TECHNISCHE UNIVERSITÄT MÜNCHEN

Fakultät für Informatik

Cooperative Behavior Planning for Autonomous Vehicles Using Mixed-Integer Programming

Tobias Kessler

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

Doktors der Naturwissenschaften (Dr. rer. nat.)

genehmigten Dissertation.

Vorsitzender:		Prof. Dr.-Ing. Matthias Nießner
Prüfer der Dissertation:	1.	Prof. Dr.-Ing. habil. Alois Knoll
	2.	Prof. Dr.techn. Daniel Watzenig

Die Dissertation wurde am 23.02.2022 bei der Technischen Universität München eingereicht und durch die Fakultät für Informatik am 15.06.2022 angenommen.

Abstract

Nowadays, traffic is organized by rules for driving safety and social conventions to achieve fairness among traffic participants and increased traffic efficiency. Drivers are expected to not behave egoistically or blindly obey the traffic rules but cooperatively resolve a situation while still pursuing an individual goal.

This thesis transfers cooperative driving practice into behavior and motion planning algorithms for autonomous vehicles. Such vehicles shall, by cooperative planning, considerately enforce their own goal in potentially ambiguous driving situations to pave the path towards a social acceptance of autonomous driving by not acting overly passive or aggressive. Current autonomous driving prototypes barely show cooperative behavior. However, not only the interaction with human-driven vehicles is essential. With an expected growth in the importance of inter-vehicle communication, also connected automated vehicles have to work together cooperatively to maintain the positive effects on traffic drivers achieve by cooperation.

This work formalizes the cooperative planning problem in a game-theoretic framework, formulates a multi-agent optimization problem, and solves it using mixed-integer programming. Constraints ensure valid vehicle kinematics and dynamics and a collision-free motion within the road environment among all agents and obstacles. The objective function leverages the interests of all interacting agents by introducing a cooperation factor to scale between altruistic and aggressive behavior seamlessly. Two scenario-agnostic algorithms are presented, using either discrete or continuous actions. Both apply robustness measures against errors in the perception and prediction of other agents. The performance is assessed in simulation. Also, test drives with a prototype autonomous vehicle are performed by embedding one algorithm into the open-source autonomous driving stack Apollo to prove the applicability in a full-size setup. This thesis describes the adaptions of the driving stack for the vehicle, including the methodology, implementation, and lessons learned for interfacing the behavior planner.

The real-road driving experiments show the applicability of the planner and the overall software stack, the performance, and the real-time capability of the implementation. A simulation-based complexity analysis further assesses the real-time capability of mixed-integer optimization-based planning and states practical restrictions. Besides three on-road driving experiments demonstrating real-time applicability with multiple obstacles present, various simulated scenarios demonstrate the effectiveness of the cooperative planning approach. These include a low-speed urban road, an intersection, and a highway scenario with varying settings, cooperation levels, and communication assumptions. The algorithms not only show passive but proactive and truly cooperative behavior producing trajectories directly trackable by a vehicle controller. The presented adaption of Apollo to a German autonomous driving prototype vehicle and interface to a mixed-integer optimization-based planner is published as open-source software and shall accelerate other research groups to validate behavior and motion planning algorithms.

Zusammenfassung

Im Straßenverkehr gelten neben den Verkehrsregeln zur Gewährleistung der Sicherheit auch soziale Konventionen. Diese sollen Fairness zwischen den Verkehrsteilnehmern sicherstellen, tragen aber auch zu einem effizienten Verkehrsgeschehen bei. Von Fahrern wird erwartet, dass sie sich zielorientiert und trotzdem kooperativ mit anderen Verkehrsteilnehmern verhalten und Verkehrssituationen nicht durch rein egoistisches Verhalten oder dem sturem Folgen der Verkehrsregeln auflösen.

Diese Arbeit überträgt kooperatives Fahrverhalten menschlicher Fahrer in Verhaltens- und Bewegungsplanungsalgorithmen für autonome Fahrzeuge. Durch kooperative Planung sollen diese Fahrzeuge situationsbewusst und rücksichtsvoll ihr eigenes Ziel in unklaren Verkehrssituationen erreichen, was aktuelle Prototypen autonomer Fahrzeuge nicht tun. Dabei soll ein Fahrzeug weder zu aggressiv noch zu passiv agieren, um im Straßenverkehr sozial akzeptiert zu werden. Neben der Interaktion mit Fahrern ist zusätzlich die Interaktion mit vernetzen Fahrzeugen notwendig, deren Bedeutung stetig wächst. Auch vernetze autonome Fahrzeug müssen zusammenarbeiten um die positiven Eigenschaften der Kooperation zwischen menschlichen Fahrern beizubehalten.

In dieser Arbeit wird die Problemstellung der kooperativen Planung zuerst in einem spieltheoretischen Ansatz formalisiert, dann als multi-Agenten Optimierungsproblem formuliert und mittels gemischt-ganzzahliger Programmierung gelöst. Nebenbedingungen stellen dabei die korrekte Abbildung der Kinematik und Dynamik eines Fahrzeugs und die Kollisionsfreiheit der Agenten untereinander, zu Hindernissen, und mit der Straßenumgebung sicher. Die Zielfunktion balanciert die Interessen aller interagierenden Agenten aus. Dafür wird ein Kooperationsfaktor eingeführt, durch den nahtlos zwischen altruistischem und aggressivem Verhalten skaliert werden kann. Zwei Szenario-agnostische Algorithmen werden vorgestellt, einmal wird ein diskreter und einmal ein kontinuierlichem Aktionsraum verwendet. Beide Absätze sind robust gegenüber Fehlern in der Wahrnehmung und Prädiktion anderer Agenten. Die Ergebnisse werden simulativ gezeigt. Zusätzlich werden Testfahren mit einem autonomen Forschungsfahrzeug durchgeführt. Dafür wird einer der Algorithmen in den quelloffenen Software-Stack Apollo integriert und die Anwendbarkeit im Gesamtsystem gezeigt. Diese Arbeit beschreibt auch die nötigen Anpassungen an Apollo für den verwendeten Forschungsdemonstrator, insbesondere die Methodik, Implementierungen und Erfahrungen bei der Anbindung der Planungskomponente.

Die Fahrversuche auf öffentlicher Straße zeigen nicht nur reale Anwendbarkeit des Planers und des Gesamtsystems, sondern auch die Leistungsfähigkeit und die Echtzeitfähigkeit der Implementierung. Eine simulationsbasierte Komplexitätsanalyse bewertet zusätzlich strukturiert die Echtzeitfähigkeit der auf gemischt-ganzzahliger Optimierung basierenden Planungsalgorithmen und zeigt praktische Einschränkungen auf. Drei Fahrversuche auf öffentlicher Straße demonstrieren die Anwendbarkeit in Echtzeit mit mehreren Hindernissen. Ebenso zeigt diese Arbeit die Möglichkeiten und Effektivität der kooperativen Planung in mehreren simulierten Fahrszenarien, wie einem Autobahnszenario, ein Kreuzungsszenario oder einem Szenario auf einer innerstädtischen Straße mit niedrigen Geschwindigkeiten. Es werden unterschiedliche Einstellungen, Grad der Kooperation und Kommunikationsmöglichkeiten analysiert. Die entwickelten Planungsalgorithmen zeigen dabei nicht ausschließlich passives, sondern proaktives und wirklich kooperatives Fahrverhalten. Die Fahrtrajektorien können direkt von einem Fahrzeugregler verwendet werden. Die in dieser Arbeit beschriebenen Anpassungen an Apollo zur Ansteuerung eines deutschen Prototypenfahrzeugs und zur Einbindung einer Planungskomponente basierend auf gemischt-ganzzahliger Optimierung sind unter offener Lizenz verfügbar und sollen andere Forschungsgruppen dabei unterstützen eigene Arbeiten zur Verhaltens- und Bewegungsplanung in realen Umgebungen zu validieren.

Contents

1 Introduction

For a fully autonomous vehicle, generating behavior that is aware of other traffic participants such as human-driven vehicles, Connected Autonomous Vehicles (CAVs), or other objects on the road is one essential ingredient to release production-grade autonomous driving functions on public roads. This thesis will develop cooperative behavior and motion planning algorithms in generic, on-road scenarios using Mixed-Integer Progamming (MIP) and demonstrate these in simulation and using a prototype autonomous vehicle.

1.1 Motivation

By cooperation human drivers achieve fairness, trust, and efficiency in ambiguous driving scenarios, where many behaviors are legal. They rely on nonverbal communication and the interpretation of intentions to cooperatively resolve unclear situations, besides following the traffic rules for safety. When sharing the road with human drivers, autonomous vehicles will also have to show cooperative behavior to be accepted by society and to fit safely into nowadays traffic scenarios while still asserting their own goals [Mer+20]. In dense traffic, where space is limited, the reactions of others must be anticipated, and an autonomous vehicle must model the uncertain interaction with the other traffic participants by planning a joint action for the ego vehicle and the surrounding vehicles. Game-theoretic approaches are capable of modeling the interaction between multiple traffic participants elegantly [SAR18]. With the transition from manual to automated driving, the beneficial features of cooperation shall be kept and transferred into algorithms. Technically, a vehicle acts cooperatively if the intents of other traffic participants influence its behavior plan and if it aims to optimize others' behavior alongside.

