 Bayesian Optimization A probabilistic surrogate model and an acquisition function to choose which HPC to assess next are the two key components of the iterative Bayesian Optimization (BO) method (Feurer and Hutter 2019). First, the surrogate model is fitted to all observations from the unknown objective function. Second, the acquisition function decides the utility of various HPCs since it is cheap to evaluate. It decides by balancing exploration and exploitation, using the prediction distribution of the probabilistic surrogate model, by sampling from areas of high uncertainty (exploration) and areas that are likely to improve over the current best observation (exploitation). The method is illustrated in Fig. 5.5.

Its name is derived from Bayes theorem (Brochu et al. 2010),

$$P\left(f \mid \mathcal{D}_{1:n}\right) \propto P\left(\mathcal{D}_{1:n} \mid f\right) P(f). \tag{5.21}$$

Here f denotes the unknown objective function $f(\mathbf{x})$, which is to be maximized. While it is unknown, certain assumptions regarding its characteristics are specified by its prior $P(f)$. The prior is multiplied by the likelihood of the observations $\mathcal{D}$ given the unknown objective function. The result is then proportional to the posterior probability of the unknown objective function given the observations, i.e., the updated belief about the unknown objective function.

Based on this posterior, an acquisition function determines the configuration of HPs to sample, yielding the subsequent observation.

A typical prior for BO is the Gaussian Process (GP), which extends the Gaussian distribution to a distribution over functions. It is specified by its mean function μ and the covariance function K,

$$f(\mathrm{x}) \sim \mathcal{GP}\left(\mu(\mathbf{x}), K\left(\mathbf{x}, \mathbf{x}'\right)\right). \tag{5.22}$$

Instead of returning a scalar value for a given value of $\mathbf{x}$, a GP returns a Gaussian distribution with a mean and variance.

Another possibility for a prior in BO is a Random Forest (RF); it is especially beneficial for larger configuration spaces as well as categorical and conditional variables (Hutter et al. 2011).

5.3.3 Problem Statement

HPO is the process of optimizing the values for parameters of an algorithm or model that typically are set manually (i.e., using expert knowledge) in an automated manner. Mathematically, HPO tries to maximize (or minimize) an unknown objective function f parameterized by its parameters $\mathbf{x} \in \mathcal{X}$, where $\mathcal{X}$ is the space of all valid parameter combinations (Shahriari et al. 2016), defined as

$$\mathbf{x}^* = \arg\max_{\mathbf{x} \in \mathcal{X}} f(\mathbf{x}). \tag{5.23}$$

It is assumed that the unknown objective function $f(\mathbf{x})$ can be evaluated for any valid value of $\mathbf{x}$ and has no closed form. The evaluations of the unknown objective function result in noisy observations $y \in \mathbb{R}$, mapping the HPs into a single-dimensional ordinal space.

5.3.4 Approach

The following nine HPs were automatically tuned in the first step with RS and BO using the Sequential Model Algorithm Configuration (SMAC) (Lindauer et al. 2022) and the highly parallelized implementation of the cooperative trajectory planning algorithm, see Section A.1.

- UCT-coefficient
- discount factor
- maximum search depth
- action duration
- action execution fraction
- progressive widening-coefficient

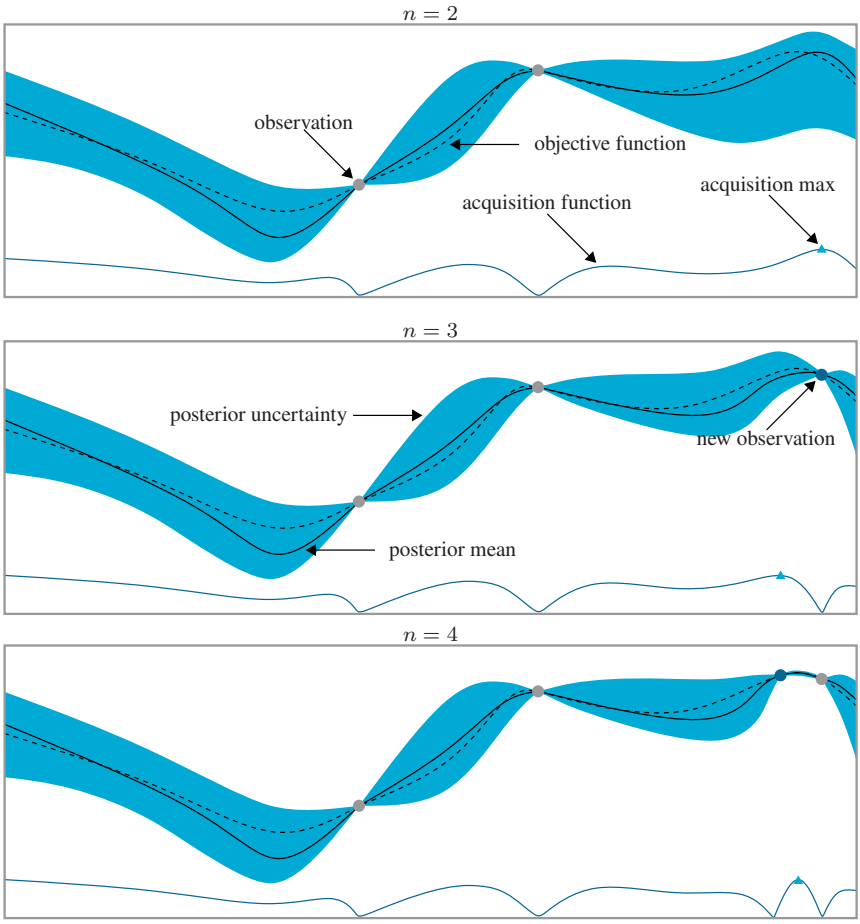


Figure 5.5: Process of Bayesian Optimization (Brochu et al. 2010): The plots depict the process over three steps after the first two observations have been made. The estimates of the mean and the uncertainty of the unknown objective function are depicted by the solid black line and the shaded blue area. For illustration purposes, the unknown objective function is shown by the dashed black line. The solid blue line displays the acquisition function at the bottom of the plots. It reaches its maximum in regions where the mean estimate is high (exploitation) and the uncertainty estimate is high (exploration).

- progressive widening-exponent
- maximum depth for progressive widening
- final selection strategy

Optimizing these parameters enables the cooperative trajectory planning algorithm to achieve improved performance, facilitating more efficient identification of optimal actions. The detailed results and analysis of the HPO are presented Section 6.6.

5.4 Further Research

Apart from the additions presented in Chapter 5, other promising improvement areas were explored. However, they were not integrated into the framework. The research has been published in the following papers Kurzer et al. 2020a and Stegmaier et al. 2022. Parts of it are based on the following supervised theses Fechner 2020 and Stegmaier 2021.

The below is merely a brief overview of these works, and the interested reader is encouraged to refer to the respective publication.

5.4.1 Uncertainty

One challenging aspect of cooperative trajectory planning is the uncertainty surrounding the state of the environment due to limited sensor accuracy. A POMDP can represent this uncertainty. This problem is addressed by extending the existing cooperative trajectory planning approach. It does so by explicitly modeling uncertainties as a root belief state, from which tree start states are sampled. After the trees have been constructed with MCTS, their results are aggregated into return distributions using kernel regression. Two risk metrics are applied for the final selection: a Lower Confidence Bound and a Conditional Value at Risk. It can

be shown that the integration of risk metrics in the final selection policy consistently outperforms the baseline algorithm in uncertain environments, generating considerably safer trajectories. (Stegmaier et al. 2022)

5.4.2 Learned Heuristics

Another challenging aspect of cooperative trajectory planning is the search space size. With a growing number of agents and planning depth, it grows so vast that most of the computational budget is spent exploring the search space in unpromising regions far from the solution. Inspired by human thinking, learned heuristics can be combined with the cooperative planning method to accelerate planning and guide the search toward promising regions. This is achieved by training a mixture density network over the action space of an agent and using it to bias its action selection strategy. It can be shown that the integration of learned heuristics outperforms the baseline algorithm and yields better solutions at lower computational costs when the computational budget is low. (Kurzer et al. 2022)

5.5 Summary

The first section of this chapter discusses how a reward function can be learned from expert demonstrations to produce intuitive behavior for AVs. Predictable trajectories for AVs are essential to avoid confusion and accidents among human drivers. The approach combines a cooperative trajectory planning algorithm using MCTS with Maximum Entropy IRL to learn a reward model that produces trajectories resembling expert demonstrations. This approach generates near-optimal trajectories for arbitrary reward models and is efficient enough to be used in an IRL setting. A linear and a nonlinear reward model (using an NN) are developed.

The subsequent section discusses the importance of real-time decision-making in autonomous driving and the benefits of parallelizing algorithms for policy

optimization. Two parallelization strategies, leaf parallelization, and root parallelization, are applied to the continuous action space of the cooperative trajectory planning algorithm. Leaf parallelization involves adapting the simulation policy and running a separate simulation for each thread. It reduces the variance, and the max aggregation promotes optimistic state-action value estimates. On the other hand, root parallelization involves growing multiple trees from a single root, fostering exploration, but requiring special considerations when merging these trees later when using a continuous actions pace. Two methods for merging these trees are proposed. One conducts the merge based on the similarity of actions, while the other uses a voting scheme inspired by earlier works for discrete domains.

The next section discusses the significance of HPO in improving the performance and robustness of the cooperative trajectory planning algorithm. Optimal HPCs are difficult to obtain manually. HPO techniques, such as GS, RS, and BO, can facilitate the process of finding optimal HPCs. While GS is suitable for low-dimensional HP spaces, it becomes intractable for higher-dimensional ones due to the exponential increase in required function evaluations. On the other hand, RS selects HPCs randomly and can outperform GS, given the same computational budget. It converges towards the global optimum without making assumptions about the underlying objective function. Model-based optimization techniques, such as BO, use probabilistic surrogate models and acquisition functions to guide the search toward promising areas. These methods are more sophisticated since they conduct an informed search and hence are particularly well-suited for optimizing unknown objective functions that are expensive to evaluate.

The chapter concludes with additional research on cooperative trajectory planning, focusing on two major challenges:uncertainty and informed action space exploration. While the results are promising, the research was not integrated into the current work.

6 Experiments

The experiments were designed to evaluate the proposed cooperative trajectory planning algorithm's performance in various traffic scenarios. All experiments employed the previously introduced multi-agent driving simulator to simulate the behavior of multiple vehicles interacting with each other in a traffic environment.

Statistical tests were conducted to assess the significance of the results by comparing the performance of the proposed algorithms to a baseline. Each unique test setup, referred to as a configuration, encompasses a specific scenario, a defined number of iterations, and unique values for the hyperparameters, i.e., HPC. To ensure the generation of statistically reliable results, each configuration was assessed using 250 different random seeds. The results of the experiments provide insights into the effectiveness of the algorithm and its extensions.

The key metric used for the performance evaluation is the success rate. The success rate denotes the fraction of evaluated runs that neither yield a collision nor invalid actions or states. An agent's state is invalid if it is outside the road boundaries; similarly, an agent's action is invalid if it is not physically drivable. According to this measure, the cooperative trajectory planning algorithm performs better when it has higher success rates.

Scatterplots and heat maps were used to depict the evaluation results. Scatter plots illustrate the absolute success rate over all scenarios, comparing configurations with an increasing number of iterations (abscissa), e.g., Fig. 6.3a displays the success rate of the baseline configuration and the semantic action classes configuration. Heatmaps visualize either the absolute success rate for a single configuration or the absolute difference in the success rate for two configurations, for an increasing number of iterations (abscissa) and across different scenarios

(ordinate). In these heatmaps, the color scale ranges from white to green for absolute success rate values and from red to blue for absolute differences, with these color scales corresponding to a success rate range of 0 to 1. Heatmaps that depict absolute differences are always relative to a specified baseline, e.g., Fig. 6.3b depicts the difference in the success rate of the baseline configuration and the semantic action classes configuration.

This chapter encompasses several experiments that closely match experiments from prior publications. However, despite this parallel, there may be noticeable differences between the results reported here and those outlined in previous works. Various factors could account for such discrepancies, including changes to the algorithm's internals over time, experimental setup variations, or the algorithms' inherent randomness.

6.1 Statistical Tests

Statistical tests are essential to the evaluation process when comparing a proposed algorithm to a baseline. They allow for a more robust and unbiased assessment of the results since the data support the conclusions drawn.

A Z-Test for two proportions was conducted between the baseline and each configuration to systematically compare the success rate. It assessed whether the observed difference in performance between the baseline and any configuration was statistically significant or if it could have occurred by chance. The Z-Test tested for two-sided equality, where the null hypothesis was that the two proportions were equal, and the alternative hypothesis was that they were not equal, (Montgomery and Runger 2020). The statistical significance level α (p-value) was set to $5\,\%$ (i.e., if the p-value of a test is less than 0.05, then it can be concluded that the observed difference between the samples is unlikely to have occurred by chance, and is instead a significant result). If the deviation from the baseline for any configuration was significant, the scatter mark was annotated by an asterisk. If a configuration was consistently unequal compared to the baseline, i.e., the

deviation for all tested numbers of iterations was significant, an asterisk marks that trace as a suffix in the legend.

6.2 Scenarios

Each scenario consists of three principal components: the road structure, the obstacles present, and the agents involved.

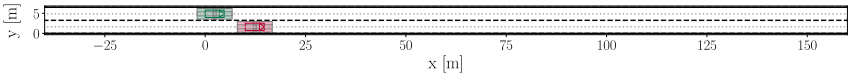
- Road: This element outlines the specifics of the road, detailing the number of lanes and their widths.
- Obstacles: This element comprises a list of all obstacles present in the scenario. Each obstacle is represented as a rectangle with specific dimensions (width and length) and assigned a unique id, position, and heading.
- Agents: This element comprises a list of all agents present in the scenario.

Each agent can be uniquely configured with attributes such as a cooperation factor, cost model, action space, desire, terminal condition, and vehicle properties.

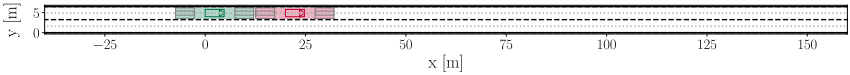
- Cost Model: This model of the costs serves as the central measure of the state-action value based on resulting states and chosen actions.
- Action Space: This delineates the range within which an agent can sample longitudinal velocity changes and lateral position changes.
- Desire: This aspect specifies an agent's desired speed and lane.
- Terminal Condition: This criterion dictates longitudinal and lateral positions that, when reached by vehicles, trigger the end of the scenario, even if the desire was not achieved.
- Vehicle: A vehicle is a type of obstacle with additional properties that include information regarding its velocity, wheelbase, maximum steering angle, maximum acceleration, and maximum speed.

The difficulty level in a scenario is primarily influenced by various complexity factors, such as the number of agents participating, the configuration of obstacles, and the initial velocities and distances between agents. While the least challenging scenarios involve only two agents and no obstacles, the most challenging ones can include up to eight agents alongside numerous obstacles, demanding impeccable coordination and precise navigation.

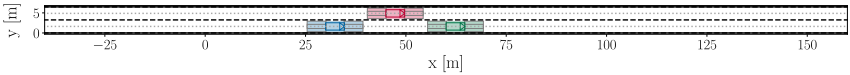
While a scenario had a fixed number of agents and obstacles, its start state was sampled from a probability distribution. Explicitly, the longitudinal and lateral positions of the agents were sampled from a normal distribution within a predefined area (cf. transparent rectangles in Fig. 6.1). Further, different random seeds were used to initialize the sampling-based trajectory planning algorithm, resulting in variations of the chosen actions.



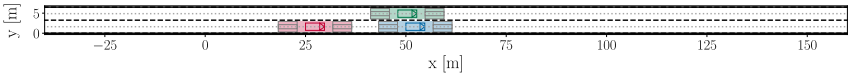
(a) Scenario 01: Delaying merge due to approaching vehicle in desired lane



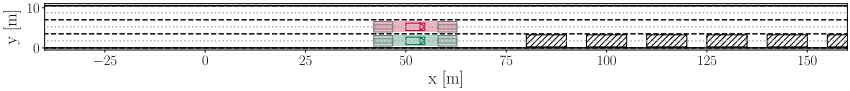
(b) Scenario 02: Reacting to approaching vehicle from behind



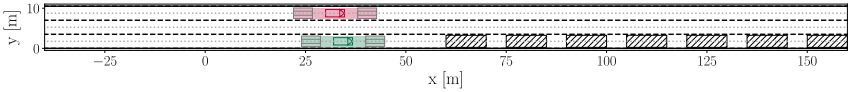
(c) Scenario 03: Merging into moving traffic



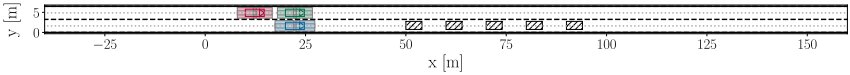
(d) Scenario 04: Merging into moving traffic with prior longitudinal adjustment



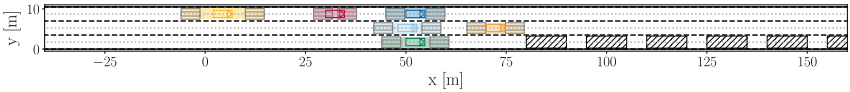
(e) Scenario 05: Changing lane as other vehicle needs to merge onto lane



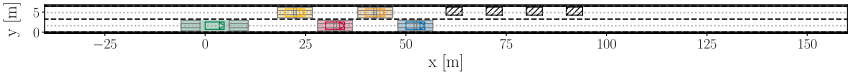
(f) Scenario 06: Delaying lane change as other vehicle needs to merge first



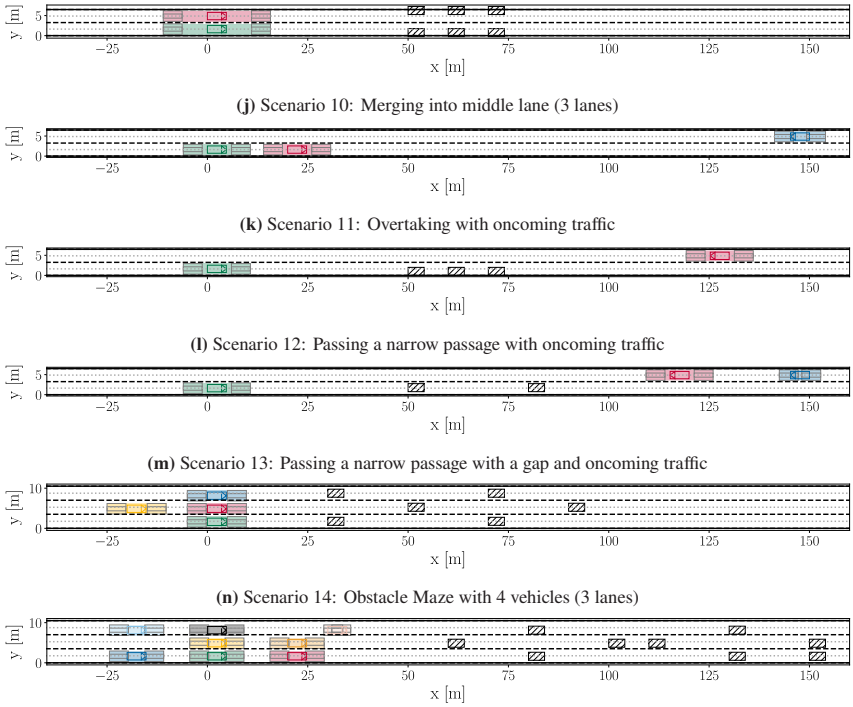
(g) Scenario 07: Merging (1 vehicle) with 3 Vehicles



(h) Scenario 08: Merging (1 vehicle) with 5 Vehicles



(i) Scenario 09: Merging (2 vehicles) with 5 Vehicles



(j) Scenario 10: Merging into middle lane (3 lanes)

(k) Scenario 11: Overtaking with oncoming traffic

(l) Scenario 12: Passing a narrow passage with oncoming traffic

(m) Scenario 13: Passing a narrow passage with a gap and oncoming traffic

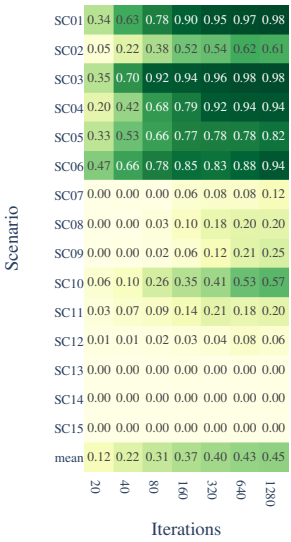
(n) Scenario 14: Obstacle Maze with 4 vehicles (3 lanes)

(o) Scenario 15: Obstacle Maze with 8 vehicles (3 lanes)

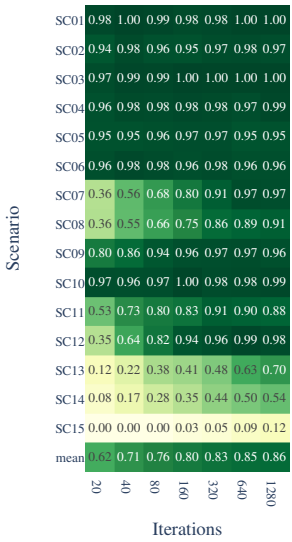
Figure 6.1: Evaluation Scenarios: Longitudinal conflicts, merging, overtaking, and navigating through a bottleneck or obstacle maze. In each scenario, at least two agents with conflicting goals were simulated using the multi-agent driving simulator.

6.3 Baseline and Extensions

The performance of the baseline algorithm (the standard, unmodified version of MCTS with DUCT, referred to as vanilla) and its extensions was evaluated for all 15 scenarios (SC01-SC15) for an increasing number of iterations.



(a) Absolute values of success rate for baseline (with invalid actions) across scenarios



(b) Absolute values of success rate for baseline (without invalid actions) across scenarios

Figure 6.2: Absolute performance of baseline without and with resampling of invalid actions across different iterations and scenarios

6.3.1 Baseline and Semantic Action Classes

The performance of the baseline algorithm was evaluated in a variety of scenarios with different numbers of traffic participants (as shown in Fig. 6.2b). The experiments exhibited a clear improvement in success rate as the number of iterations increased, reaching 92 % for 160 iterations and 97 % for 1280 iterations in scenarios SC01-SC12. Even in the more challenging scenarios SC13-SC15 (with oncoming traffic and many interacting agents), the success rate still exhibited an upward trend as the number of iterations increased. This demonstrates the algorithm's ability to handle complex situations since these scenarios require precise coordination of actions due to the number of agents and obstacle configuration. Overall, the results demonstrate the general feasibility of the approach for a wide range of scenarios.

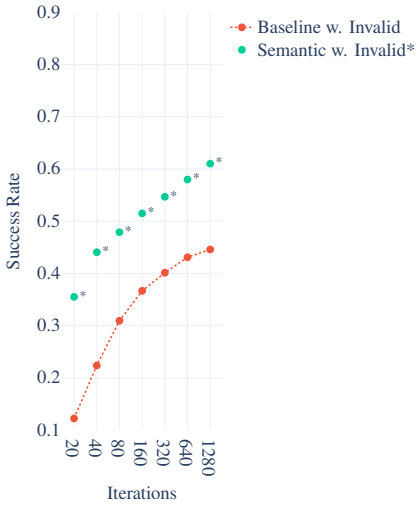
The results depicted in Figure 6.3a support the hypothesis that using semantic action classes (Section 4.3.3), as opposed to randomly selected actions, can improve the performance of the baseline trajectory planning algorithm. This can be attributed to the fact that the semantic action classes are informed by domain knowledge and are designed to increase the frequency of valid actions. Variations in the performance of semantic action classes for different scenarios could be attributed to the fact that they were designed for lane-change and lane-keeping maneuvers rather than subtle movements that some scenarios require. Scenarios SC01-SC06 showed better performance with semantic action classes, especially at fewer iterations. On the other hand, scenarios SC08-SC12 benefited more from semantic action classes at a higher number of iterations. This difference may be related to the fact that scenarios SC01-SC06 were already solved well at a high number of iterations, leaving little room for improvement, in contrast to scenarios SC08-SC12, Fig. 6.2a.

The use of semantic action classes improved the performance of the trajectory planning algorithm when actions were only sampled once. However, the advantage diminished when invalid actions were resampled, as seen in Fig. 6.3c. This can be attributed to the fact that the computation of semantic action classes generated valid trajectories more frequently than random sampling. These results emphasize the need to evaluate the advantages and disadvantages of using semantic action classes in trajectory planning. While they may provide significant benefits in certain situations, they are not generally better.

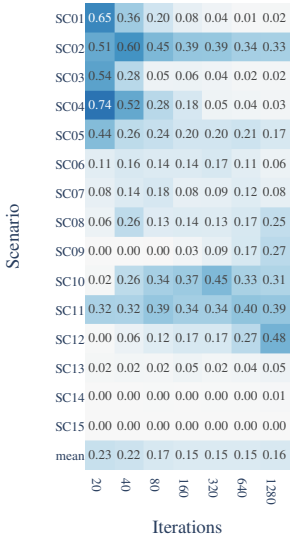
In addition, further evaluations of the blind value, similarity update, and action grouping extensions were conducted, depicted in Fig. 6.7.

6.3.2 Blind Value

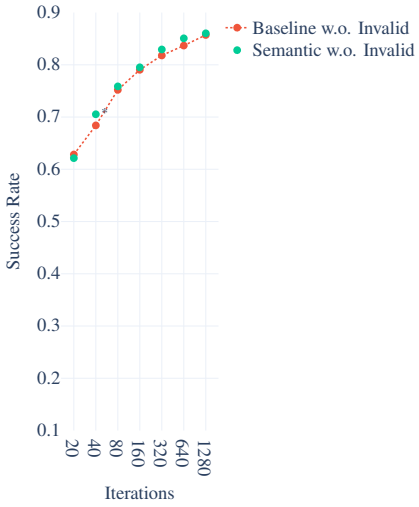
The results in Fig. 6.4a demonstrate that the blind value heuristic to inform action selection during the expansion phase of the MCTS significantly improves the success rate, as compared to random action selection. This can be related to the heuristic allowing the MCTS to make educated decisions about which actions



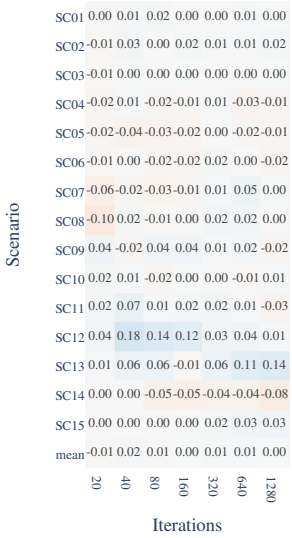
(a) Success rate comparison of baseline (with invalid actions) and semantic action classes (with invalid actions) across iterations



(b) Absolute difference in success rate between baseline (with invalid actions) and semantic action classes (with invalid actions) across scenarios

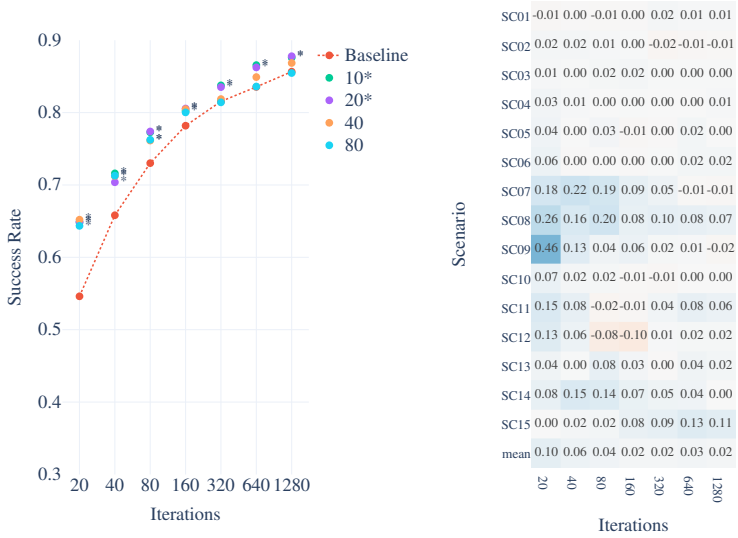


(c) Success rate comparison of baseline (without invalid actions) and semantic action classes (without invalid actions) across iterations



(d) Absolute difference in success rate between baseline (without invalid actions) and semantic action classes (without invalid actions) across scenarios

Figure 6.3: Performance comparison of baseline without and with resampling of invalid actions and semantic action classes



(a) Success rate comparison of baseline and blind value for differing numbers of samples across iterations

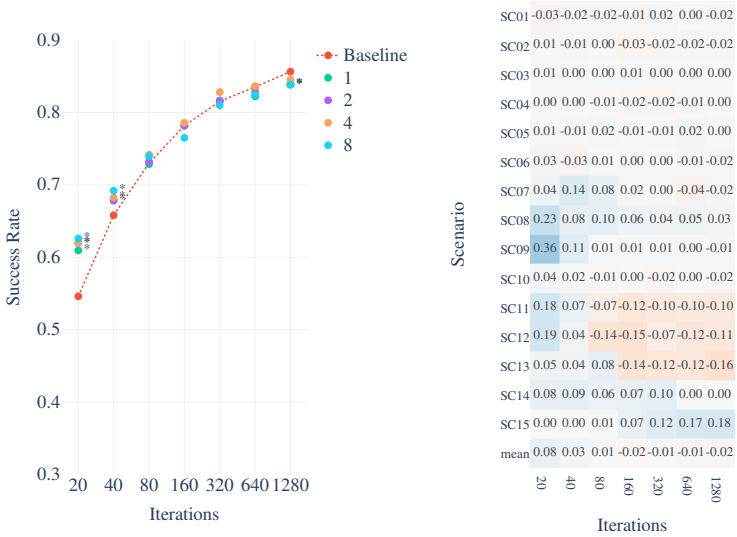
(b) Absolute difference in success rate between baseline and blind value for 10 samples across scenarios

Figure 6.4: Performance comparison of baseline and blind value

to explore based on previously explored actions. The number of samples in the blind value heuristic refers to the number of candidate actions that are evaluated for selection. It was observed that further increases in samples did not result in corresponding performance improvements, particularly for higher numbers of iterations. This effect warrants further investigation to better understand the underlying causes.