Figure 1.1 sketches an everyday driving scene that is easily solved using cooperation: Both vehicles can pass the obstacles without coming to a complete stop with slight deviations from the desired steering and velocity, the optimal solution in this case. Still, several other reasonable driving options are possible. Even in this straightforward situation, state-of-the-art planning algorithms often fail to find the optimal solution as the strategic decision on the behavior is mostly separated from the motion planning layer. However, strategy and motion are closely intertwined in dense scenarios. Since each vehicle's action potentially interferes with all other traffic participants, the computational complexity grows with the number of participants. Still, for a comprehensive, interaction-aware plan, the behavior of each interacting traffic participant has to be included. Solution algorithms often rely on a random sampling of the solution space or lack guarantees of convergence and thus pose open questions towards certification of such systems. For data-driven approaches, it is hard to prove deterministic behavior and generalization to unseen situations.

For general applicability, a planning approach has to be scenario-agnostic and should not only be fine-tuned for one specific use case. Otherwise, switching between algorithms is necessary, introducing additional complexity into an already highly complicated system. Also, the engineering and certification effort increases with more algorithms in place. The first autonomous vehicles have to interact with potentially non-cooperative human-driven vehicles, especially in ambiguous and conflict situations, without relying on Vehicle-to-Vehicle (V2V) communication. Nevertheless, the importance of connectivity will rise soon with a higher penetration rate of automated driving functions and road traffic digitization such as real-time digital twins of highways [Krä+22]. Autonomous driving systems developed now should already be designed to utilize such information or even coordinate entire groups of CAVs. Explicit communication among connected vehicles

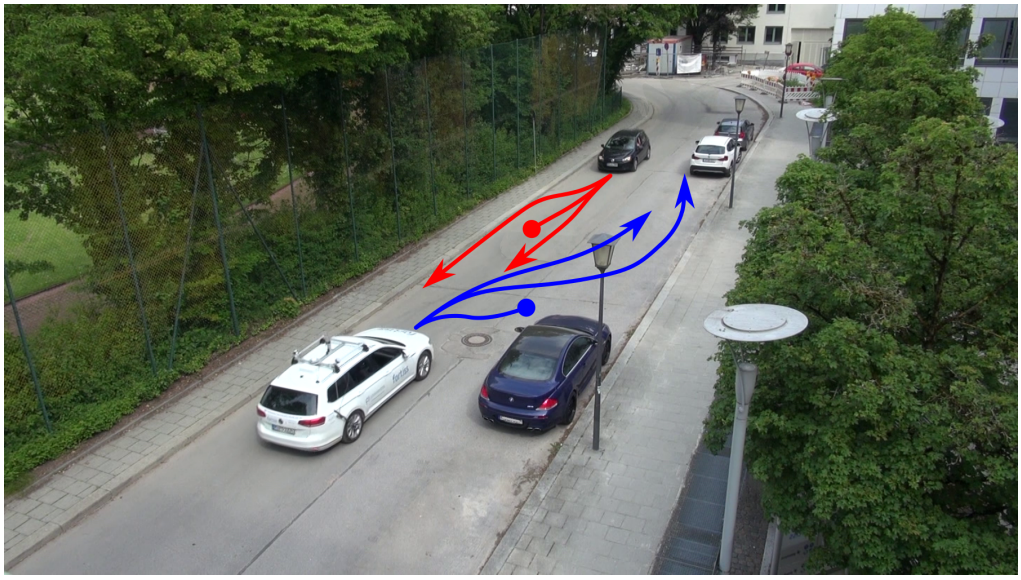


Figure 1.1: Illustrative example traffic scene with multiple behavior options: With a cooperative behavior plan, neither the white nor the black vehicle must come to a complete stop (visualized by a dot) to let the other pass. Several reasonable behavior options are possible, and the action of either vehicle influences the possibilities of the other vehicle.

will significantly enhance the overall performance of the traffic system [Ulb+15] by planning a coordinated behavior for a group of vehicles. Algorithms developed now should already address all these varieties of application scenarios.

To prove the applicability of research algorithms, the evaluation using an experimental vehicle platform in real-world scenarios is essential. Often the performance of behavior and motion planning research algorithms is only shown in simulation rather than using a prototype vehicle (in Europe); notable exceptions are, e.g., [Zie+14a; Gra+21; Wan+21]. Reasons are, for example, the technical and organizational complexity to setup and maintain a complete software stack for autonomous driving, the limited availability of research vehicles and test areas, and legal aspects when operating an experimental vehicle on public roads. Hence, various published planning algorithms exist that rely on restrictive assumptions or simplification where it is questionable if these can be transferred to reality. Without showing on-road experiments, even for algorithms demonstrated in advanced simulators, still, a reality gap exists. Algorithms from demonstration by automotive companies, on the other hand, are often not available to the research community and hence cannot be compared to state-of-the-art research approaches.

1.2 Research Gap and Contributions

To facilitate fully autonomous driving, a unified behavior coordination approach operating in various locations and scenarios that can deal with all possible constellations of human-driven, fully automated, communicating, or non-communicating vehicles, cooperative or egoistic behavior of others has to be implemented. Also, the layers for motion planning (smooth and collision-free trajectories) and behavior planning (tactic decisions, rule-compliance) have to be integrated and not treated separately for high performance in generic scenarios. MIP-based planning algorithms have been proposed to tackle these challenges. However, most proposed methods only generate valid results on a small subset of scenarios, namely straight roads, and become invalid in any other environment (roundabouts, intersections), or even during obstacle avoidance at low speeds, as the

valid scope of the model formulation is limited. With the contributions and gaps from previous research in mind, this thesis addresses the following research questions.

1. **How can an existing open-source autonomous driving software stack be transferred to an experimental vehicle for behavior and motion planning research?** A fully autonomous vehicle is a highly complex and complicated system, and production-grade vehicles will probably be among the most advanced machines that will be released to non-expert users. Prototype vehicles share this complexity. The majority of complexity is tackled by software. This thesis aims to develop and apply the planning software component in a realistic vehicle setup and therefore needs a hardware platform and a practical software setup for other software functionalities. Open-source autonomous driving stacks [Bai17; Kat+18] offer these software functionalities but research software then has to be integrated into the stack. Research algorithms also pose requirements on the generic software stack that might not be fulfilled. Also, the stack has to be interfaced with the vehicle platform. This thesis will elaborate on the choice of Apollo [Bai17] as a platform, whether the engineering effort of integrating it was valuable, and what results can be expected.

2. **Can we develop a scenario-agnostic, unified cooperative planning approach integrating behavior and trajectory planning?** Behavior and motion planning algorithms are an integrative component in each autonomous driving stack and ideally do not pose requirements on the driving scenario. For systems with a limited operational design domain, such as a traffic jam pilot, tuning the functionality to a specific use case is valuable. However, this methodology will not scale for fully autonomous driving. Switching the planning algorithm raises the additional complexity of deciding when or where to switch and to interpret the scenario at hand correctly. Also, with an algorithmic break between high-level behavior planning and low-level motion planning, the risk is introduced that a high-level, long-term decision cannot be executed with respect to the current vehicle kinematics and dynamics or the traffic state within the next second. Furthermore, interacting agents can be of a different type, such as human-driven vehicles, CAVs, or vulnerable road users. This thesis takes the interactivity of traffic participants into account to plan an executable motion in alignment with all strategic goals.

3. **How can autonomous vehicles plan cooperative behavior, like human drivers, while still achieving their own goals?** Most human drivers try to leverage their own and others driving interests besides blindly following the traffic rules or acting purely egoistically. Examples are letting other vehicles merge when entering freeways or waiving their own right of way at narrow intersections. Automated functions shall show the same behavior to be accepted by human drivers but also to not lose these beneficial social conventions. On the other hand, autonomous vehicles must not react too passively to achieve their driving goals. They also have to be able to cope with human drivers that behave egoistically and do not show cooperation or even break traffic rules. By multi-agent planning, this thesis does not only model the own driving intent but also the estimated intent of other vehicles to coordinate an optimal solution of the traffic scenario.

4. **Can MIP be applied for real-time planning and, if so, which restrictions apply?** Mathematical optimization is one approach to behavior and motion planning of autonomous vehicles, MIP being one subdomain. It offers benefits such as optimality with respect to the model formulation and the optimization of continuous and discrete states alongside. However, it also poses restrictions on the model formulation, and the problem formulation is usually NP-complete with an exponential worst-case runtime. Still, the planning in a vehicle has to be executed in real-time. In this thesis, we develop planning algorithms based on MIP and will evaluate in which scenarios these can leverage the benefits and where the scaling of the complexity is too high for real-time application.

Parts of this dissertation are based on and have already been published in the following papers.

1. Kessler and Knoll [KK17]: *"Multi Vehicle Trajectory Coordination for Automated Parking"* proposes a motion coordination approach for maneuvering scenarios using pre-calculated paths and an optimal assignment based on MIP.

2. Kessler and Knoll [KK19]: *"Cooperative Multi-Vehicle Behavior Coordination for Autonomous Driving"* proposes a planning approach for cooperative multi-vehicle scenarios based on the generation of a option graph and selecting the best option using Mixed-Integer Linear Programming (MILP).

3. Kessler, Bernhard, Buechel, Esterle, Hart, Malovetz, Truong Le, Diehl, Brunner, and Knoll [Kes+19]: *"Bridging the Gap between Open Source Software and Vehicle Hardware for Autonomous Driving"* proposes a strategy to setup an open-source software stack for fully autonomous driving on a prototype vehicle and introduces the research vehicle used in this research.

4. Esterle, Kessler, and Knoll [EKK20]: *"Optimal Behavior Planning for Autonomous Driving: A Generic Mixed-Integer Formulation"* proposes an approach to formulate the vehicle motion planning problem as Mixed-Integer Quadratic Programming (MIQP) and a methodology to derive the necessary linear constraints.

5. Kessler, Esterle, and Knoll [KEK20]: *"Linear Differential Games for Cooperative Behavior Planning of Autonomous Vehicles Using Mixed-Integer Programming"* extends [EKK20] to a multi-agent behavior planning problem with a joint cost function.

6. Kessler, Esterle, and Knoll [KEK22]: *"Mixed-Integer Motion Planning on German Roads within the Apollo Driving Stack"* shows how to integrate [EKK20; KEK20] in an open-source driving stack and and evaluates on-road driving experiments. © 2022 IEEE. Reprinted, with permission, from [KEK22].

While this thesis evolved, this work's author also authored or co-authored the following papers that share ideas with this thesis but are not a core contribution in this work.

1. Lenz, Kessler, and Knoll [LKK15]: *"Stochastic Model Predictive Controller with Chance Constraints for Comfortable and Safe Driving Behavior of Autonomous Vehicles"* proposes a Model-Predictive Control (MPC) controller achieving a smooth motion that is safe by design.

2. Minnerup, Lenz, Kessler, and Knoll [Min+16]: *"Debugging Autonomous Driving Systems Using Serialized Software Components"* proposes a method to debug errors found in test drives a posteriori.

3. Kessler, Minnerup, Lenz, and Knoll [Kes+17]: *"Systematically comparing control approaches in the presence of actuator errors"* shows how to asses the performance of the vehicle control layer in the presence of sensor and actuator errors.

4. Kessler, Minnerup, Esterle, Feist, Mickler, Roth, and Knoll [Kes+18]: *"Roadgraph Generation and Free-Space Estimation in Unknown Structured Environments for Autonomous Vehicle Motion Planning"* generate a static environment representation only from (lidar) sensor data.

5. Buechel, Schellmann, Rosier, Kessler, and Knoll [Bue+19]: *"Fortuna: Presenting the 5G-connected automated vehicle prototype of the project PROVIDENTIA"* introduces the research vehicle used here and the criteria that led to the selection of the hardware components.

6. Bernhard, Esterle, Hart, and Kessler [Ber+20]: *"BARK: Open Behavior Benchmarking in Multi-Agent Environments"* proposes the behavior benchmarking tool BARK to assess the quality of different planning algorithms.

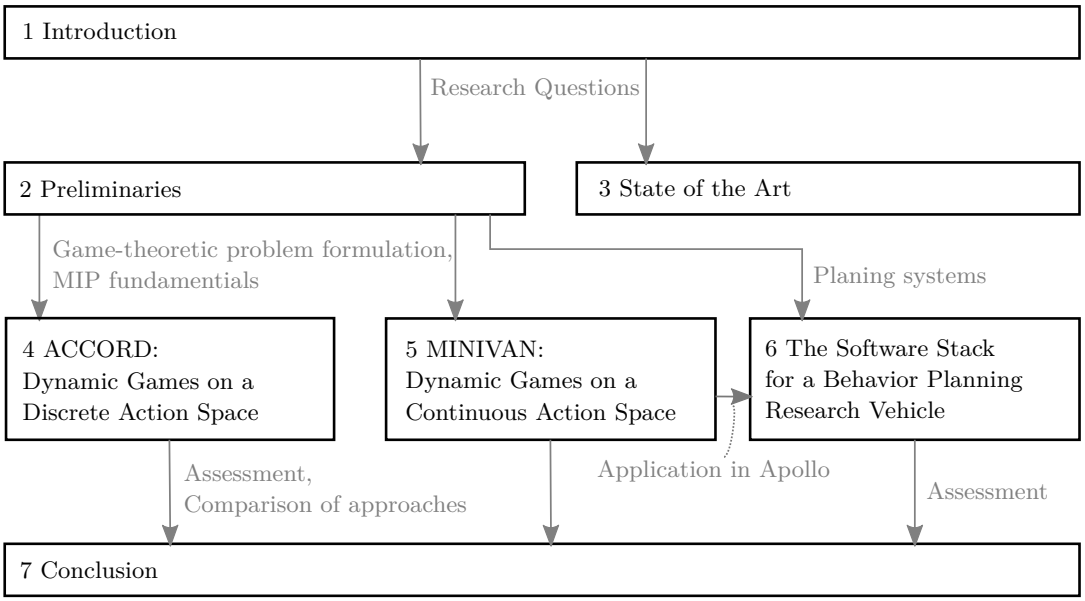


Figure 1.2: Structural overview of this thesis summarizing the chapters and their relations

1.3 Outline of this Thesis

This thesis is structured in seven chapters, that are set in relation in Figure 1.2. After the introduction in this chapter, Chapter 2 will then introduce a game-theoretic problem formulation of the multi-agent behavior planning problem used in the following chapters. The chapter also briefly introduces the technology of MIP and give an overview on how the planning component integrates into a full autonomous driving stack. Chapter 3 gives an overview on the state of the art in the motion and behavior planning research field with a focus on cooperative and mixed-integer optimization approaches. The chapter also presents the Apollo driving stack alongside with an overview of hard- and software for autonomous driving. Chapter 4 will first introduce the Autonomous Car Coordination (ACCORD) planning approach, then analyze the algorithm regarding real-time capability, followed by a demonstration in simulation. The second planner developed in this thesis, the Mixed Integer Interactive Planning (MINIVAN) planning approach, will then be introduced in Chapter 5, which follows a similar structure. Besides both behavior planning algorithms, and extensive simulations studies to analyze their capabilities and shortcomings, the adaption of the Apollo driving stack to our prototype autonomous vehicle is a major contribution of this work. The methodology and findings when integrating a MIP-based planner into Apollo and applying the modified stack in a car is elaborated in Chapter 6 alongside with the analysis of on-road test drives. Also, a the complete hard- and software setup of the demonstration platform is described. Chapter 7 compares the developed planning approaches, states further open research questions, summarizes and concludes this work.

2 Preliminaries

This chapter introduces fundamental information required for this thesis. The general terms and notations will be introduced in Section 2.1, followed by an overview of how the behavior and motion planning functionality is embedded in a complete autonomous driving stack and illustrates the general interfaces of a planning component in Section 2.2. Section 2.3 gives an overview of the different application areas the algorithms developed in this work are used for. This chapter will further formally introduce the multi-agent planning problem using a game-theoretic framework in Section 2.4 and introduce Mixed-Integer Progamming (MIP) as a solution strategy in Section 2.5.

2.1 Terms and Definitions

This section will introduce the general notation used in this work to describe problems and scenarios.

Agent An agent is a decision-making entity in the current scene. This can be the autonomous ego vehicle, another communicating or non-communicating vehicle. Each agent has a time-independent goal that is a state or a set of states it wants to reach. This can, e.g., be a desired final vehicle pose. State denotes the tuple of agent's position, orientation, velocity, and further kinematic and dynamic properties. Each agent tries to reach its goal with a specific intention. Intention in this work denotes the strategic plan of an agent in the current traffic scene such as, e.g., "perform a lane change to drive faster". Technically, this narrows down to the current reference path of the agent leading to its goal plus a desired speed and acceleration.

Behavior and Trajectory An agent's intention plus a tangible, time-dependent, physically feasible motion forms its behavior. Motion generally refers to path or trajectory. A path is a time-independent sequence of kinematic states, such as x, y pose coordinates. A trajectory is a path with timing information resulting in a sequence of dynamic states, including, e.g., the velocity and acceleration values. When specifying a motion for a standard road vehicle, the motion is non-holonomic: The vehicle cannot turn at its point and can only rotate with a certain turning radius. Mathematically, the states of the system are described by nonlinear differential equations.