6.3.3 Similarity Update

The similarity update was employed to incorporate information from similar but not identical nodes into the update phase. This was achieved by weighting the visit count and mean state-action value of other nodes using a kernel, with the width of the kernel controlled by the parameter γ (where larger values of γ correspond to



(a) Success rate comparison of baseline and similarity update for differing γ-values across iterations

(b) Absolute difference in success rate between baseline and similarity update for $\gamma = 8$ across scenarios

Figure 6.5: Performance comparison of baseline and similarity update

a smaller kernel width). Despite these efforts, no significant change was observed in the overall performance. However, it was noted that the performance did improve significantly for the lowest number of iterations, ranging from 20 to 40, as shown in Fig. 6.5a. A wider kernel width and an increase in the number of iterations should incorporate more information during the similarity update, potentially reducing the accuracy of the state-action value estimate for a particular action. This hypothesis can explain the results, but further investigation is needed to confirm its validity.

6.3.4 Action Grouping

The impact of action grouping is demonstrated in Fig. 6.6a, it utilized semantic action classes. Three boolean configuration options were evaluated for action

grouping. The first option utilized the average statistics of all actions within a group to make the final action decision. The second option combined progressive widening by first selecting the optimal action group and then applying progressive widening within that group. Moreover, the third option relied on the action group statistics instead of node statistics to check if the progressive widening criterion was met. All possible combinations of the mentioned options for action grouping were tested. The results of the experiments showed that different option combinations resulted in vastly different performance outcomes.

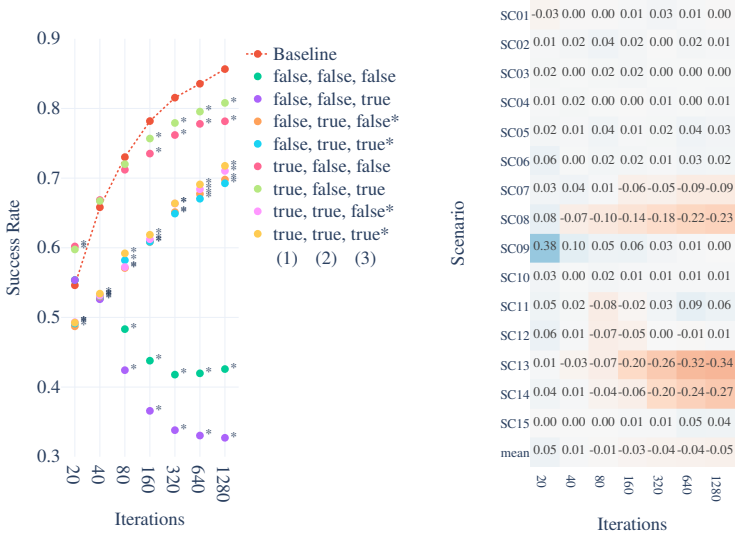
The best performance was achieved when action grouping was used for the final decision without biasing progressive widening (true, false, ?). However, the performance significantly declined when option two was used to bias progressive widening (?, true, ?). On the other hand, when only option three was considered (false, false, ?), the performance of action grouping for the progressive widening criterion was worse compared to any other combinations.

None of the options outperformed the baseline. Similarly to using semantic action classes, the usage of action groups in combination with semantic action classes only had a minor effect for an extremely low number of iterations.

6.3.5 Summary

In conclusion, the evaluation of the baseline trajectory planning algorithm and its extensions demonstrate the approach's feasibility and effectiveness in various scenarios. A combination of all extensions performed similar to its sum of improvements, Fig. 6.7a and Fig. 6.7b.

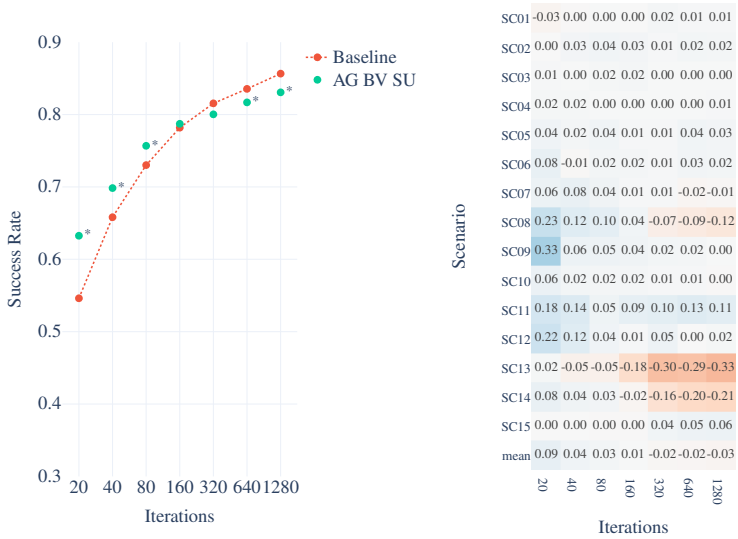
Semantic action classes improved the algorithm's performance, but the advantage diminished when invalid actions were resampled. The blind value heuristic significantly improved the success rate, while the similarity update showed improvements only for low numbers of iterations. The introduction of action grouping showed no significant improvements.



(a) Success rate comparison of baseline and action grouping, used for the final decision (1), used as a bias for progressive widening (2), used as a criterion for progressive widening (3) across iterations

(b) Absolute difference in success rate between baseline and action grouping for final decision(true), bias for progressive widening(false), criteria for progressive widening(true) across scenarios

Figure 6.6: Performance comparison of baseline and action grouping



(a) Success rate comparison of baseline and the combination of the best-found configurations for action grouping, blind value and similarity update (depicted in b, b, and b) across iterations

(b) Absolute difference in success rate between baseline and the combination of the best-found configurations for action grouping, blind value and similarity update (depicted in b, b, and b) across scenarios

Figure 6.7: Performance comparison of baseline and and the combination of action grouping (AG), blind value (BV) and similarity update (SU)

6.4 Inverse Reinforcement Learning

For each of the twelve scenarios (SC01-SC12), a set $\mathcal{T}_\mathrm{E}$ of 50 expert trajectories for each agent in the scenario were generated that depicted (approximately) optimal behavior. Each trajectory had a length of approximately $27\,\mathrm{s}$. This resulted in 600 multi-agent trajectories with a total duration of $4.5\,\mathrm{h}$. The scenarios and the expert trajectories for each agent are depicted in Fig. 6.10.

A linear as well as two nonlinear models were trained. The nonlinear models differed in the number of weights in the hidden layer. The smaller one used five weights, while the larger one used 20. The reward models were trained for 420 gradient steps with a learning rate of 0.0003.

The performance of the models throughout the training is shown in Fig. 6.8. Fig. 6.8a and Fig. 6.8b can be used to gauge the approximation performance with respect to the expert behavior by the learned models. Fig. 6.8a depicts the relative return difference between the expert trajectories $\mathcal{T}_\mathrm{E}$ and the sample trajectories $\mathcal{T}_\mathrm{S}$ given the parametrization of the reward model. The return, as a scalar measure of the cumulative reward obtained over a particular trajectory, indicates the expert's underlying reward function. A smaller relative difference between the returns of the expert and the sample trajectories from the learned model implies a higher degree of alignment between the learned reward function and the expert's behavior. It can be seen that all models reduced the relative difference in the returns throughout the training. The larger nonlinear model reached an absolute minimum relative return difference of $0.3\,\%$ after 130 steps. The smaller nonlinear model reached $3.0\,\%$ after 300 steps, and the linear model reached $11.69\,\%$ after 410 steps.

The mean Euclidean distance between the sample and the expert trajectories considers the agents' longitudinal and lateral positions over the trajectory, accounting for the entire trajectory of the agent rather than just a single point. This can provide a more comprehensive picture of how well the learned model mimics the expert's behavior. However, instead of comparing a sampled trajectory with all

expert trajectories, comparing it with the most similar ones is more effective, accounting for potential variations in expert behavior, e.g., due to multi-modalities. Hence, only the Euclidean distance to the k-nearest neighbor expert trajectories was calculated, providing a more nuanced evaluation. Again it is visible that all models decreased the distance to the expert trajectories. The larger nonlinear model reached a minimum mean Euclidean distance of $4.90\,\mathrm{m}$ after 140 steps. The smaller nonlinear model achieved $4.73\,\mathrm{m}$ after 240 steps, and the linear model reached $8.09\,\mathrm{m}$ after 250 steps.

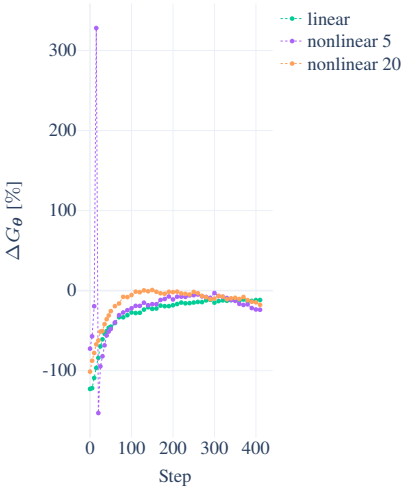
Fig. 6.8c shows that the learned models could reach the desired lane. At the same time, Fig. 6.8d indicates that attaining the desired velocity proved more challenging for these models. However, it is essential to note that only $61.0\,\%$ of the expert trajectories used for training the reward models reached both the desired lane and velocity, which implies that these tasks are intrinsically challenging, with success in achieving these two goals being dependent on the recorded trajectory's length. Hence, it is understandable that the learned models encounter similar difficulties in accomplishing these tasks.

Lastly, Fig. 6.8e and Fig. 6.8f depict the percentage of invalid and colliding trajectories, respectively. Both nonlinear models outperformed the linear model for both metrics, with the larger nonlinear model slightly outperforming the smaller one. The larger model achieved a minimum number of invalid trajectories of $0.91\,\%$ after 220 steps versus $1.03\,\%$ after 340 steps for the smaller model, concerning the number of colliding trajectories, the trend is consistent with $0.91\,\%$ after 150 steps versus $1.26\,\%$ after 190 steps for the smaller model.

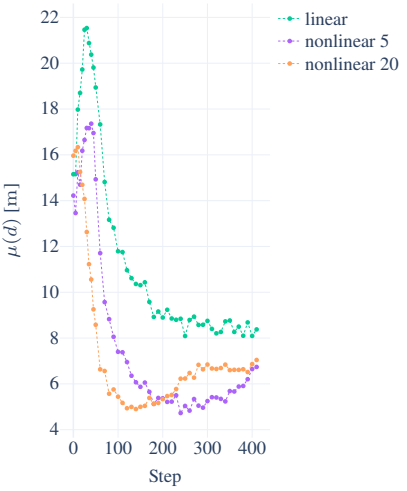
6.4.1 Summary

In conclusion, the objective of converging toward the collected expert behavior was fulfilled by all models. The results of the linear model being outperformed by the nonlinear ones and the larger nonlinear performing better than, the smaller ones indicated that nonlinear models with larger capacity are better suited for the task than smaller ones. The visual resemblance of the generated samples to

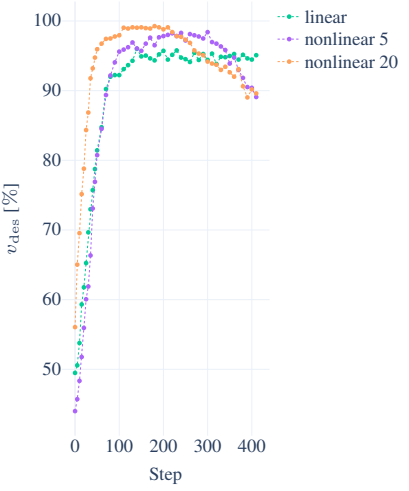
the expert trajectories by the larger nonlinear reward model can be assessed in Fig. 6.10.



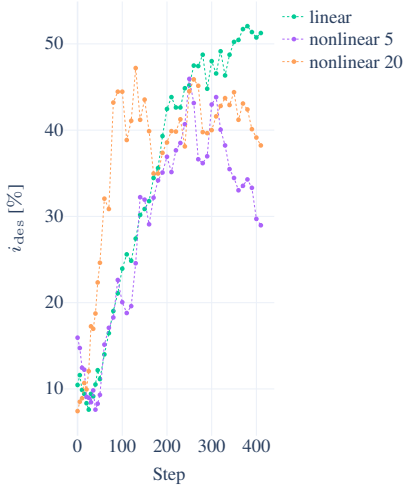
(a) Relative return difference between the expert trajectories $\mathcal{T}_{\mathrm{E}}$ and the sample trajectories $\mathcal{T}_{\mathrm{S}}$



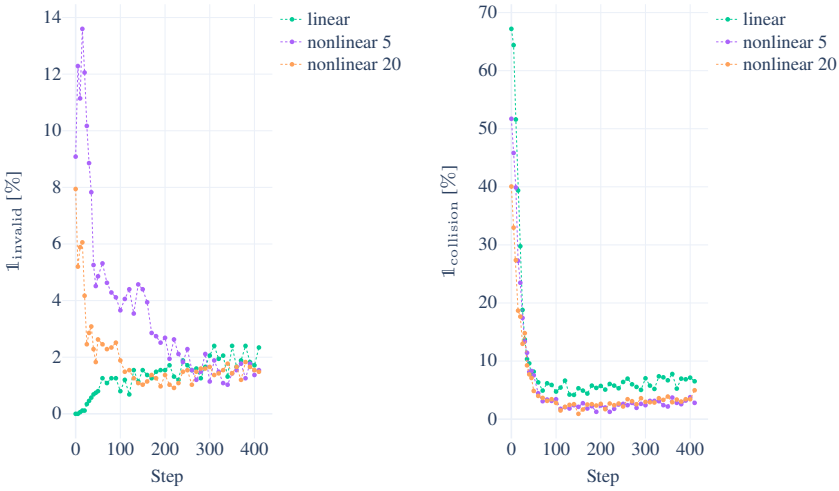
(b) Mean Euclidean distance of $\mathcal{T}_{\mathrm{S}}$ to the k-nearest neighbors ($k = 3$) expert trajectories $\mathcal{T}_{\mathrm{E}}$



(c) Fraction of trajectories reaching the agent's desired lane



(d) Fraction of trajectories reaching the agent's desired velocity



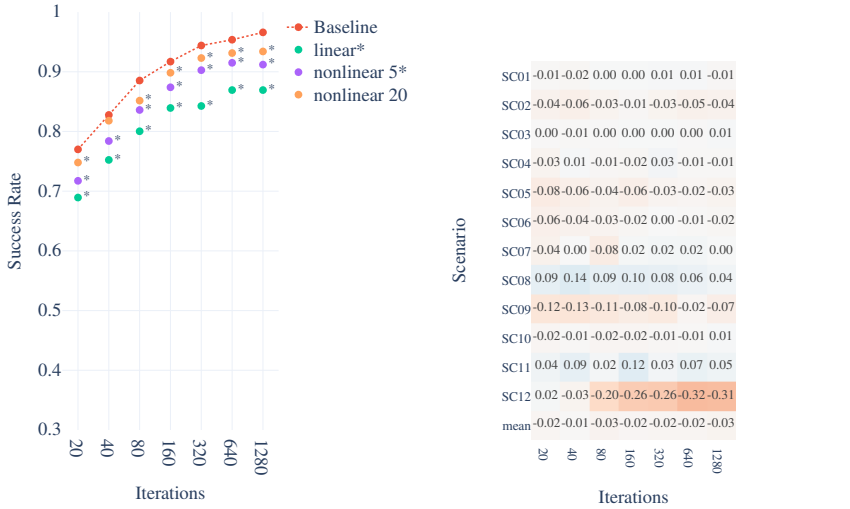
(e) Fraction of trajectories with an invalid state or action

(f) Fraction of trajectories with a collision

Figure 6.8: Inverse Reinforcement Learning Model Comparison: Comparison of the reward models throughout the training with respect to different metrics

An evaluation of the performance of the learned models compared to the manually tuned baseline is visualized in Fig. 6.9. Similarly to the IRL performance, the larger nonlinear model outperformed the linear and smaller nonlinear model, cf. Fig. 6.9a. For all but 40 iterations, the larger nonlinear model performs significantly worse than the baseline. However, as Fig. 6.9b depicts, the performance decrease was small and can be largely attributed to the performance decrease of scenario 12.