Interactive Behavior Planning This work aims at planning behaviors in multi-agent systems. Motion planning is the task of generating a path or trajectory as a sequence of states from a given initial state of the system along a time horizon into the future with respect to given constraints, such as collision avoidance. A behavior plan is then a trajectory plan that follows the given intentions. This can be performed in a setting with one or more agents. In such a multi-agent context, goals can be conflicting and the planning problem is finding an optimal behavior plan for all agents, that is free of conflicts and collisions. A behavior plan is denoted as interactive, if all agents in the multi-agent systems are viewed as decision making entities and the ego vehicle planner not only operates on a fixed predicted motion of the other agents.

Communication Agents can be communicating or non-communicating. Our model of communication in this work is basic and only refers to the explicit exchange of states, trajectories, goal, or intentions. If other agent states are not known from communication, they have to be estimated

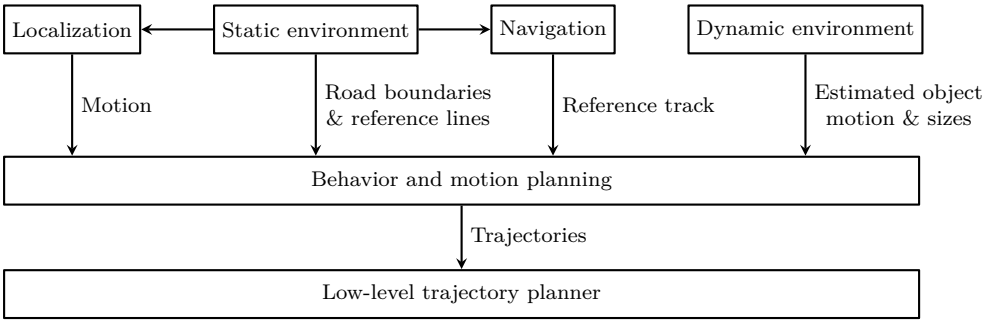


Figure 2.1: Overview of the components interacting with the behavior coordination system and the exchanged information (graphic from [KK19], ©2019 IEEE)

from the ego vehicle's sensor information. The future motion is then predicted based on current and historic states. This work does not account for delays, jitter, security aspects, or faults in the communication.

Cooperation Cooperation in this work denotes the endeavor of an agent to not only achieve its own goal but to consider the intention of other agents into the behavior plan to find an interactive plan where all agents reach their goal. The level of cooperation can be very different. From purely altruistic (such as letting an emergency vehicle pass by while temporarily giving up on the own goal) to balanced (such as changing a lane on a highway to allow others to merge from an entry ramp) to purely egoistic (such as preventing being overtaken in a racing scenario). Every scenario requires the cooperation and interaction of more than one agent.

Coordination In general, the ego agent plans a behavior for itself, including the intention of other agents. Only the behavior for the ego agent will be executed in the end. If one agent, a group of agents, or a traffic controller plans the behavior for all agents in the scene, and all agents then execute the joint behavior plan, this is referred to as coordination. Uncontrolled agents, such as human-driven vehicles, are not expected to obey the assigned behavior fully, but this work implements measures to be robust against these prediction errors. Behavior coordination is also referred to as fully cooperative behavior.

Real-Time Applicability To be able to use a planning approach in autonomous driving, the approach has to be real-time capable. Here only a weak definition of real-time applicability is applied: At any time, a sufficient horizon of the behavior and motion plan has to be available which means that the end of a trajectory is never reached if not on an intentional stopping point. The computation is assumed hardware powerful enough to achieve this but in this work no guarantees from hard- or software are required.

Optimality Optimal solution denotes the globally optimal solution of the planning problem (if it exists). A global optimum does not have to be unique. The problem can have several local optima, that are optimal in a local neighborhood and which local optimal solution an algorithm finds is dependent on the initial solution. Completeness describes the property of an algorithm to guarantee convergence to a valid solution if one exists.

2.2 Planning System Overview

The behavior planning outputs a future trajectory for each participant that is modeled as an individual agent in the traffic scene. This property makes our approach particularly suited in a

cooperative setup where each vehicle follows its calculated motion. For all other agents, the output trajectory represents the predicted motion and intention. Four components serve as main inputs of the behavior planning component. Figure 2.1 shows the main interfaces, that are briefly discussed in the following.

Static Environment The static environment is a map representation containing road and lane reference lines including possible connections (e.g., at an intersection). Also, it represents the road boundaries and known static obstacles in a polygonal way. This component will be referred to as static environment model. Perfect perception and localization of the ego vehicle on the map is assumed in this work.

Dynamic Environment Planning algorithms also rely on a dynamic environment model. It contains the current states (such as pose, orientation, velocity) of all traffic participants in the scene. The source can be the output of a sensor data fusion pipeline or received data from a Vehicle-to-Vehicle (V2V) or Vehicle-to-Everything (V2X) communication unit. Also, shape and size of the traffic participant is assumed to be available. By design this work handles communication-enabled vehicles equivalent to non-communicating (human driven) vehicles. The only difference is how the motion and strategic intention of the vehicle is gathered. In the communicating case, it is assumed that the other vehicle aims to find a cooperative solution and communicates its exact intention. In case of a non-communicating vehicle the intention has to be observed and quantified and this vehicle might not behave as estimated. From an architectural point of view, all types of agents are handled alike, this also applies to the ego agent. Each object in the dynamic environment can be viewed as an agent, a decision-making entity who's actions can change based on the ego actions or as a dynamic obstacle, who's behavior is regarded as fixed and independent of the ego actions.

Localization The third data source is the vehicle's localization component. It provides the ego vehicle states (pose, orientation, velocity) and the localization on the static map.

Navigation A navigation entity provides a goal position, resulting in a reference track within the static environment map. Most of the considered scenarios are on-road driving scenarios, where each agent wants to follow a given time-independent reference line (e.g., a lane) with a desired speed. This reference line can change over time (e.g., when re-routing) and does not have to be collision-free or trackable by the non-holonomic kinematics of a vehicle. In combination with a reference speed, a reference trajectory is derived. With a reference speed of zero at a point on the reference track the special case of stopping at a point is realized.

Low-Level Trajectory Planner For the ego vehicle, a low-level trajectory planner with a subsequent controller is assumed to possibly subsample, track the computed trajectory as close as possible, and react to safety-critical events occurring in the discretization interval of the behavior planner.

2.3 Application Areas of Multi-Agent Behavior Planning

An automated vehicle should operate safely in a wide range of environments and conditions. This work focuses on three different use cases, as sketched in Figure 2.2:

- *Reactive* behavior,
- *Proactive* behavior, and
- *Coordinated* behavior.

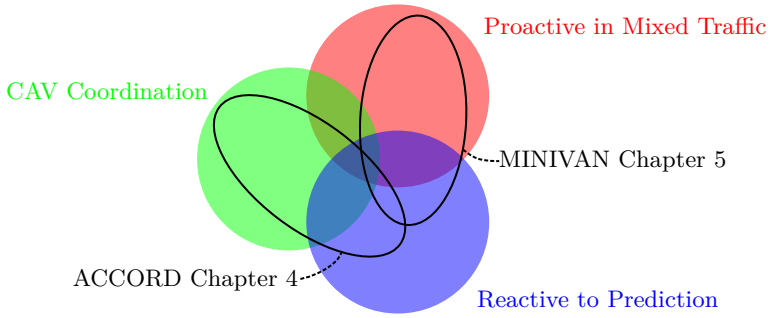


Figure 2.2: The behavior planners developed in this work shall operate in different applications areas. ACCORD is mainly a CAV coordination approach, MINIVAN a proactive, multi-agent planner. Both planners inherently handle all three use cases, including reactive behavior to a prediction.

These three classes cannot be strictly separated but are overlapping. Ideally, the same planning system handles all three classes. This will not only increase the applicability and robustness but also ensures that the implementation is applicable with changing traffic situations, such as an increasing penetration of Connected Autonomous Vehicles (CAVs).

In this thesis, two behavior-aware planning approaches are developed:

- Autonomous Car Coordination (ACCORD) and
- Mixed Integer Interactive Planning (MINIVAN).

Both operate in all three use cases but have different strengths. ACCORD was developed as a coordination approach among CAVs that is also capable of interacting with human-driven vehicles. MINIVAN was mainly developed as a multi-agent planner interacting with human-driven (uncontrolled) agents.