The previous indicated the general feasibility of learning reward functions from expert behavior, even for highly interactive trajectory planning tasks, making manual specification and tuning obsolete.



(a) Success rate comparison of baseline and learned reward models across iterations

(b) Absolute difference in success rate between baseline and nonlinear 20 across scenarios

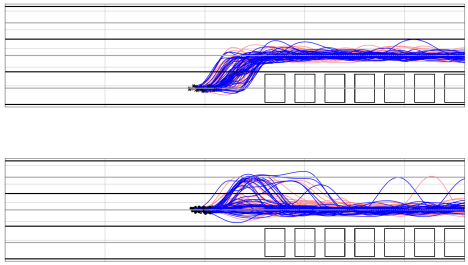
Figure 6.9: Performance comparison of learned reward models

(a) Scenario 01: Delaying merge due to approaching vehicle in desired lane

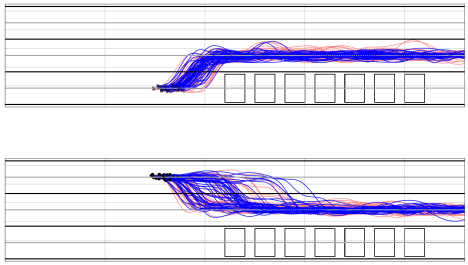
(b) Scenario 02: Reacting to approaching vehicle

(c) Scenario 03: Merging into moving traffic

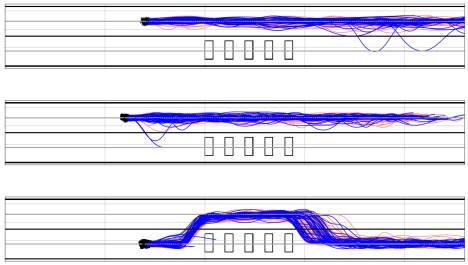
(d) Scenario 04: Merging into moving traffic with prior longitudinal adjustment



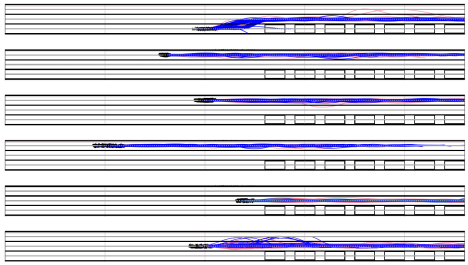
(e) Scenario 05: Changing lane as other vehicle needs to merge onto lane



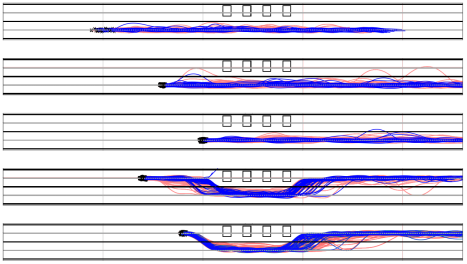
(f) Scenario 06: Delaying lane change as other vehicle needs to merge first



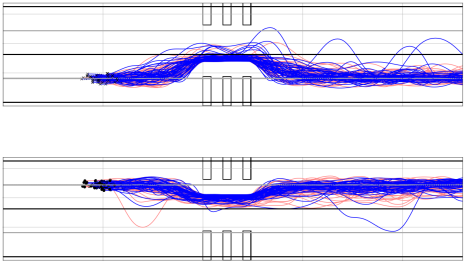
(g) Scenario 07: Merging (1 vehicle) with 3 Vehicles



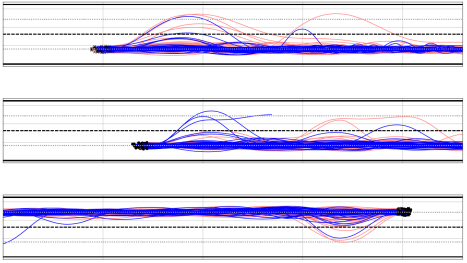
(h) Scenario 08: Merging (1 vehicle) with 5 Vehicles



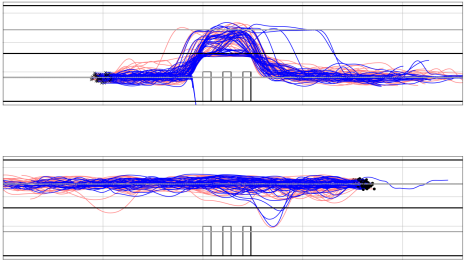
(i) Scenario 09: Merging (2 vehicles) with 5 Vehicles



(j) Scenario 10: Merging into middle lane (3 lanes)



(k) Scenario 11: Merging into moving traffic



(l) Scenario 12: Passing a narrow passage with oncoming traffic

Figure 6.10: Sample Trajectories from the Learned Reward Model: The expert trajectories (red) are used to learn the parameters of the reward model, which generates the optimal trajectories (blue, produced by the best performing nonlinear reward model, nonlinear 20). To enhance comprehension, the trajectories are depicted per agent for each scenario, cf. Fig. 6.1.

6.5 Parallelization

Each parallelization technique was evaluated for different numbers of iterations and threads. Due to the high computational complexity of the parallelized implementation, the evaluation was limited to the most challenging scenarios, SC07 - SC15, and the lower iteration counts of 20 - 320.

Based on nine scenarios, five different numbers of iterations, and three different numbers of threads, 33750 evaluations for each scatter plot were produced. In addition, the influence of the HP γ used in the similarity merge and similarity vote of the root parallelization was assessed. All parallelization strategies were compared to the performance of the average single-threaded baseline.

6.5.1 Leaf Parallelization

The evaluation of the leaf parallelization strategies indicated a clear benefit from parallelization. Both the mean aggregation, Fig. 6.11a, and the max aggregation, Fig. 6.11c outperformed the baseline significantly for all numbers of threads and iterations.

As mentioned, it is reasonable that the max aggregation strategy outperforms the mean aggregation in cooperative scenarios since its estimate of the state-action value is less pessimistic (Soemers et al. 2016). This was visible when comparing the performance gains from different numbers of threads. While there was a visible trend for the max aggregation, Fig. 6.11c, (an increasing number of threads increases performance), the difference was less pronounced for the mean aggregation, Fig. 6.11a. For example, in scenario 12 and scenario 13, the navigable space is extremely limited with the additional challenge of oncoming traffic. In these scenarios, most actions, such as a collision or driving off the road, will lead to an undesirable state. The mean aggregation of the parallelization had trouble identifying the limited number of desirable actions, as it averaged the returns of all threads, Fig. 6.11b. Since the max aggregation was optimistic,

choosing the max of the returns of all threads made it much more likely to find one of the desirable actions, yielding a higher success rate, Fig. 6.11c.

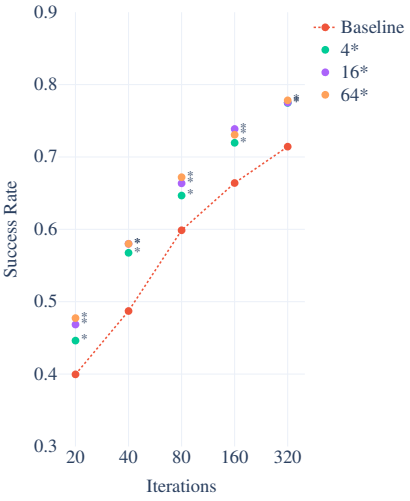
6.5.2 Root Parallelization

The evaluation of root parallelization, similar to leaf parallelization, showed the advantages of parallelization. However, similarity merge and similarity vote use two distinct methods to incorporate knowledge from several search trees using root parallelization.

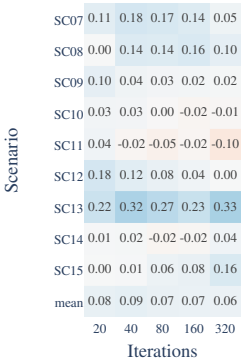
The similarity merge method considers all actions from all previously simulated trees, while the similarity vote method only considers the actions with the maximum state-action value of each tree. Hence, the size of the similarity matrix and thus the number of similarity updates grows quadratically with the effective branching factor (the number of actions explored from a given node) when the similarity merge approach is used.

Given the computational complexity of the similarity merge method, $\mathrm{O}\left((|\mathcal{A}||\Xi|)^2\right)$ as detailed earlier, experiments involving 64 threads proved infeasible. These simulations were prematurely terminated due to an inability to progress, likely due to the exponential increase in computational demands. As a result, all results presented in Fig. 6.12 are based solely on experiments utilizing 4 and 16 threads.

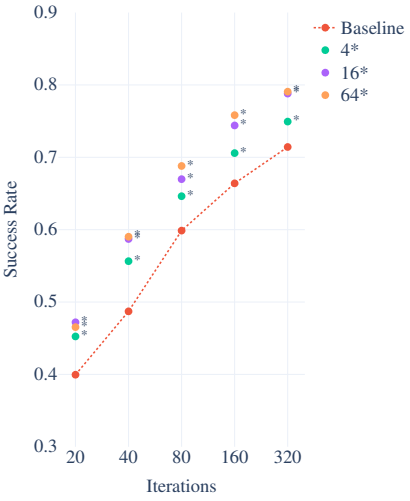
Figure 6.12 visualizes diminishing returns for an increase in the number of iterations and threads for similarity merge, with 160 and 320 iterations performing worse than 4 threads and the single-threaded baseline for $\gamma = 2.5$ respectively, Fig. 6.12a. This decay in performance can likely be attributed to the combination of kernel width for the similarity update and the increase in the number of similarity updates (as the number of iterations increases sufficiently progressive widening starts to take place, increasing the effective branching factor). If the kernel width is large, many dissimilar actions influence the evaluation of an action. With progressive widening adding actions in the vicinity of others, the likelihood



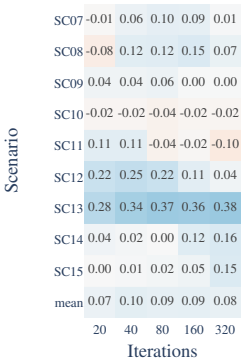
(a) Success rate comparison of baseline and mean aggregation for differing numbers of threads across iterations



(b) Absolute difference in success rate between baseline and 64 threads with mean aggregation across scenarios



(c) Success rate comparison of baseline and max aggregation for differing numbers of threads across iterations



(d) Absolute difference in success rate between baseline and 64 threads with max aggregation across scenarios

Figure 6.11: Performance comparison of aggregation strategies for leaf parallelization

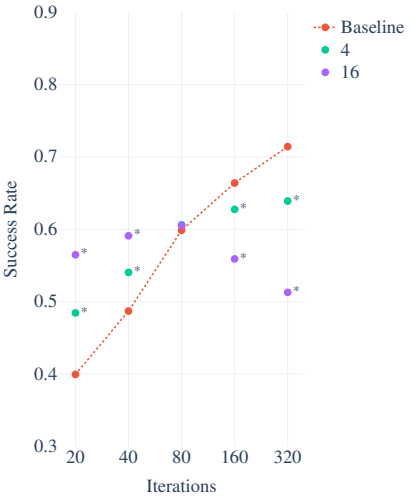
of two dissimilar actions influencing each other thus increases with an increasing number of iterations and threads.

A possible remedy could be a dynamic kernel width inversely proportional to the number of iterations and threads or a higher static kernel width as the results in Fig. 6.12c suggest. However, due to time constraints, only a few static values for γ were evaluated.

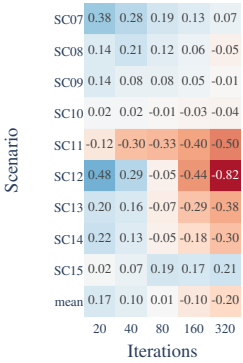
While similarity vote also suffered from diminishing returns for an increase in the number of iterations and threads, it performed consistently better than similarity merge, Fig. 6.13 and it was observed to be less sensitive to kernel width. However, a counterintuitive result was observed in that the performance for 4 threads decreased compared to the baseline for 160 and 320 iterations, while performance significantly improved for a higher number of threads. Further investigation is needed to clarify the underlying reasons for this behavior.

6.5.3 Summary

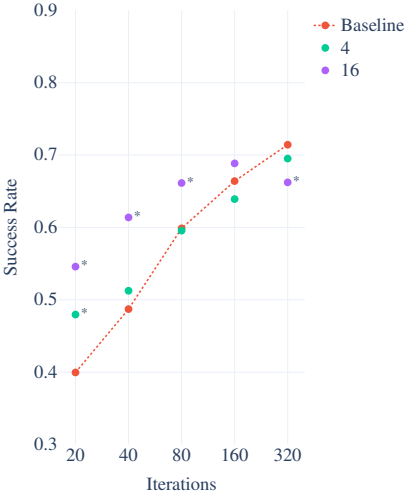
In conclusion, various methods to parallelize MCTS were developed and evaluated. All of them show improvements compared to the single-threaded baseline. However, the evaluation results indicated that root parallelization with similarity vote outperformed all other methods by a significant margin.