A general assumption here is that other agents do not behave destructively, e.g., do not aim for collisions. Each agent has an individual goal that can be conflicting with other goals, but no adversarial behavior is shown. The algorithms explicitly model that not all agents are interested in cooperation but can show egoistic yet reasonable behavior. This assumption is a distinction to air traffic scenarios, where either perfect cooperation is given or in military use-cases hostile agents are modeled. Our solution algorithm aims for a solution close to the global optimum. This global optimum will often result in a sub-optimal motion of an individual vehicle, e.g., waiting for another vehicle to pass. A non-cooperating vehicle cannot be forced to follow this individual sub-optimal plan and might take egoistic actions but does not behave entirely destructive.

Reactive Behavior with Respect to a Given Prediction

In structured road environments, where the solution space is not limited or conflicting, such as highways or roads with fixed lane assignments, it is often sufficient to be reactive to a motion prediction or a communicated motion of other road users. In this case, the algorithm does not aim to influence the other traffic participants with the ego planning; may these be fellow autonomous vehicles or human-driven vehicles. It is sufficient not to plan an interactive joint behavior for all vehicles but to follow the ego plan and react to the predicted motion of other traffic participants.

Example scenarios are tracking a single lane with a desired speed while avoiding collision with a vehicle in front. Also, in loose highway traffic without oncoming vehicles, for overtaking a slower vehicle, it is sufficient to know the speed deviations between the involved vehicles and plan the own motion accordingly. Furthermore, at signalized intersections with known state transitions of the traffic light, barely dense interaction with other traffic participants is necessary, and a

passive reaction to the prediction of prioritized traffic participant's motion is sufficient to handle the scenario.

Coordination of Connected Automated Vehicles

Autonomous vehicles also need to operate safely in scenarios in scenarios only having autonomous vehicles. In this case, it is valid to assume that these are also communicating to agree on a common plan, which is a coordination problem among CAVs. Here, the algorithm needs to plan a joint behavior where the selected ego behavior also influences the behavior of all other vehicles.

Coordination can yield enormous benefits in various scenarios. Platooning can help to safely improve the throughput on highways while saving energy. Also, with an automated intersection management system the efficiency of an intersection can be improved by coordinated motion of agents. Another use case is fully automated valet parking where a higher density and throughput than in classic parking lots is achieved by orchestrating the maneuvering vehicles.

Proactive Behavior in Mixed-Traffic Scenarios

This work also examines interactive mixed-traffic scenarios where one or more autonomous vehicles are interacting with one or more human-driven vehicles. The algorithm plans a joint behavior for all vehicles with the behavior of the uncontrolled vehicles serving as prediction of the agents. Here, explicitly non-cooperating and selfish agents that do not aim for a globally optimal solution of the traffic scene are addressed. The ego agent has to follow a proactive behavior plan to achieve its own goals in the traffic scene while still operating safely.

As an example, this work considers (dense) merging scenarios, where the ego agent has to potentially first create a gap before it can successfully merge into another lane. Here it is not sufficient to just rely on a prediction of other traffic participants as this would yield a very passive behavior not achieving the own goal. Another scenario is negotiating a solution in unclear situations, e.g., at a blocked road, where the traffic rules are ambiguous which agent has priority.

2.4 Game-Theoretic Problem Formulation

This work uses a game-theoretic notation to formulate the behavior and motion planning problem. Specifically, the planning problem in this work is formulated as a multi-agent dynamic game. Another common formulation is sequential game. A differential game is a dynamic game with continuous time. Following LaValle [LaV06] and Başar and Olsder [BO98], a differential game is defined by the following properties:

1. A set of n agents[1] $A \in \mathcal{A}$. Each agent A^i operates on a continuous *state space* $\mathcal{X}^i$.

2. Each agent A^i has a finite, nonempty set of actions U^i, called *action space* of A^i. The action space can either be discrete or continuous.

3. The game is non-zero sum. The *cost function* $J : U^1 \times \mathcal{X}^1 \times \cdots \times U^n \times \mathcal{X}^n \to \mathbb{R}$ takes into account each agent (joint cost function). The cost function is used to control the level of cooperation.

4. The game can either be cooperative or noncooperative.

5. For each agent A^i, a deterministic and known state transition function exists $f : \mathcal{X}^i \times U^i \to \mathcal{X}^i$.

[1] In game theory literature, *agents* are also referred to as *players*.

6. The states of all agents are known. For the ego agent, the state is received from a localization component. For all other agents either a suitable perception and data fusion pipeline exists or the states are transmitted via explicit communication.

7. The environment[2] prohibits certain states of the agents and therefore interferes with the game. Prohibited states are states that are either colliding, physically not admissible, or prohibited as, e.g., leaving the road. Also, static and dynamic obstacles can be part of the environment.

Using this formulation, multiple decision makers are present, referred to as agents. If the environment contains dynamically moving obstacles, formally also the environment can take decisions. The scene then consists of all individual states, the road geometry, and obstacle information. Each planning instance is formulated over time as one step of the game. Solving the steps of the game over time with knowledge from the previous steps can then be viewed as a receding horizon solution strategy. This work discretizes the time horizon of one planning iteration into N steps with a time interval Δt. The discrete timestep is denoted by k and the interval of N steps by $\mathcal{K}$. The goal is thus to find a sequence of actions for the ego vehicle that minimizes its costs while following the constraints of the game such as avoiding collisions with the environment. A special form of differential games are linear differential games. Here, a linear function exists translating states and actions from one timestep to the next

$$\frac{d}{dt}\begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix} = \mathfrak{A}\begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix} + \mathfrak{B}\begin{pmatrix} u_1 \\ \vdots \\ u_n \end{pmatrix} \tag{2.1}$$

for states $x_i \in \mathcal{X}_i$ and actions $u_i \in U$ and constant matrices $\mathfrak{A}$ and $\mathfrak{B}$.

The continuous action formulation in Chapter 5 is expressed as a linear differential game due to the linear property of constraints in the model. The discretized actions in Chapter 4 form a differential game, as an arbitrary (nonlinear) function can be used to map states from one timestep to the next. Note that in some definitions a differential game also requires continuous actions, which is not the case in Chapter 4. Also, the system dynamics are in this work not described by a pure set of ordinary differential equations but differential equations with continuous and integer constraints on the states. Some differential game definitions require ordinary differential equations.

2.5 Mixed-Integer Programming as Solution Approach

This thesis uses MIP as a solution strategy for the formulated (differential) game. MIP is an optimization strategy that optimizes discrete and continuous decision variables alongside with the ability to converge to the global optimum. In this section, we will briefly introduce the theory. Various books and resources are devoted to this topic, e.g., [CBD11; SS15; Int21a].

Basics of Mixed-Integer Progamming (MIP)

A mixed-integer program is a linear program where a subset of decision variables x can only take integer values. Formally, this can be written as

$$\text{minimize } f(x) \tag{2.2a}$$
$$\text{subject to } A_{\text{eq}}x = b_{\text{eq}} \tag{2.2b}$$
$$A_{\text{iq}}x \leq b_{\text{iq}} \tag{2.2c}$$
$$l_{\text{lb}} \leq x \leq l_{\text{ub}} \tag{2.2d}$$

[2] In game theory literature, *environment* is also referred to as *nature*.

with appropriate matrices A_{eq}, A_{iq} and vectors b_{eq}, b_{iq}, l_{lb}, l_{ub}. The subset $x_{\text{int}} \subseteq x$ of all decision variables $x \in \mathbb{R}^n$ can only take integer values $x_{\text{int}} \in \mathbb{Z}^n$. If the objective function f (2.2a) is a linear function of decision variables $f(x) = qx$ the problem at hand is called a Mixed-Integer Linear Programming (MILP). If the objective function is quadratic $f(x) = x^T Qx$ in the decision variables, the problem is referred to as a Mixed-Integer Quadratic Programming (MIQP), with an appropriate vector q or matrix Q. All constraints (2.2b) and (2.2c) are linear and form a set of equality and inequality constraints. (2.2d) bounds the decision variables between an upper and lower limit – theoretically these limits can be infinite. Geometrically spoken, without integer constraints, the set of constraints forms a convex polyhedron in a n-dimensional space that is constrained by hyperplanes. A corner of this polyhedron is the optimal solution of this so-called LP-relaxation of the MIP. With active integer constraints, only a set of integer points inside the polyhedron are valid solutions of the optimization problem (2.2). LP-relaxations play an important role in some efficient solution algorithms. Mixed-integer programs can be infeasible (the constraints do not form a valid polyhedron), infinite (the polyhedron is open), have exactly on optimal solution, or have multiple or infinite optimal solutions.

Solution Algorithms for Mixed-Integer Progamming (MIP)

Most mixed-integer programs are NP-hard and therefore only algorithms exist that solve the problem in exponential time in the worst case. In practice, with efficient modern solution algorithms the optimal solution can be found fast. MIP was first defined in the 1940s, since the 1970s early computer algorithms exist to solve medium-scale problems. Beginning from the 1990s, modern solution algorithms and efficient implementations to solve MIP exist. Since then, mature commercial solvers have evolved, and an enormous speedup in solution time can be observed [Bix12].