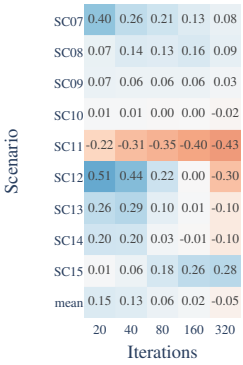
(a) Success rate comparison of baseline and similarity merge with $\gamma = 2.5$ across iterations



(b) Absolute difference in success rate between baseline and similarity merge with $\gamma = 2.5$ and 16 threads across scenarios

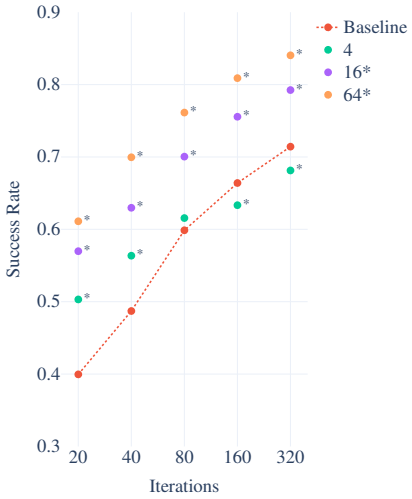


(c) Success rate comparison of baseline and similarity merge with $\gamma = 7.5$ across iterations

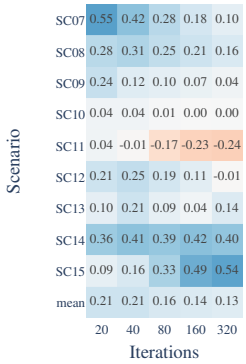


(d) Absolute difference in success rate between baseline and similarity merge with $\gamma = 7.5$ and 16 threads across scenarios

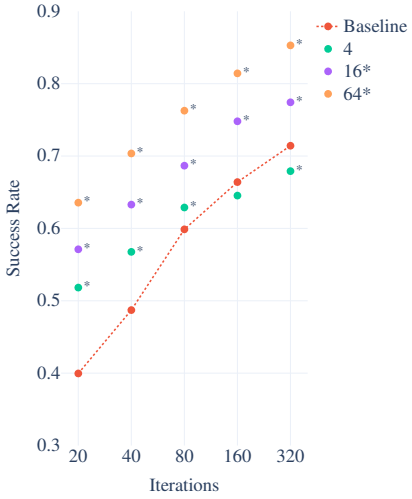
Figure 6.12: Performance comparison of kernel size for root parallelization using similarity merge



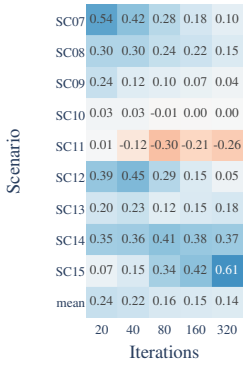
(a) Success rate comparison of baseline and similarity vote with $\gamma = 0.5$ across iterations



(b) Absolute difference in success rate between baseline and similarity vote with $\gamma = 0.5$ and 64 threads across scenarios



(c) Success rate comparison of baseline and similarity vote with $\gamma = 1.5$ across iterations



(d) Absolute difference in success rate between baseline and similarity vote with $\gamma = 1.5$ and 64 threads across scenarios

Figure 6.13: Performance comparison of kernel size for root parallelization using similarity vote

6.6 Hyperparameter Optimization

In order to identify the most promising HPO technique, model-free RS and model-based BO were contrasted based on the approach described in Section 5.3.4, to tune nine HPs.

Scenarios where the baseline already achieved a performance close to 1 (SC01-SC06) were excluded from the HPO, and only the subset of scenarios SC7 to SC15 was considered. By excluding these scenarios, a more objective evaluation of the performance of HPO technique could be provided.

In line with previous experiments, 250 random seeds were used to generate statistically accurate estimates for the success rate for each evaluated HPC. However, due to the runtime complexity of the optimization, the number of iterations within the MCTS was limited to 20, 80, and 320. Since the optimization was formulated as a minimization problem (1- success rate), the optimal value is 0.

The performance comparison results are presented in two figures: Fig. 6.14a and Fig. 6.14b, with the former displaying the performance distribution of the evaluated HPCs, while the latter depicts their performance for 100 optimization steps. The performance distribution comparison suggests that the RS is unimodal, comparable to a normal distribution, whereas BO exhibits two distinct modes, implying a bi-modal distribution. This observation is sensible since RS randomly explores the space of possible HPCs, leading to a more uniform coverage. On the other hand, a bi-modal distribution of BO was expected as it uses a surrogate model (RF) to approximate the objective function and guide the search through expected improvement, explicitly focusing on promising regions of the HP space. Bayesian Optimization outperformed RS with a mean of 0.513 versus 0.645 (1- success rate) for all evaluated HPCs, implying that it explored better performing HPC on average. Figure 6.14b further highlights this difference, with BO significantly outperforming RS over the course of the optimization.

Additionally, BO was executed for 1000 steps, which took approximately 188 hours. The results were compared to the manually specified baseline and are

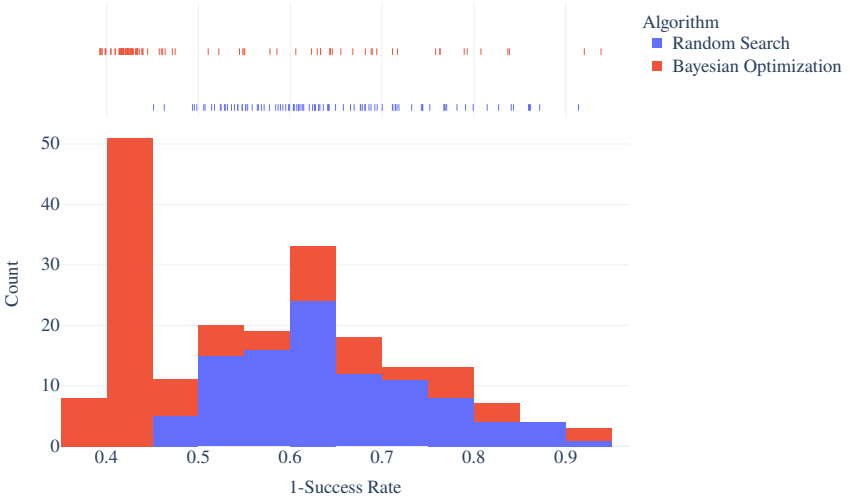
Parameter	Value
action duration	2.2203
maximum search depth	5
discount factor	0.9896
final selection strategy	MaxActionValue
action execution fraction	0.9370
progressive widening-coefficient	4.9696
progressive widening-exponent	0.8281
maximum depth for progressive widening	5
UCT-coefficient	0.3059

Table 6.1: Incumbent Parameters of the Vanilla Baseline

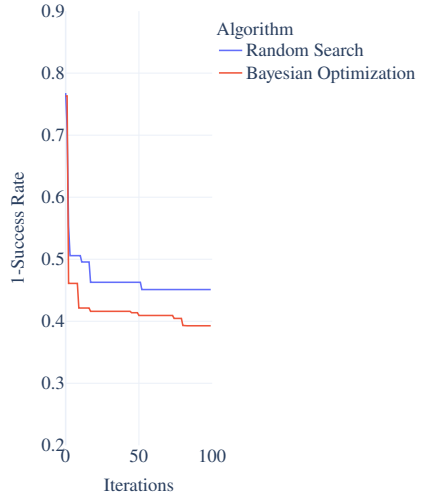
shown in Fig. 6.15. The plot demonstrates that the automatically tuned HPC is significantly superior to the manually specified baseline for all evaluated numbers of iterations. Notably, the performance gains plateau at around 640 iterations, which may be attributed to the optimization being limited to iterations smaller than 320.

Based on the output of the HPO, the marginal contribution of a parameter can be estimated using functional Analysis of Variance (fANOVA), (Hutter et al. 2014). For this, an RF was fitted to connect the objective function to the HPCs. The influence of a parameter can be estimated by altering a HP while marginalizing over all other HPs for each HP. The marginals for the action duration and maximum search depth are displayed in Fig. 6.16. From the plots, one can infer that the values of both HPs impact the algorithm's performance, and the directionality of the correlation can be deduced. As can be seen, the cost (1-success rate) decreases for larger search depths and shorter action durations.

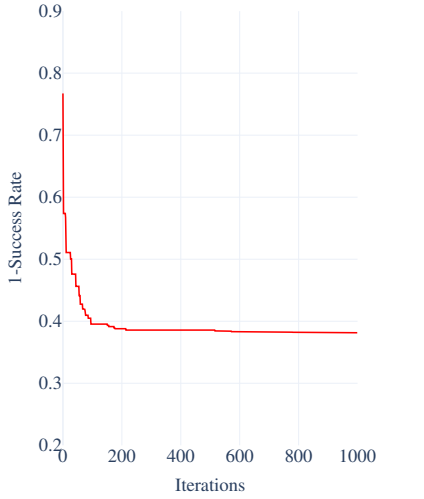
The HPC for the vanilla baseline used for most experiments, and the HPC for action grouping were generated using BO. The resulting values can be found in Table 6.1 and Table 6.2, respectively.



(a) Comparison of RS and BO through their performance distributions



(b) Optimization performance comparison of RS and BO over 100 optimization steps

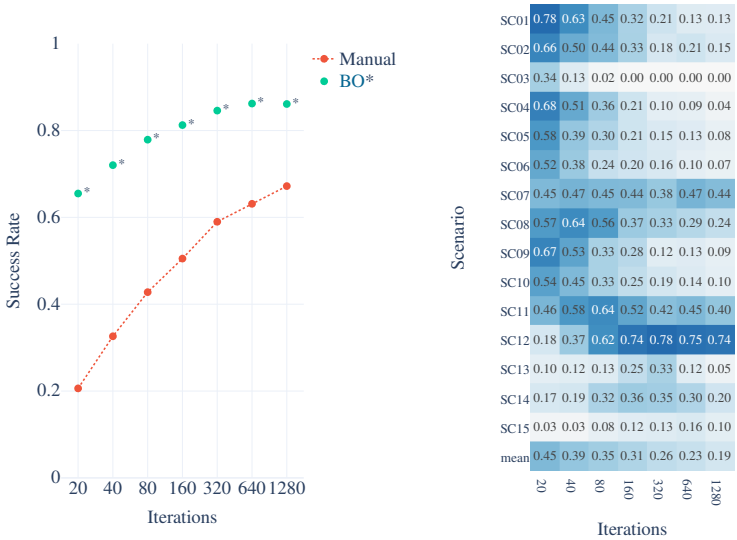


(c) Optimization performance of BO over 1000 optimization steps

Figure 6.14: Comparison of Hyperparameter Optimization: Random Search vs. Bayesian Optimization

Parameter	Value
UCT-coefficient within action grouping	0.4618
final selection strategy based on action grouping	False
progressive widening-active within action grouping	True
maximum depth for progressive widening within action grouping	5
progressive widening-coefficient within action grouping	1.2793
progressive widening-exponent within action grouping	0.6415

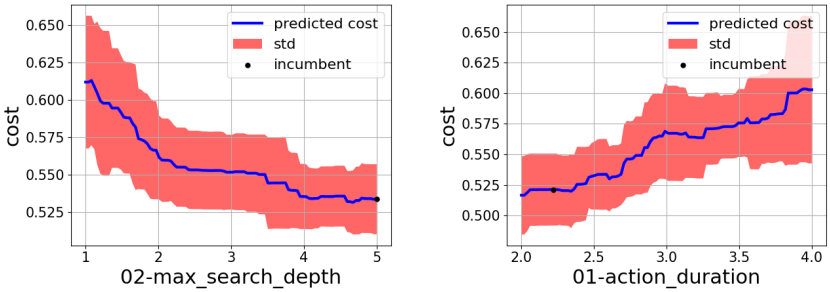
Table 6.2: Incumbent Parameters of Action Grouping



(a) Success rate comparison of baseline and automatic HP tuning across iterations

(b) Absolute difference in success rate between baseline and Bayesian Optimization across scenarios

Figure 6.15: Performance comparison of Hyperparameter Optimization



(a) Marginal of maximum search depth

(b) Marginal of action duration

Figure 6.16: Marginals of Two Hyperparameters

6.6.1 Summary

This section focused on identifying the best HPO technique by comparing model-free RS and model-based BO for tuning HPs. In conclusion, BO outperformed RS, demonstrating better exploration of high-performing hyperparameter configurations on average. Furthermore, when executed for 1000 steps, automatically tuned HPCs significantly outperformed the manually specified baseline.

7 Summary

The following provides a concise summary of this thesis's central research questions and findings, outlining its contributions to the field of AD.

This work started with a first research question on how cooperative driving without communication could be modeled.

It proposes a method for planning cooperative trajectories in challenging interactive urban scenarios and tight spaces. The framework is based on a decentralized continuous MCTS. This approach adapts and extends the original MCTS algorithm to handle simultaneous move games and continuous state and action spaces. Key extensions in this framework include DUCT for addressing simultaneous decision-making and progressive widening to address continuous action spaces by gradually expanding the action space of a node. The framework also employs heuristics such as blind value to guide the sampling process toward more promising regions of the action space. While efforts were made to reduce computational complexity using action grouping and semantic action groups, no significant improvements were observed. Nevertheless, the evaluation of the core algorithm demonstrated its feasibility across a wide range of scenarios.

The second research question aimed to effectively and efficiently address the modeled problem. More specific inquiries were formulated based on the algorithm's requirements mentioned in Section 4.2.

First, it was investigated how a reward function can be designed so that the resulting behavior represents human driving. A reward function can be designed to result in human-like driving behavior through IRL. This work combines Maximum Entropy IRL with MCTS to learn reward models for the cooperative trajectory

planning problem. The efficacy of MCTS to generate (approximately) optimal samples for arbitrary reward models quickly in combination with adjusting the sampling distribution after gradient updates yield reward models that quickly converge towards the experts. Evaluations showed that the performance of the learned (linear and nonlinear) reward models was comparable to or better than the manually tuned baseline model, demonstrating the effectiveness of this approach for designing reward functions that represent human driving behavior in cooperative environments with implicit communication.

Next, it was explored how parallel computation can be leveraged to accelerate convergence speed. Parallel computing can be used to accelerate the convergence speed of MCTS in continuous domains through various parallelization strategies such as leaf and root parallelization. Standard Leaf parallelization resulted in modest performance improvements. However, root parallelization, which involves the exploration of multiple trees from a single root, can considerably improve performance when combined with a novel way to merge results using a similarity voting mechanism. Although the improvements were substantial, future research should focus on developing more efficient parallelization techniques concerning the scalability of the number of threads.

Last, it was studied how HPO can be employed to improve the solution quality. Using HPO mitigates the need for manual selection and fine-tuning of HPs, which quickly becomes convoluted in high-dimensional search spaces with complicated interactions for humans to reason about. Model-based optimization techniques like BO allowed for a more targeted exploration of the solution space (as opposed to model-free optimization), resulting in improved HPCs and tremendous increases in the algorithm's overall performance. Hence, incorporating HPO approaches in the cooperative trajectory planning algorithm proved essential since it significantly enhances the solution quality and maximizes the efficiency of action search.

7.1 Outlook

While the current research constitutes a substantial advancement in the capacity of AVs for implicit cooperation with other traffic participants, various research opportunities remain unexplored to fully realize the potential of AD.

A major challenge in AD is ensuring safety, particularly in complex and dynamic situations involving multiple implicitly interacting traffic participants. Addressing this challenge necessitates additional research for safeguarding all road users.

Enhancing the accuracy and robustness of planning algorithms is equally crucial. Therefore, prospective studies should tackle partial observability e.g. by incorporating approaches proposed by Stegmaier et al. (Stegmaier et al. 2022) and expanding the investigation to cover a broader spectrum of scenarios.

Moreover, there is immense potential for accelerating the search process through the implementation of learned heuristics, as suggested by Kurzer et al. (Kurzer et al. 2021), and by estimating the state of other traffic participants utilizing methodologies illustrated by Tian et al. (Tian et al. 2021) and Sunberg et al. (Sunberg et al. 2017).

By delving into and addressing these challenges, the research community can move progressively towards the full realization of AVs potential, ultimately contributing to improved transportation experiences for all.

A Appendix

A.1 Design and Implementation of the ProSeCo Planning Package

The ProSeCo Planning package consists of a C++ (online) as well as Python (offline) part. The C++ part is comprised of a library as well as a ROS interface (Quigley et al. 2009). The library includes the runtime critical components, such as the MCTS, the extensions, and the simulator. The Python part is comprised of four different modules, namely the Evaluator, the Dashboard, the Hyperparameter Optimization, and the Inverse Reinforcement Learning, see Fig. A.1.

The source code of the ProSeCo Planning package is openly accessible on GitHub (https://github.com/ProSeCo-Planning) under the BSD 3-Clause License. The runtime environment required to execute the ProSeCo Planning package is available as a container on dockerhub (https://hub.docker.com/r/karlkurzer/proseco).

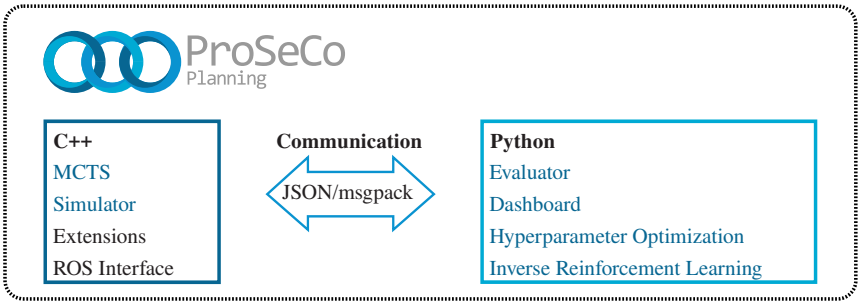


Figure A.1: Overview of the ProSeCo Planning Package with Its C++ and Python Part

A.1.1 Simulator

The simulator performs the task of multi-agent vehicle simulation. It has been designed to simulate interactions among agents, specifically vehicles in this case. The simulator's architecture encompasses several core components, including the environment model of the road, vehicles, as well as obstacles. Different vehicle profiles can be modeled, each with characteristics such as size, wheelbase, maximum acceleration, and maximum steering angle. Beyond environment and vehicle representation, the simulator conducts trajectory generation and collision checking.

Despite the complexity of these operations, the simulator is highly efficient and capable of conducting approximately $300\,000$ collision checks$/\mathrm{s}$. This enables the algorithm to explore a large action space in a short amount of time, which is important to facilitate fast and accurate decision-making in complex multi-agent systems.

A.1.2 Evaluator

The evaluator module is a crucial component of the ProSeCo Planning package, which facilitates the parallel and distributed evaluation of different algorithm and scenario configurations. It operates on a Ray cluster, an open-source software system designed explicitly for scalable distributed computing (Moritz et al. 2018). The configurations for the evaluator module are created using JavaScript Object Notation (JSON), offering a structured, readable format for setting up complex evaluation tasks.

An exemplary configuration is provided in Code 1. This configuration, for instance, would yield 81 different evaluations by combining various scenarios, discount factors, iteration numbers, and random seeds.

This level of configurability allows for comprehensive testing and comparison of different extensions over a wide range of scenarios.

```
{
        "name": "example_evaluation",
        "options": ["example_options"],
        "scenarios": ["sc01", "sc02", "sc03"],
        "options_alterations": {
                "compute_options": {
                        "discount_factor": [0.7, 0.8, 0.9],
                        "n_iterations": [500, 1000, 2000],
                        "random_seed": [331, 650, 28]
                }
        },
        "cluster": {
                "address": "192.168.1.100",
                "max_workers": 100
        }
}
```

Code 1: Evaluator Configuration: Exemplary configuration of the evaluator module that would yield 81 different evaluations (scenarios × discount factors × iteration numbers × random seeds)

Similarly, the ProSeCo Planning package allows users to configure various options and scenarios, as seen in Code 2 and Code 3. Options refer to the settings related to the algorithm. Scenarios, on the other hand, define the environmental setup for simulations. This includes the number and type of vehicles, road configuration, and obstacles in the simulation.

A.1.3 Dashboard

The dashboard module provides a visual interface for examining the evaluations generated by the evaluator. It enables users to understand the performance of various configurations at a glance. Through the dashboard, researchers can track performance metrics over time, compare different configurations side by side, and visually inspect individual runs to diagnose issues or understand unusual results. An example of the dashboard view is provided in Fig. A.2.

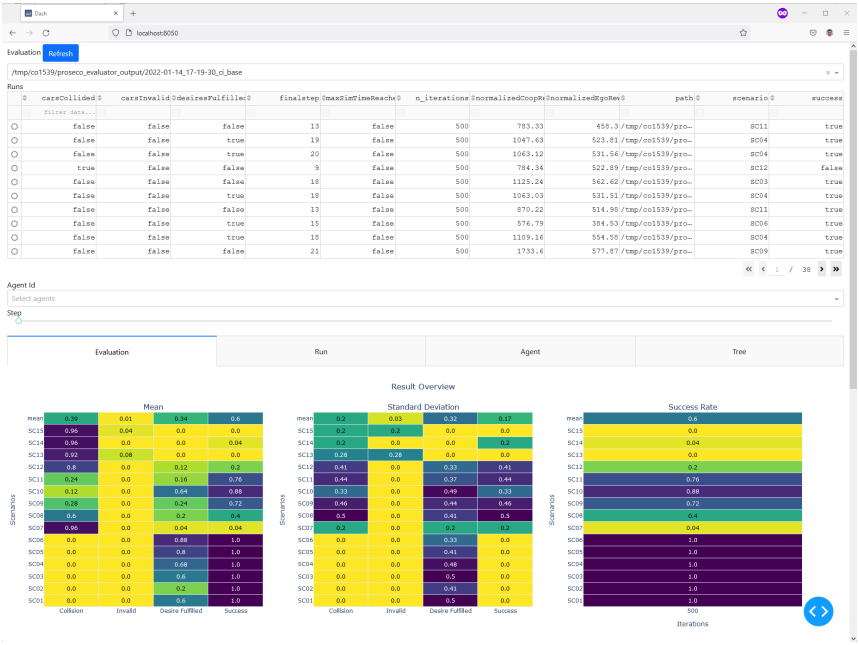


Figure A.2: A Screenshot of the ProSeCo Planning Dashboard

```
"compute_options": {
        "action_duration": 2.22,
        "action_noise": {
                "active": false,
                "mean_vx": 0.0,
                "mean_y": 0.0,
                "sigma_vx": 0.1,
                "sigma_y": 0.1
        },
        "collision_checker": "circleApproximation",
        "delta_t": 0.1,
        "discount_factor": 0.9896,
        "end_condition": "scenario",
        "max_invalid_action_samples": 25,
        "max_scenario_duration": 0.0,
        "max_scenario_steps": 40,
        "max_search_depth": 5,
        "max_step_duration": 0.0,
        "n_iterations": 640,
        "noise": {
                "active": false,
                "mean": 0.0,
                "sigma": 0.15
        },
        "parallelization_options": {
                "n_simulationThreads": 16,
                "n_threads": 16,
                "similarity_gamma": 1.5,
                "similarity_voting": true,
                "simulation_aggregation": "max"
        },
        "policy_options": {
                "expansion_policy": "UCT",
                "final_selection_policy": "maxActionValue",
                "policy_enhancements": {
                        "action_execution_fraction": 0.9370,
                        "available_action_type": "random",
                        "move_grouping": {
                                "active": false,
                                "cp": 0.4618,
```

```
                                "final_decision": false,
                                "move_grouping_bias_pw": false,
                                "move_grouping_criteria_pw": {
                                        "active": false,
                                        "coefficient_pw": 1.2793,
                                        "exponent_pw": 0.6415
                                }
                        },
                        "progressive_widening": {
                                "coefficient": 4.9697,
                                "exponent": 0.8281,
                                "max_depth_pw": 5
                        },
                        "q_scale": 100.0,
                        "search_guide": {
                                "n_samples": 51,
                                "type": "blind_value"
                        },
                        "similarity_update": {
                                "active": false,
                                "gamma": 1.0
                        }
                },
                "selection_policy": "UCTProgressiveWidening",
                "simulation_policy": "random",
                "update_policy": "UCT"
        },
        "random_seed": 0,
        "safety_distance": 0.0,
        "trajectory_type": "jerkOptimal",
        "uct_cp": 0.3059
},
"output_options": {
        "export": ["result"],
        "export_format": "msgpack"
}
```


Code 2: Exemplary Configuration of the Cooperative Trajectory Planning Algorithm

```
{
        "name": "SC07",
        "road": {
                "lane_width": 3.25,
                "number_lanes": 2,
                "random": false,
                "sigma_lane_width": 0.25
        },
        "agents": [
        {
                "action_space": {
                        "delta_velocity": 1.6666666269302368,
                        "max_lateral_change": 5.0,
                        "max_velocity_change": 5.0,
                        "type": "rectangle"
                },
                "cooperation_factor": 0.5,
                "cost_model": {
                        "cost_collision": -1000.0,
                        "cost_invalid_action": 0.0,
                        "cost_invalid_state": -1000.0,
                        "name": "costExponential",
                        "w_acceleration_x": 0.0,
                        "w_acceleration_y": -5.0,
                        "w_lane_center_deviation": 85.0,
                        "w_lane_change": -10.0,
                        "w_lane_deviation": 100.0,
                        "w_velocity_deviation": 500.0
                },
                "desire": {
                        "lane": 1,
                        "lane_center_tolerance": 1.0,
                        "velocity": 8.0,
                        "velocity_tolerance": 2.0
                },
                "id": 0,
                "is_predefined": false,
                "terminal_condition": {
                        "comparator_position_x": "larger",
                        "comparator_position_y": "none",
```

```
                "position_x": 125.0,
                "position_y": 0.0
        },
        "vehicle": {
                "heading": 0.0,
                "length": 4.709000110626221,
                "max_acceleration": 9.807000160217285,
                "max_speed": 36.0,
                "max_steering_angle": 0.2630000114440918,
                "position_x": 19.844329833984375,
                "position_y": 4.545746326446533,
                "random": true,
                "sigma_heading": 0.0,
                "sigma_length": 0.0,
                "sigma_position_x": 1.0,
                "sigma_position_y": 0.20000000298023224,
                "sigma_velocity_x": 0.0,
                "sigma_velocity_y": 0.0,
                "sigma_width": 0.0,
                "velocity_x": 8.0,
                "velocity_y": 0.0,
                "wheel_base": 2.8510000705718994,
                "width": 1.8270000219345093
        }
},
{
        "action_space": {
                "delta_velocity": 1.6666666269302368,
                "max_lateral_change": 5.0,
                "max_velocity_change": 5.0,
                "type": "rectangle"
        },
        "cooperation_factor": 0.5,
        "cost_model": {
                "cost_collision": -1000.0,
                "cost_invalid_action": 0.0,
                "cost_invalid_state": -1000.0,
                "name": "costExponential",
                "w_acceleration_x": 0.0,
                "w_acceleration_y": -5.0,
                "w_lane_center_deviation": 85.0,
```

```
                "w_lane_change": -10.0,
                "w_lane_deviation": 100.0,
                "w_velocity_deviation": 500.0
        },
        "desire": {
                "lane": 1,
                "lane_center_tolerance": 1.0,
                "velocity": 8.0,
                "velocity_tolerance": 2.0
        },
        "id": 1,
        "is_predefined": false,
        "terminal_condition": {
                "comparator_position_x": "larger",
                "comparator_position_y": "none",
                "position_x": 125.0,
                "position_y": 0.0
        },
        "vehicle": {
                "heading": 0.0,
                "length": 4.709000110626221,
                "max_acceleration": 9.807000160217285,
                "max_speed": 36.0,
                "max_steering_angle": 0.2630000114440918,
                "position_x": 10.891773223876953,
                "position_y": 5.0637006759643555,
                "random": true,
                "sigma_heading": 0.0,
                "sigma_length": 0.0,
                "sigma_position_x": 1.0,
                "sigma_position_y": 0.20000000298023224,
                "sigma_velocity_x": 0.0,
                "sigma_velocity_y": 0.0,
                "sigma_width": 0.0,
                "velocity_x": 8.0,
                "velocity_y": 0.0,
                "wheel_base": 2.8510000705718994,
                "width": 1.8270000219345093
        }
},
{
```

```
"action_space": {
        "delta_velocity": 1.6666666269302368,
        "max_lateral_change": 5.0,
        "max_velocity_change": 5.0,
        "type": "rectangle"
},
"cooperation_factor": 0.5,
"cost_model": {
        "cost_collision": -1000.0,
        "cost_invalid_action": 0.0,
        "cost_invalid_state": -1000.0,
        "name": "costExponential",
        "w_acceleration_x": 0.0,
        "w_acceleration_y": -5.0,
        "w_lane_center_deviation": 85.0,
        "w_lane_change": -10.0,
        "w_lane_deviation": 100.0,
        "w_velocity_deviation": 500.0
},
"desire": {
        "lane": 0,
        "lane_center_tolerance": 1.0,
        "velocity": 8.0,
        "velocity_tolerance": 2.0
},
"id": 2,
"is_predefined": false,
"terminal_condition": {
        "comparator_position_x": "larger",
        "comparator_position_y": "none",
        "position_x": 125.0,
        "position_y": 0.0
},
"vehicle": {
        "heading": 0.0,
        "length": 4.709000110626221,
        "max_acceleration": 9.807000160217285,
        "max_speed": 36.0,
        "max_steering_angle": 0.2630000114440918,
        "position_x": 20.801456451416016,
        "position_y": 1.466967225074768,
```

```
                "random": true,
                "sigma_heading": 0.0,
                "sigma_length": 0.0,
                "sigma_position_x": 1.2999999523162842,
                "sigma_position_y": 0.20000000298023224,
                "sigma_velocity_x": 0.0,
                "sigma_velocity_y": 0.0,
                "sigma_width": 0.0,
                "velocity_x": 8.0,
                "velocity_y": 0.0,
                "wheel_base": 2.8510000705718994,
                "width": 1.8270000219345093
        }
}
],
"obstacles": [
{
        "heading": 0.0,
        "id": 0,
        "length": 4.0,
        "position_x": 50.0,
        "position_y": 1.75,
        "random": false,
        "sigma_heading": 0.0,
        "sigma_length": 0.0,
        "sigma_position_x": 0.0,
        "sigma_position_y": 0.0,
        "sigma_width": 0.0,
        "width": 2.0
},
{
        "heading": 0.0,
        "id": 1,
        "length": 4.0,
        "position_x": 60.0,
        "position_y": 1.75,
        "random": false,
        "sigma_heading": 0.0,
        "sigma_length": 0.0,
        "sigma_position_x": 0.0,
        "sigma_position_y": 0.0,
```

```
            "sigma_width": 0.0,
            "width": 2.0
        },
        {
            "heading": 0.0,
            "id": 2,
            "length": 4.0,
            "position_x": 70.0,
            "position_y": 1.75,
            "random": false,
            "sigma_heading": 0.0,
            "sigma_length": 0.0,
            "sigma_position_x": 0.0,
            "sigma_position_y": 0.0,
            "sigma_width": 0.0,
            "width": 2.0
        },
        {
            "heading": 0.0,
            "id": 3,
            "length": 4.0,
            "position_x": 80.0,
            "position_y": 1.75,
            "random": false,
            "sigma_heading": 0.0,
            "sigma_length": 0.0,
            "sigma_position_x": 0.0,
            "sigma_position_y": 0.0,
            "sigma_width": 0.0,
            "width": 2.0
        },
        {
            "heading": 0.0,
            "id": 4,
            "length": 4.0,
            "position_x": 90.0,
            "position_y": 1.75,
            "random": false,
            "sigma_heading": 0.0,
            "sigma_length": 0.0,
            "sigma_position_x": 0.0,
```

```
                "sigma_position_y": 0.0,
                "sigma_width": 0.0,
                "width": 2.0
        }
        ]
}
```

Code 3: Exemplary Configuration of a Scenario

List of Figures

List of Tables

List of Algorithms

List of Publications

Related Contributions

K. Kurzer, F. Engelhorn, and J. M. Zöllner. Decentralized cooperative planning for automated vehicles with continuous monte carlo tree search. In *Intelligent Transportation Systems Conference (ITSC)*. IEEE, 2018a. doi: 10.1109/ITSC.2018.8569988.