Two classes of solution algorithm exist, heuristic and exact algorithms. For exact algorithms a proof is available, that the optimal solution is found or that the problem is infeasible or unbounded, given infinite solution time for the algorithm. Popular algorithms are Cutting Planes, Branch-and-Bound or the combination to Branch-and-Cut.

The Cutting Plane algorithm first computes the LP-relaxed solution using the Simplex Algorithm. This yields a dual (upper) bound on the solution. Then iteratively new inequalities are added, that exclude the LP-relaxed optimum but do not exclude integer points, a so-called cutting plane. If an integer solution is found, this is the optimal solution of the mixed-integer program.

The Branch-and-Bound algorithm in its general form tries to enumerate all integer solutions of the problem in a tree structure. The root of the tree is the LP-relaxed solution of the problem. The algorithm then branches by splitting the range of values for one decision variable into several sub-problems, and iteratively again solve LP-relaxations and branch until the optimal integer solution has been found. Using the dual bound available after solving one node in the tree, branches of the tree can be cut off (bounded), as these cannot lead to the optimal solution. The combination of Cutting Planes and Branch-and-Bound is called Branch-and-Cut. This algorithm has proven to be very efficient and is used in most modern solvers.

An integer-feasible solution, that is not necessarily optimal is referred to as primal solution. This yields a lower bound on the optimal solution. Using primal and dual solutions, the optimality gap can be computed as the distance of primal and dual without knowing the exact optimum. With this gap, the possible improvements of a primal solution can be quantified and algorithmically proven that an integer solution is optimal.

Heuristic solution methods cannot offer these theoretical guarantees but are in general very fast to apply. Often, the performance of specific heuristics is highly dependend on the problem at hand. An example of a heuristic is exploring the neighborhood around a known integer solution to find more, potentially better solutions. Integrated into a Branch-and-Cut algorithm, heuristics can find primal or dual solutions faster than pure cutting and as the solutions are integrated into the Branch-and-Cut tree the proof of optimality is still possible.

Modern solvers are general-purpose solvers, but often, as we will see in this work, the problem formulation and solver parameterization are of high relevance for a good performance in practice.

Behavior and Motion Planning Based on Mixed-Integer Progamming (MIP)

We in this work develop two different MIP formulations for the multi-agent behavior planning problem, each providing specific benefits in specific use cases. In Chapter 4, a discrete action space is used, meaning that the optimizer chooses from a discrete set of actions to translate from one time instance to the next. This discretization highly narrows the solution space and by appropriate selection of the possible actions the optimization can be guided. This approach has the downside that every discrete action has to be enumerated and if the true optimal action is not among the enumerated action set, it cannot be chosen by the subsequent optimization. Chapter 5 introduces a continuous action space formulation for the multi-agent planning problem, where the optimizer may choose an action on a continuous scale. This comes at the price of a far more complex optimization problem to solve, but the solver will theoretically find the global optimal solution for the given scenario.

Generally, trajectory planning for multiple agents in the presence of holonomy and differential constraints is considered NP-hard, or even PSPACE-hard [VJ16; Yu16; SA20; Pad+16]. Even the underlying, optimal solution of the single-agent constrained path planning problem is known to be PSPACE-hard [Rei79]. With assumptions on the problem formulation, for the path planning problem polynomial-time algorithms are known. For the optimal solution of the single-agent trajectory planning problem no such formulations are known [Pad+16], which also translates to the multi-agent case. This work aims to find an efficient formulation for the multi-agent behavior and motion planning problem alongside with an effective optimal solution algorithm that solves generic multi-agent planning scenarios in tractable time.

3 State of the Art

This chapter will discuss related research activities, first in the field of planning, second regarding hard- and software for autonomous driving research vehicles. This chapter will discuss, that there are already several cooperative behavior planners, also based on Mixed-Integer Progamming (MIP), available in the literature. However, most are developed and tuned for one specific traffic scenario and conceptional or technical limitation prevent the approaches from general applicability. Deploying these algorithms to prototype vehicles to assess if the plans are dynamically feasible and executable is hardly done in literature, especially not for MIP-based planners.

In Section 3.1, an overview of different behavior and motion planning algorithms is given, approaches focusing on cooperative planning are discussed in Section 3.2. As the algorithms developed in this work are based on MIP, Section 3.3 is devoted to related approaches based on MIP and their restrictions. Parts of this summary of the state of the art have already been published by the author [KK17; KK19; EKK20; KEK20].

The second part of the chapter starts with the discussion on the hardware setup of autonomous driving prototype vehicles and how the setup changed over the years in Section 3.4. Section 3.5 concentrates on full-fledged open-source software solutions for autonomous driving, mainly Apollo [Bai17] and Autoware [Aut21a]. As we chose to integrate Apollo in our research vehicle, Section 3.6 reviews Apollo with a special focus on the integrated planning and control system. Parts of this review of the related work have already been published by the author [Kes+19; KEK22].

3.1 Behavior and Motion Planning

The software component to plan the motion is an integral part of each autonomous vehicle. Recent reviews on planning and decision making [Gon+16; Pad+16; Pen+17; SAR18; Yur+20; Cla+20] give an overview of different technologies and planning system architectures.

This sections follows the categorization of Schwarting et al. [SAR18], grouping approaches into three categories: sequential planning, behavior-aware planning, and end-to-end driving. Figure 3.1 outlines the three categories and the embedding of the different components into a sense-plan-act framework, that is widely used within the robotics community. The algorithms developed in this work fall into the group of behavior-aware planners.

Sequential Planning

Classical planning architectures follow a component-based hierarchical, sequential structure of decoupling high-level planning, prediction, and motion planning. They operates on a world representation received from a perception pipeline and potentially also from a Vehicle-to-Everything (V2X) communication component. The winning teams in the Defense Advanced Research Projects Agency (DARPA) Urban Challenge [Mon+09; Urm+08; Rei+09] all applied a three-level hierarchical planning architecture. The prediction of other vehicles' motions is decoupled from the ego-motion planning, which makes the planning process non-interactive.

On a high level, a global navigation route is planned within a road network. This step is followed by a strategic planner that takes local, tactical driving decisions, such as selecting a lane. Here, also other road users are taken into account. A subsequent motion planner generates a comfortable and collision-free trajectory from the strategic plan. This separation of tasks works well in structured, mainly static environments with a limited number of other road users. [SAR18; Gon+16]

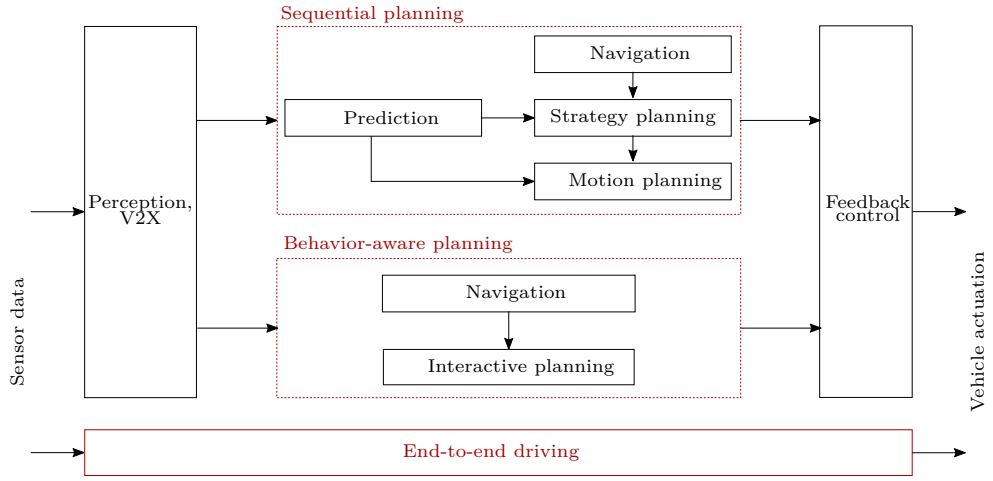


Figure 3.1: Overview of three different paradigms for planning and decision-making: sequential planning, behavior-aware planning, and end-to-end driving.

Navigation Mostly, the navigation (mission planning) is done on a graph representation of the road environment. Classical graph search algorithms are well-suited for small- and medium-scale problems but get impractical for big road networks. [Bas+16] give an overview on practical route planning in transportation networks, not limited to the navigation of vehicles. This work does not evaluate navigation approaches but assume a suitable navigation component.

Strategic Planning The strategy planner is expected to track the reference line at a target speed with respect to the road environment and to account for other road users. In the DARPA Urban Challenge, most teams relied on hand-crafted finite state machines [Mon+09; Urm+08; Rei+09] interpreting the current traffic scenario and trigger transitions based on simple measures such as, e.g., time to collision, individual traffic rules, or distance to a stop line. However, this approach cannot account for complex situations. Modern approaches account, e.g., for the uncertainty in other vehicle's motions using Partially-observable Markov Decision Processes (POMDP) [UM13] or use deep neural networks [Len+17]. Strategic planning heavily relies on the accuracy of the motion and intention prediction. Lefèvre et al. [LVL14] present a survey.

Motion Planning Motion planning for mobile robotic systems has been around since decades. Early approaches split path and velocity planning [KZ86], but such approaches require over-approximations and simplifications to work well in dynamically changing environments and hence do not efficiently use the solution space. Classical trajectory planners can be divided in four groups: Graph-search, sampling-based, curve interpolation, and optimal control-based planners.

Prominent examples of graph-search planners are state lattice planners [WZT10], that generate discrete trajectory alternatives in a local (Frenet) frame as polynomial functions, rate them according to some optimization criterion, and map those back to the global frame again. Also, the A* algorithm [Dol+08] and its variants are based on graph search, enhanced by heuristics.

A prominent example of a sampling-based planner is the RRT* algorithm [KF11]. The runtime of sampling-based planners is extremely dependent on the chosen discretization. Sampling-based planners can be very fast even in high-dimensional configuration spaces but yield suboptimal, non-smooth solutions due to the (randomly) sampled exploration of the configuration space. Gammell and Strub [GS21] present an overview on (asymptotically) optimal sampling-based path planning. The review states properties, restrictions, and assumptions of state-of-the-art algorithms.

Curve interpolation planners rely on a series of waypoints and fit a smooth function, such as a Bézier curve or a cubic spline to theses points. While producing smooth results, the trajectories

rely on the quality of the input points and can become costly when avoiding an obstacle intersecting the waypoints.

Optimal control methods incorporate a model of the kinematics, which is propagated for a given planning time horizon, and usually formulate constraints to account for feasibility and safety while constructing a cost function to account for comfort and other desired aspects. Optimal control provides deterministic solution algorithms converging to an optimum and has been successfully applied to safety-critical systems. Local numerical optimization also operates on a series of waypoints as a reference. At every timestep an optimization problem is solved that generates a smooth and kinematically feasible motion along the reference and takes the road and obstacles into account. Most optimization formulations require a convex problem formulation for fast convergence. Therefore, in real-world scenarios often heuristics are applied to generate a (set of) convex subproblem [Ben+15; AD18; Est+18]. Such algorithms have proven to operate safely under real-time conditions [Gut+16; Zie+14a; Zie+14b]. Recently, also deep learning comes up as a technology for motion planning [HRK19].

Behavior-Aware Planning

Behavior-aware planning approaches are often referred to as interactive planning. Here, the behavior prediction of other traffic participants is coupled with the decision making and motion planning of the ego vehicle. Time-invariant decisions, such as a navigation goal on a map, is often assumed to be an input to the behavior planner. Game-theoretic approaches, learning-based approaches, and probabilistic approaches are popular in the literature. In Table 3.1, we set a selection of different approaches in contrast that are discussed in this and the following sections. It is easily observed, that the algorithms vary in terms of the suitability for specific traffic scenarios, the usage of communication and the presence of non-communicating agents, and the applied solution methods. Cooperative approaches, see Section 3.2, are one approach to plan a behavior-aware motion.

Game-Theoretic Approaches Game-theoretic formulations can be employed to model collaboration, often realized by a decision tree. The game is often solved using iterative algorithms operating on a discretized action or state space. Even if converging to a Nash equilibrium, the result will often be only sub-optimal. Monte Carlo Tree Search (MCTS) can be used to plan collaborative behavior, effectively solving a multi-agent, non-zero-sum dynamic game [LKK16]. A cooperation factor serves as a tuning parameter in the ego agents' cost function. The formulation is highly flexible and can incorporate any transition function for modeling the environment. An extensive-form game is formulated in [Bah+16], where the other traffic participants are modeled as part of the environment. While the framework is highly flexible and has proven to work in a real car under real-time requirements, the approach does not ensure convergence to an optimal solution. A two-player dynamic, non-zero-sum game is formulated as a bimatrix game in [LL20], which allows for an efficient calculation of the Nash equilibrium. Schwarting et al. [Sch+21] use iterative dynamic programming to solve a multi-agent dynamic, non-zero-sum game. They explicitly model partial observability of the intention of others. The proposed believe-space variant of the iterative Linear Quadratic Gaussian (iLQG) algorithm can be executed in real-time. Exact costs and dynamics of other agents are assumed to be known, and the algorithm converges to a potentially sub-optimal Nash equilibrium. Constraints are often treated by introducing high penalties in the cost function and therefore there is no guarantee for convergence. Multi-vehicle driving as a potential game is formulated in [FG20] and solved using Mixed-Integer Quadratic Programming (MIQP). The potential function allows the authors to compute a ϵ-mixed-integer Nash equilibrium, a driving strategy, that is almost individually optimal with respect to constraints. However, the discrete lateral action and state space complicate applying this approach in reality. They use a double integrator model in longitudinal direction and discrete actions in lateral direction, e.g., for

Table 3.1: Overview of a selection of related approaches for interactive planning illustrating different strategies to incorporate interaction and solution algorithms. We distinguish the demonstrated scenario, how interaction is handled, if explicit communication is involved, and the main solution algorithm.

Reference	Scenario	Interactiveness	V2X	Solution approach
Schwarting and Pascheka [SP14]	Highway	Conflict detection and recursive resolution	no	Behavior prediction using motion primitives
Bahram et al. [Bah+16]	Highway	Two-player dynamic game	no	Alpha-beta pruning
Liniger and Lygeros [LL20]	Racing	Multi-agent dynamic game, non-zero-sum	no	DP solving Nash equilibrium
Schwarting et al. [Sch+21]	Racing	Multi-agent dynamic game, non-zero-sum	no	Iterative DP solving Nash equilibrium
Fabiani and Grammatico [FG20]	Highway	Potential game	no	Iterative Nash equilibrium solved via MIQP
M. Wang et al. [Wan+19]	Racing	zero-sum two-player game	no	Iterative best response
Eilbrecht and Stursberg [ES17]	Overtaking	Global cooperative costs, shared plans	no	MIQP with iterative conflict resolution
Manzinger and Althoff [MA18]	Several	Centralized planning, shared plans	no	Reachability analysis with auction-based conflict resolution
Frese and Beyerer [FB11]	Several	Global cooperative costs, centralized	no	MILP
Kurzer et al. [KEZ18; KZZ18]	Highway	Cooperative planning	no	MCTS, reinforcement learning
Hubmann et al. [Hub+17; Hub+18]	Highway, Intersection	Intention prediction with uncertainty	no	POMDP
Rodrigues de Campos et al. [RFS13; Rod+17]	Intersection	Control scheduling periodization rule	yes	Decentralized sequential local optimization
Murgovski et al. [MCS15]	Intersection	Centralized resolution of conflict space	yes	Spatial optimal control for all crossing sequences
Shen et al. [She+15]	Highway	Distribution motion coordination	yes	Graph search
Düring and Pascheka [DP14]	Highway	Decentralized decision making	yes	Maneuver precalculation, reduction, combination
Peng and Akella [PA05]	Unstructured	Collision segments identification and scheduling	yes	Centralized MINLP
Rewald and Stursberg [RS16]	Intersection	Negotiation of driving plans	yes	High level auction-based control, low level MPC
Lehmann et al. [LGW18]	Several	Conflict detection and maneuver negotiation	yes	Right of way, communication of desire and plan
Burger and Lauer [BL18]	Highway	Centralized optimization with joint cost function	yes	MIQP

lane changes where collision freeness of the lateral action cannot be guaranteed. It operates on straight roads but the model assumptions will not hold for arbitrary road geometries. M. Wang et al. [Wan+19] define a two-player, zero-sum game to solve competitive racing scenarios. In Le Cleac'h et al. [LSM21] the authors propose a real-time capable solver for constrained dynamic games with multiple densely interacting agents based on the Augmented Lagrangian Method. The planner computes a joint motion for multiple agents and converges to a Nash equilibrium. Initializing a Model-Predictive Control (MPC) approach with the open-loop strategies based on the game provides a fast update rate with dense states and mitigates the assumption that the cost function of each agent is known. However, global convergence to a Nash equilibrium cannot be guaranteed for nonlinear and nonconvex constraints (such as collision and dynamics constraints).