K. Kurzer, C. Zhou, and J. M. Zöllner. Decentralized cooperative planning for automated vehicles with hierarchical monte carlo tree search. In *Intelligent Vehicles Symposium (IV)*. IEEE, 2018b. doi: 10.1109/IVS.2018.8500712.

K. Kurzer, M. Fechner, and J. M. Zöllner. Accelerating cooperative planning for automated vehicles with learned heuristics and monte carlo tree search. In *Intelligent Vehicles Symposium (IV)*. IEEE, 2020a. doi: 10.1109/IV47402.2020.9304597.

K. Kurzer, C. Hörtnagl, and J. M. Zöllner. Parallelization of monte carlo tree search in continuous domains. arXiv, 2020b. doi: 10.48550/ARXIV.2003.13741.

K. Kurzer, M. Bitzer, and J. M. Zöllner. Learning reward models for cooperative trajectory planning with inverse reinforcement learning and monte carlo tree search. In *Intelligent Vehicles Symposium (IV)*. IEEE, 2022. doi: 10.1109/IV51971.2022.9827031.

P. Stegmaier, K. Kurzer, and M. J. Zöllner. Cooperative trajectory planning in uncertain environments with monte carlo tree search and risk metrics. In

Intelligent Transportation Systems Conference (ITSC). IEEE, 2022. doi: 10.1109/ITSC55140.2022.9921861.

Other Contributions

K. Kurzer, P. Schörner, A. Albers, H. Thomsen, K. Daaboul, and J. M. Zöllner. Generalizing decision making for automated driving with an invariant environment representation using deep reinforcement learning. In *Intelligent Vehicles Symposium (IV)*. IEEE, 2021. doi: 10.1109/IV48863.2021.9575669.

P. Wolf, K. Kurzer, T. Wingert, F. Kuhnt, and J. M. Zöllner. Adaptive behavior generation for autonomous driving using deep reinforcement learning with compact semantic states. In *Intelligent Vehicles Symposium (IV)*. IEEE, 2018. doi: 10.1109/IVS.2018.8500427.

Supervised Theses

M. Bitzer. Learning reward functions for cooperative driving with inverse reinforcement learning, 2020.

C. Buchert. Intersection navigation using reinforcement learning based on an invariant environment representation, 2018.

F. Engelhorn. Decentralized cooperative planning of autonomous vehicles on the basis of a continuous monte carlo tree search, 2018.

M. Fechner. Combining monte carlo tree search and deep neural networks to accelerate motion planning for cooperative driving in multiagent scenarios, 2020.

C. Gabrian. Quantifying the need and will to cooperate in traffic, 2022.

C. Hörtnagl. Parallelization of continuous monte carlo tree search for planning cooperative driving maneuvers, 2020.

M. Käser. Tree-reuse for the monte carlo tree search in continuous action spaces for autonomous cooperative driving, 2020.

T. Klein. Combining reinforcement learning and search for cooperative trajectory planning, 2021.

S. Müller. Continuous trajectory planning for automated driving through deep reinforcement learning, 2020.

P. Reboud. Hyperparameter optimization of expensive cost functions applied to trajectory planning for automated driving, 2020.

M. Ruchay. Semantic scene description and identification of cooperation aspects, 2017.

J. Schuck. Learning a maneuver based driving behavior with reinforcement learning and monte carlo tree search, 2019.

P. Stegmaier. Addressing uncertainties in cooperative trajectory planning using monte carlo tree search, 2021.

L. Thede. A comparative study on the capabilities of opensource autonomous driving frameworks, 2019.

T. Wingert. Learning a semantic action policy for autonomous driving, 2018.

C. Zhou. Decentralized planning of macroactions for cooperative automated vehicles with hierarchical monte carlo tree search, 2018.

Bibliography

P. Abbeel and A. Y. Ng. Apprenticeship learning via inverse reinforcement learning. In *International Conference on Machine Learning (ICML)*. Association for Computing Machinery, 2004. doi: 10.1145/1015330.1015430.

S. Arora and P. Doshi. A survey of inverse reinforcement learning: Challenges, methods and progress. *Artificial Intelligence*, 2021. doi: 10.1109/ICRA40945.2020.9196795.

R. Axelrod and W. Hamilton. The evolution of cooperation. *Science*, 1981. doi: 10.1126/science.7466396.

S. Bae, D. Saxena, A. Nakhaei, C. Choi, K. Fujimura, and S. Moura. Cooperation-aware lane change maneuver in dense traffic based on model predictive control with recurrent neural network. In *American Control Conference (ACC)*, 2020. doi: 10.23919/ACC45564.2020.9147837.

M. Bahram, C. Hubmann, A. Lawitzky, M. Aeberhard, and D. Wollherr. A combined model- and learning-based framework for interaction-aware maneuver prediction. *Transactions on Intelligent Transportation Systems*, 2016a. doi: 10.1109/TITS.2015.2506642.

M. Bahram, A. Lawitzky, J. Friedrichs, M. Aeberhard, and D. Wollherr. A game-theoretic approach to replanning-aware interactive scene prediction and planning. *Transactions on Vehicular Technology*, 2016b. doi: 10.1109/TVT.2015.2508009.

J. Bergstra and Y. Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 2012.

A. Bourki, G. Chaslot, M. Coulm, V. Danjean, H. Doghmen, J.-B. Hoock, T. Hérault, A. Rimmel, F. Teytaud, O. Teytaud, P. Vayssière, and Z. Yu. Scalability and parallelization of monte-carlo tree search. In *Computers and Games*. Springer, 2011.

C. Boutilier. Sequential optimality and coordination in multiagent systems. In *International Joint Conference on Artificial Intelligence (IJCAI)*. Morgan Kaufmann Publishers Inc., 1999.

M. Bouton, A. Cosgun, and M. J. Kochenderfer. Belief state planning for autonomously navigating urban intersections. In *Intelligent Vehicles Symposium (IV)*. IEEE, 2017. doi: 10.1109/IVS.2017.7995818.

M. Bouton, A. Nakhaei, K. Fujimura, and M. J. Kochenderfer. Cooperation-aware reinforcement learning for merging in dense traffic. In *Intelligent Transportation Systems Conference (ITSC)*. IEEE, 2019a. doi: 10.1109/ITSC.2019.8916924.

M. Bouton, A. Nakhaei, K. Fujimura, and M. J. Kochenderfer. Safe reinforcement learning with scene decomposition for navigating complex urban environments. In *Intelligent Vehicles Symposium (IV)*. IEEE, 2019b. doi: 10.1109/IVS.2019.8813803.

M. Bouton, A. Nakhaei, D. Isele, K. Fujimura, and M. J. Kochenderfer. Reinforcement learning with iterative reasoning for merging in dense traffic. In *Intelligent Transportation Systems Conference (ITSC)*. IEEE, 2020. doi: 10.1109/ITSC45102.2020.9294338.

B. Bouzy. Move-pruning techniques for monte-carlo go. In *Advances in Computer Games*. Springer, 2006.

E. Brochu, V. M. Cora, and N. de Freitas. A tutorial on bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning. arXiv, 2010. doi: 10.48550/ARXIV.1012.2599.

C. B. Browne, E. Powley, D. Whitehouse, S. M. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton. A survey of monte

carlo tree search methods. *Transactions on Computational Intelligence and AI in Games*, 2012. doi: 10.1109/TCIAIG.2012.2186810.

E. Brunskill. Lectures on reinforcement learning. url: https://web.stanford.edu/class/cs234/CS234Win2019/index.html, 2019.

M. Carroll, R. Shah, M. K. Ho, T. Griffiths, S. Seshia, P. Abbeel, and A. Dragan. On the utility of learning about humans for human-ai coordination. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, 2019.

T. Cazenave and N. Jouandeau. On the parallelization of uct. 2007.

G. M. J.-B. Chaslot, M. H. M. Winands, H. J. V. D. Herik, J. W. H. M. Uiterwijk, and B. Bouzy. Progressive strategies for monte-carlo tree search. *New Mathematics and Natural Computation*, 2008a. doi: 10.1142/S1793005708001094.

G. M. J. B. Chaslot, M. H. M. Winands, and H. J. van den Herik. Parallel monte-carlo tree search. In *Computers and Games*. Springer, 2008b.

J. Chen, C. Tang, L. Xin, S. E. Li, and M. Tomizuka. Continuous decision making for on-road autonomous driving under uncertain and interactive environments. In *Intelligent Vehicles Symposium (IV)*. IEEE, 2018. doi: 10.1109/IVS.2018.8500605.

B. E. Childs, J. H. Brodeur, and L. Kocsis. Transpositions and move groups in monte carlo tree search. In *Symposium On Computational Intelligence and Games (CIG)*. IEEE, 2008.

D. Connell and H. M. La. Extended rapidly exploring random treebased dynamic path planning and replanning for mobile robots. *International Journal of Advanced Robotic Systems*, 2018. doi: 10.1177/1729881418773874.

A. Couëtoux, J.-B. Hoock, N. Sokolovska, O. Teytaud, and N. Bonnard. Continuous upper confidence trees. In C. A. C. Coello, editor, *Learning and Intelligent Optimization*. Springer, 2011. doi: 10.1007/978-3-642-25566-3_32.

A. Couëtoux, H. Doghmen, and O. Teytaud. Improving the exploration in upper confidence trees. In Y. Hamadi and M. Schoenauer, editors, *Learning and Intelligent Optimization*. Springer, 2012.

R. Coulom. Efficient selectivity and backup operators in monte-carlo tree search. In H. J. van den Herik, P. Ciancarini, and H. H. L. M. J. Donkers, editors, *Computers and Games*. Springer, 2007.

M. Düring and P. Pascheka. Cooperative decentralized decision making for conflict resolution among autonomous agents. In *International Symposium on Innovations in Intelligent Systems and Applications (INISTA)*. IEEE, 2014. doi: 10.1109/INISTA.2014.6873612.

S. C. Eimler and S. Geisler. *Zur Akzeptanz Autonomen Fahrens Eine A-Priori Studie*. De Gruyter, 2015. doi: doi:10.1515/9783110443905-075.

C. Englund, L. Chen, J. Ploeg, E. Semsar-Kazerooni, A. Voronov, H. H. Bengtsson, and J. Didoff. The grand cooperative driving challenge 2016: boosting the introduction of cooperative automated vehicles. *Wireless Communications*, 2016. doi: 10.1109/MWC.2016.7553038.

M. Enzenberger and M. Müller. A lock-free multithreaded monte-carlo tree search algorithm. In *Lecture Notes in Computer Science*. Springer, 2010. doi: 10.1007/978-3-642-12993-3_2.

C. Ericson. *Real-Time Collision Detection*. CRC Press, 2004.

M. Feurer and F. Hutter. *Hyperparameter Optimization*. Springer, 2019. doi: 10.1007/978-3-030-05318-5_1.

C. Finn, S. Levine, and P. Abbeel. Guided cost learning: Deep inverse optimal control via policy optimization. In *International Conference on Machine Learning (ICML)*. JMLR.org, 2016. doi: 10.5555/3045390.3045397.

C. Frese, J. Beyerer, and P. Zimmer. Cooperation of cars and formation of cooperative groups. In *Intelligent Vehicles Symposium (IV)*. IEEE, 2007. doi: 10.1109/IVS.2007.4290119.

A. Gupta, J. Johnson, L. Fei-Fei, S. Savarese, and A. Alahi. Social gan: Socially acceptable trajectories with generative adversarial networks. In *Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE/CVF, 2018. doi: 10.1109/CVPR.2018.00240.

J. R. Hershey and P. A. Olsen. Approximating the kullback leibler divergence between gaussian mixture models. In *International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2007. doi: 10.1109/ICASSP.2007.366913.

R. Hine. *A Dictionary of Biology*. Oxford University Press, 2019.

J. Ho and S. Ermon. Generative adversarial imitation learning. In D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 2016.

C.-J. Hoel, K. Driggs-Campbell, K. Wolff, L. Laine, and M. J. Kochenderfer. Combining planning and deep reinforcement learning in tactical decision making for autonomous driving. *Transactions on Intelligent Vehicles*, 2020. doi: 10.1109/TIV.2019.2955905.

C. Hubmann, M. Becker, D. Althoff, D. Lenz, and C. Stiller. Decision making for autonomous driving considering interaction and uncertain prediction of surrounding vehicles. In *Intelligent Vehicles Symposium (IV)*. IEEE, 2017. doi: 10.1109/IVS.2017.7995949.

C. Hubmann, J. Schulz, M. Becker, D. Althoff, and C. Stiller. Automated driving in uncertain environments: Planning with interaction and uncertain maneuver prediction. *Transactions on Intelligent Vehicles*, 2018a. doi: 10.1109/TIV.2017.2788208.

C. Hubmann, J. Schulz, G. Xu, D. Althoff, and C. Stiller. A belief state planner for interactive merge maneuvers in congested traffic. In *Intelligent Transportation Systems Conference (ITSC)*. IEEE, 2018b. doi: 10.1109/ITSC.2018.8569729.

C. Hubmann, N. Quetschlich, J. Schulz, J. Bernhard, D. Althoff, and C. Stiller. A pomdp maneuver planner for occlusions in urban scenarios. In *Intelligent Vehicles Symposium (IV)*. IEEE, 2019. doi: 10.1109/IVS.2019.8814179.

F. Hutter, H. H. Hoos, and K. Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In C. A. C. Coello, editor, *Learning and Intelligent Optimization*. Springer, 2011.

F. Hutter, H. Hoos, and K. Leyton-Brown. An efficient approach for assessing hyperparameter importance. In *International Conference on Machine Learning (ICML)*. JMLR.org, 2014.

D. Isele. Interactive decision making for autonomous vehicles in dense traffic. In *Intelligent Transportation Systems Conference (ITSC)*. IEEE, 2019. doi: 10.1109/ITSC.2019.8916982.

L. Kaiser, M. Babaeizadeh, P. Miłos, B. Osiński, R. H. Campbell, K. Czechowski, D. Erhan, C. Finn, P. Kozakowski, S. Levine, A. Mohiuddin, R. Sepassi, G. Tucker, and H. Michalewski. Model based reinforcement learning for atari. In *International Conference on Learning Representations (ICLR)*, 2020.

A. Kesting, M. Treiber, and D. Helbing. General lane-changing model mobil for car-following models. *Transportation Research Record*, 2007. doi: 10.3141/1999-10.

S. Khandelwal, W. Qi, J. Singh, A. Hartnett, and D. Ramanan. What-if motion prediction for autonomous driving. arXiv, 2020. doi: 10.48550/ARXIV.2008.10587.

D. Klimenko, J. Song, and H. Kurniawati. Tapir: A software toolkit for approximating and adapting pomdp solutions online. In *Australasian Conference on Robotics and Automation (ACRA)*, 2014.

L. Kocsis and C. Szepesvári. Bandit based monte-carlo planning. In J. Fürnkranz, T. Scheffer, and M. Spiliopoulou, editors, *Machine Learning: ECML*. Springer, 2006.

J. Kong, M. Pfeiffer, G. Schildbach, and F. Borrelli. Kinematic and dynamic vehicle models for autonomous driving control design. In *Intelligent Vehicles Symposium (IV)*. IEEE, 2015. doi: 10.1109/IVS.2015.7225830.

M. Kuderer, S. Gulati, and W. Burgard. Learning driving styles for autonomous vehicles from demonstration. In *International Conference on Robotics and Automation (ICRA)*. IEEE, 2015. doi: 10.1109/ICRA.2015.7139555.

A. Kuefler, J. Morton, T. Wheeler, and M. Kochenderfer. Imitating driver behavior with generative adversarial networks. In *Intelligent Vehicles Symposium (IV)*. IEEE, 2017. doi: 10.1109/IVS.2017.7995721.

S. M. LaValle. *Planning Algorithms*. Cambridge University Press, 2006.

A. Lawitzky, D. Althoff, C. F. Passenberg, G. Tanzmeister, D. Wollherr, and M. Buss. Interactive scene prediction for automotive applications. In *Intelligent Vehicles Symposium (IV)*. IEEE, 2013. doi: 10.1109/IVS.2013.6629601.

D. Lenz, T. Kessler, and A. Knoll. Tactical cooperative planning for autonomous highway driving using monte-carlo tree search. In *Intelligent Vehicles Symposium (IV)*. IEEE, 2016. doi: 10.1109/IVS.2016.7535424.

M. Lindauer, K. Eggensperger, M. Feurer, A. Biedenkapp, D. Deng, C. Benjamins, T. Ruhkopf, R. Sass, and F. Hutter. Smac3: A versatile bayesian optimization package for hyperparameter optimization. *Journal of Machine Learning Research*, 2022.

P. Lindenfors. *For Whose Benefit?* Springer, 2017.

M. L. Littman. Markov games as a framework for multi-agent reinforcement learning. In W. W. Cohen and H. Hirsh, editors, *Machine Learning Proceedings*. Morgan Kaufmann, 1994. doi: 10.1016/B978-1-55860-335-6.50027-1.

F. G. Lopez, J. Abbenseth, C. Henkel, and S. Dörr. A predictive online path planning and optimization approach for cooperative mobile service robot navigation in industrial applications. In *European Conference on Mobile Robots (ECMR)*, 2017. doi: 10.1109/ECMR.2017.8098677.

G. J. McLachlan, S. X. Lee, and S. I. Rathnayake. Finite mixture models. *Annual Review of Statistics and Its Application*, 2019. doi: 10.1146/annurev-statistics-031017-100325.

S. A. Mirsoleimani., A. Plaat., J. van den Herik., and J. Vermaseren. An analysis of virtual loss in parallel mcts. In *International Conference on Agents and Artificial Intelligence (ICAART)*. INSTICC, SciTePress, 2017. doi: 10.5220/0006205806480652.

S. A. Mirsoleimani., J. van den Herik., A. Plaat., and J. Vermaseren. A lock-free algorithm for parallel mcts. In *International Conference on Agents and Artificial Intelligence (ICAART)*. INSTICC, SciTePress, 2018. doi: 10.5220/0006653505890598.

V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis. Human-level control through deep reinforcement learning. *Nature*, 2015. doi: 10.1038/nature14236.

T. M. Moerland, J. Broekens, A. Plaat, and C. M. Jonker. A0c: Alpha zero in continuous action space. arXiv, 2018. doi: 10.48550/ARXIV.1805.09613.

D. C. Montgomery and G. C. Runger. *Applied statistics and probability for engineers*. John Wiley & Sons, 2020.

P. Moritz, R. Nishihara, S. Wang, A. Tumanov, R. Liaw, E. Liang, M. Elibol, Z. Yang, W. Paul, M. I. Jordan, and I. Stoica. Ray: A distributed framework for emerging ai applications. In *Conference on Operating Systems Design and Implementation (OSDI)*. USENIX Association, 2018.

R. B. Myerson. *Game theory: analysis of conflict*. Harvard university press, 1997.

M. Naumann and C. Stiller. Towards cooperative motion planning for automated vehicles in mixed traffic. arXiv, 2017. doi: 10.48550/ARXIV.1708.06962.

M. Naumann, M. Lauer, and C. Stiller. Generating comfortable, safe and comprehensible trajectories for automated vehicles in mixed traffic. In *Intelligent Transportation Systems Conference (ITSC)*. IEEE, 2018. doi: 10.1109/ITSC.2018.8569658.

M. Naumann, L. Sun, W. Zhan, and M. Tomizuka. Analyzing the suitability of cost functions for explaining and imitating human driving behavior based on inverse reinforcement learning. In *International Conference on Robotics and Automation (ICRA)*. IEEE, 2020. doi: 10.1109/ICRA40945.2020.9196795.

A. Y. Ng and S. J. Russell. Algorithms for inverse reinforcement learning. In *International Conference on Machine Learning (ICML)*. Morgan Kaufmann Publishers Inc., 2000.

F. Nielsen. On the jensenshannon symmetrization of distances relying on abstract means. *Entropy*, 2019. doi: 10.3390/e21050485.

I. Open Source Robotics Foundation. Robotic operating system, 2020. URL https://www.ros.org.

A. B. Owen. *Monte Carlo theory, methods and examples*. Owen, Art B., 2013. URL https://artowen.su.domains/mc.

M. Quigley, B. Gerkey, K. Conle, J. Faust, T. Foote, J. Leibs, E. Berger, R. Wheeler, and A. Ng. Ros: an open-source robot operating system. In *International Conference on Robotics and Automation (ICRA)*. IEEE, 2009.

D. Reynolds. *Gaussian Mixture Models*. Springer, 2015. doi: 10.1007/978-1-4899-7488-4_196.

K. Rocki and R. Suda. Large-scale parallel monte carlo tree search on gpu. In *International Symposium on Parallel and Distributed Processing (IPDPS)*. IEEE, 2011. doi: 10.1109/IPDPS.2011.370.

P. Rolet, M. Sebag, and O. Teytaud. Upper confidence trees and billiards for optimal active learning. In *CAP*. HAL, 2009.

SAE J3216. Taxonomy and Definitions for Terms Related to Cooperative Driving Automation for On-Road Motor Vehicles. Technical report, SAE International, 2021.

J. Saito, M. Winands, J. Uiterwijk, and H. Van Den Herik. Grouping nodes for monte-carlo tree search. In *Belgian-Dutch Conference on Artificial Intelligence*, 2007.

D. M. Saxena, S. Bae, A. Nakhaei, K. Fujimura, and M. Likhachev. Driving in dense traffic with model-free reinforcement learning. In *International Conference on Robotics and Automation (ICRA)*. IEEE, 2020. doi: 10.1109/ICRA40945.2020.9197132.

M. S. N. S. J. Schaeffer, N. Shafiei, et al. Comparing uct versus cfr in simultaneous games. In *International Joint Conference on Artificial Intelligence (IJCAI)*. AAAI Press, 2009.

D. Schramm, M. Hiller, and R. Bardini. *Vehicle Dynamics*. Springer, 2018. doi: 10.1007/978-3-662-54483-9.

J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz. Trust region policy optimization. In F. Bach and D. Blei, editors, *International Conference on Machine Learning (ICML)*. PMLR, 2015.

J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. arXiv, 2017. doi: 10.48550/ARXIV.1707.06347.

W. Schwarting and P. Pascheka. Recursive conflict resolution for cooperative motion planning in dynamic highway traffic. In *Intelligent Transportation Systems Conference (ITSC)*. IEEE, 2014. doi: 10.1109/ITSC.2014.6957825.

W. Schwarting, A. Pierson, J. Alonso-Mora, S. Karaman, and D. Rus. Social behavior for autonomous vehicles. *Proceedings of the National Academy of Sciences*, 2019. doi: 10.1073/pnas.1820676116.

U. Seiffert. Artificial neural networks on massively parallel computer hardware. *Neurocomputing*, 2004. doi: 10.1016/j.neucom.2004.01.011.

K. M. Seiler, H. Kurniawati, and S. P. N. Singh. An online and approximate solver for pomdps with continuous action space. In *International Conference on Robotics and Automation (ICRA)*. IEEE, 2015. doi: 10.1109/ICRA.2015.7139503.

B. Shahriari, K. Swersky, Z. Wang, R. P. Adams, and N. de Freitas. Taking the human out of the loop: A review of bayesian optimization. *Proceedings of the IEEE*, 2016. doi: 10.1109/JPROC.2015.2494218.

S. Shalev-Shwartz, S. Shammah, and A. Shashua. On a formal model of safe and scalable self-driving cars. arXiv, 2018. doi: 10.48550/ARXIV.1708.06374.

S. E. Shladover. Opportunities and challenges in cooperative road vehicle automation. *Open Journal of Intelligent Transportation Systems*, 2021. doi: 10.1109/OJITS.2021.3099976.

Y. Shoham and K. Leyton-Brown. *Multiagent systems: Algorithmic, game-theoretic, and logical foundations*. Cambridge University Press, 2008.

D. Silver. *Reinforcement Learning and Simulation-Based Search in Computer Go*. PhD thesis, University of Alberta, 2009.

D. Silver. Lectures on reinforcement learning. url: https://www.davidsilver.uk/teaching/, 2015.

D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis. Mastering the game of go with deep neural networks and tree search. *Nature*, 2016. doi: 10.1038/nature16961.

Y. Soejima, A. Kishimoto, and O. Watanabe. Evaluating root parallelization in go. *Transactions on Computational Intelligence and AI in Games*, 2010. doi: 10.1109/TCIAIG.2010.2096427.

D. J. N. J. Soemers, C. F. Sironi, T. Schuster, and M. H. M. Winands. Enhancements for real-time monte-carlo tree search in general video game playing. In

Conference on Computational Intelligence and Games (CIG). IEEE, 2016. doi: 10.1109/CIG.2016.7860448.

D. O. Stahl and P. W. Wilson. Experimental evidence on players' models of other players. *Journal of Economic Behavior & Organization*, 1994. doi: 10.1016/0167-2681(94)90103-1.

Z. N. Sunberg, C. J. Ho, and M. J. Kochenderfer. The value of inferring the internal state of traffic participants for autonomous freeway driving. In *American Control Conference (ACC)*, 2017. doi: 10.23919/ACC.2017.7963408.

R. S. Sutton and A. G. Barto. *Reinforcement learning : an introduction*. MIT Press, 2018.

L. Tai, J. Zhang, M. Liu, and W. Burgard. Socially compliant navigation through raw depth inputs with generative adversarial imitation learning. In *International Conference on Robotics and Automation (ICRA)*. IEEE, 2018. doi: 10.1109/ICRA.2018.8460968.

M. J. W. Tak, M. Lanctot, and M. H. M. Winands. Monte carlo tree search variants for simultaneous move games. In *Conference on Computational Intelligence and Games (CIG)*. IEEE, 2014. doi: 10.1109/CIG.2014.6932889.

A. Takahashi, T. Hongo, Y. Ninomiya, and G. Sugimoto. Local path planning and motion control for agv in positioning. In *International Conference on Intelligent Robots and Systems (IROS)*. IEEE/RSJ, 1989. doi: 10.1109/IROS.1989.637936.

R. Tian, L. Sun, M. Tomizuka, and D. Isele. Anytime game-theoretic planning with active reasoning about humans' latent states for human-centered robots. In *International Conference on Robotics and Automation (ICRA)*. IEEE, 2021. doi: 10.1109/ICRA48506.2021.9561463.

E. Tolstaya, R. Mahjourian, C. Downey, B. Vadarajan, B. Sapp, and D. Anguelov. Identifying driver interactions via conditional behavior prediction. In *International Conference on Robotics and Automation (ICRA)*. IEEE, 2021. doi: 10.1109/ICRA48506.2021.9561967.

P. Trautman and A. Krause. Unfreezing the robot: Navigation in dense, interacting crowds. In *International Conference on Intelligent Robots and Systems (IROS)*. IEEE/RSJ, 2010. doi: 10.1109/IROS.2010.5654369.

M. Treiber, A. Hennecke, and D. Helbing. Congested traffic states in empirical observations and microscopic simulations. *Phys. Rev. E*, 2000. doi: 10.1103/PhysRevE.62.1805.

A. Vemula, K. Muelling, and J. Oh. Modeling cooperative navigation in dense human crowds. In *International Conference on Robotics and Automation (ICRA)*. IEEE, 2017. doi: 10.1109/ICRA.2017.7989199.

Y. Wang, J.-Y. Audibert, and R. Munos. Algorithms for infinitely many-armed bandits. In *International Conference on Neural Information Processing Systems (NIPS)*. Curran Associates Inc., 2008.

M. Werling, J. Ziegler, S. Kammel, and S. Thrun. Optimal trajectory generation for dynamic street scenarios in a frenét frame. In *International Conference on Robotics and Automation (ICRA)*. IEEE, 2010. doi: 10.1109/ROBOT.2010.5509799.

Z. Wu, L. Sun, W. Zhan, C. Yang, and M. Tomizuka. Efficient sampling-based maximum entropy inverse reinforcement learning with application to autonomous driving. *Robotics and Automation Letters*, 2020. doi: 10.1109/LRA.2020.3005126.

M. Wulfmeier, P. Ondruska, and I. Posner. Maximum entropy deep inverse reinforcement learning. arXiv, 2016a. doi: 10.48550/ARXIV.1507.04888.

M. Wulfmeier, D. Z. Wang, and I. Posner. Watch this: Scalable cost-function learning for path planning in urban environments. In *International Conference on Intelligent Robots and Systems (IROS)*. IEEE/RSJ, 2016b. doi: 10.1109/IROS.2016.7759328.

T. Yee, V. Lisy, and M. Bowling. Monte carlo tree search in continuous action spaces with execution uncertainty. In *International Joint Conference on Artificial Intelligence (IJCAI)*. AAAI Press, 2016.

K. Zhang, Z. Yang, and T. Başar. *Multi-Agent Reinforcement Learning: A Selective Overview of Theories and Algorithms*. Springer, 2021. doi: 10.1007/978-3-030-60990-0_12.

B. D. Ziebart, A. Maas, J. A. Bagnell, and A. K. Dey. Maximum entropy inverse reinforcement learning. In *National Conference on Artificial Intelligence (AAAI)*. AAAI Press, 2008.

J. Ziegler and C. Stiller. Fast collision checking for intelligent vehicle motion planning. In *Intelligent Vehicles Symposium (IV)*. IEEE, 2010. doi: 10.1109/IVS.2010.5547976.