Probabilistic Approaches Wei et al. [WDL13] model ramp-merging scenarios as a Markov decision process of high-level actions. The multi-step method first estimates the probability of an agent to perform a certain action, then generates future scenarios, and rates them according to a cost functional. The simplistic Bayes model to predict the future motion of other agents is not transferable to arbitrary scenarios. Bernhard et al. [BPK19] address the problem of uncertain intents of other agents by introducing a risk-sensitive behavior planning. The two-step method first uses deep distributional reinforcement learning in a simulation training environment to learn risk-sensitive actions offline. At runtime, a risk-assessment, using standard metrics, is performed and the respective optimal risk-sensitive action is selected. A popular approach when formulating agent interactions in a probabilistic fashion is a Partially Observable Markov Decision Process (POMDP) where other agents' intentions cannot be observed directly. The approach of Hubmann et al. [Hub+17] assigns an optimal acceleration for vehicles moving on pre-defined paths in conflicting scenarios at unsignalized intersections. Integrating an A* heuristic in combination with Monte-Carlo sampling, also combined longitudinal-lateral maneuvers can be achieved [Hub+18]. For different scenarios different action spaces are chosen to fulfill real-time requirements.

Learning-Based Approaches Furthermore, various learning-based approaches exist to achieve behavior-aware planning. Learning-based methods can generate real-world behavior from data. Besides end-to-end driving, where the agent's future actions are learned based on sensor readings and the behavior and motion planning is completely implicit, also a behavior plan (or trajectory) can be learned based on an environmental model. Generally, learning based approaches have shown superior results over classical handcrafted models to represent human behavior [Bha+20; Len+17; HCL19]. Correlations and dependencies between two human-driven vehicles in the data are accounted for implicitly. Most learning-based planning approaches still separate decision making and motion planning. Also the output trajectories or actions often suffer from nondeterminism and limited explainability. Furthermore, the generalization to during training unseen situations is an open research question [SAR18]. Vallon et al. [Val+17] apply support vector machines to decide on lane changes, Lenz et al. [Len+17] use Gaussian Mixture Models. Inverse reinforcement learning aims to capture the human decision making process by learning the unknown reward function from data [Fer+21], which is a major advantage to manually designed functions. Driggs-Campbell et al. [DGB17] target the challenge for an autonomous vehicle to integrate into mixed traffic by acquiring a model of how human drivers drove in the scenario at hand from highway trajectory data and mimic this behavior in the trajectory planning.

End-to-End Driving

End-to-end driving approaches do not follow a sense-plan-act structure. They do not explicitly create a world model and a trajectory but usually rely one deep learning technologies to generate vehicle control inputs directly from sensor data. Data is either gathered from (manual) drives

[Cal+18] or from simulation [Wol+17]. Some demonstrations on real roads exist [Ken+19; Wan+21], but safety and generalization, including verification is still an unsolved issue.

3.2 Cooperative Planning and Coordination

Cooperative planning and the coordination of multiple agents is a special case of behavior-aware planning. Cooperation does not necessarily have to involve explicit communication, but communication among Connected Autonomous Vehicles (CAVs) rules out most of the uncertainty in other vehicles' behavior. Vehicle-to-Vehicle (V2V) communication and V2X communication offers new possibilities and potentially enables an improved traffic flow. We follow the distinction of Ulbrich et al. [Ulb+15] dividing cooperative driving into explicit inter-vehicle communication and cooperation in the form of collaboration. Mertens et al. [Mer+20] distinguish into active and passive cooperation. This work gives an overview of various, also non-technical aspects of cooperation and communication.

3.2.1 Cooperative Planning Among CAVs

Aramrattana et al. [Ara+15] structure cooperative driving along three axis: the number of involved agents, the complexity of the driving task, and if the goals of the agents are within a limited (local) or broad (global) scope. Also, L. Chen and Englund [CE16] review intersection management including motion planning strategies without accounting for human-driven vehicles. All papers stress the opportunities of cooperative driving, such as improving traffic safety and efficiency, but also identify challenges. These are, among others, the penetration of V2X communication infrastructure or the limited capabilities of perception frameworks to estimate the intent of a human driver in complex situations. Most algorithms are limited to vehicles with the same level of automation and connectivity, which is limiting in the expected mixed-traffic scenarios. Also, often resolution algorithms are specific to a class of scenarios without the possibility to transfer to generic situations. Rios-Torres and Malikopoulos [RM17] give an overview of coordination approaches for CAVs in merging and intersection scenarios. Human-driven vehicles are not considered here. The Grand Cooperative Driving Challenges [Nun+12; Ara+18] proved that with a very heterogeneous vehicle setup, cooperative behavior could be achieved relying on standardized communication protocols. As well, platooning and intersection scenarios have been studied.

A scenario-independent conflict resolution strategy among CAVs is proposed by Lehmann et al. [LGW18]. Each vehicle communicates its planned trajectory along with a potentially different desired trajectory. Based on these, other vehicles can adapt their plans without explicit negotiation or acknowledgments. The coordination is computed on a trajectory level in a local Frenet coordinate system without assumptions how these are computed. A cooperative maneuver is executed in three phases: A detection phase followed by a negotiation phase and an execution phase. Rewald and Stursberg [RS16] also propose a hierarchical approach to negotiate driving plans among connected vehicles using an auction-based negotiation scheme, followed by a MPC. In the auction step, the vehicles bid on the precedence over another vehicle. Bids are the increase in costs for a vehicle if it accepts the precedence of another vehicle. The auction approach then minimizes the overall increase in costs resolving all conflicts. The bids are tuned to incentivize cooperation by adding additional cost from previously lost auctions. The possible high-level maneuvers are encoded in a state automaton. The approach does not account for human-driven vehicles. Z. Wang et al. [Wan+18] formulate the coordination of CAVs as centralized optimization and rephrase it as consensus optimization. By decomposition, convexification, and parallelization, a specialized solver computes the solution fast even for a high number of vehicles. Düring and Pascheka [DP14] base the coordination of CAVs on a set of possible maneuvers represented as motion primitives with associated costs. Based on the exchange of these sets, each vehicle individually selects a cooperative maneuver combination. Merging scenarios are evaluated, also in combination

with non-communicating vehicles. Rodrigues de Campos et al. [RFS13] present a coordination algorithm for automated vehicles at uncontrolled intersections. The proposed control strategy is fully decentralized with positions and velocities to be exchanged among the vehicles. The exact paths of the vehicles are assumed to be controllable. The velocities on these paths are optimized for all vehicles minimizing the deviation to a desired speed with respect to vehicle dynamics and safety constraints. The optimization is performed in a sequential way. The vehicles are ordered and each vehicle adapts its control strategy to the higher-ordered vehicles. As a prioritization strategy the authors use the vehicle's distance to the set of states leading the vehicle into the shared space of the intersection.

3.2.2 Cooperative Planning Including Non-communicating Vehicles

Kurzer et al. [KZZ18] take into account non-communicating vehicles. Here, maneuver primitives are used to compute a coordinated motion of vehicles without communication using MCTS. The interaction is modeled using a joint reward function with scaling factors for egoistic or cooperative behavior. The authors also extended their work to continuous state spaces [KEZ18]. The ego vehicles' prediction of the other vehicles cooperating influences the cooperative character of the solution. However, the algorithm is only applicable to a limited set of scenarios. Schwarting and Pascheka [SP14] propose to first plan egoistic maneuvers neglecting other vehicles' interests, afterwards possible conflicts are detected and recursively resolved, starting from the furthest vehicle. The basic driving strategies are assembled from motion primitives. In a structured environment, like the investigated highway scenarios, these primitives cover the most likely vehicle actions and shrink the planning space. By predicting the intention of other vehicles the need for inter-vehicle communication is avoided. Evaluating a cost function the best possible conflict-free maneuver combination is selected. The conflict resolution algorithm is proven to be real-time capable and demonstrated in a real-world highway scenario. However, the approach cannot be directly transferred to arbitrary scenarios and the ordering of vehicles to resolve conflicts can yield sub-optimal results. Eilbrecht and Stursberg [ES17] formulate a two-layered approach of iterative conflict resolution using a cooperative cost function. The underlying behavior of each agent is generated through an optimal control problem using MIQP for each agent while ensuring obstacle avoidance to the (known) plan of the other agents. However, their approach is only valid for straight driving on straight roads, and the iterative conflict resolution does not offer any guarantees to converge to a global optimum. Manzinger and Althoff [MA18] use reachability analysis to compute conflicting space-time cells, which might be occupied by multiple vehicles. An auction algorithm then solves for those conflicts. Their approach requires a discretization of the state space. Viana et al. [Via+21] model the uncertain willingness of a human-driven vehicle to cooperate as POMDP, where the cooperativeness is a non-observable state. MCTS is used as a solution method. The driver and vehicle models are parameterized using observations and an additional yielding classifier is introduced to model the probability that a human-driven vehicle will cooperate (yield) to the autonomous vehicle. Within a dataset and from closed-loop simulations the cooperativeness is demonstrated in lane-merging scenarios with three agents. The used car-following models are parameterized highly usecase-specific and plan no lateral motion, which prohibits the usage in arbitrary scenarios. Toghi et al. [Tog+21] use end-to-end learning instead of explicit driver models to implicitly account for the willingness of a human driver to cooperate in the behavior planning. A multi-agent reinforcement learning strategy, the synchronous Advantage Actor-Critic algorithm is trained on a discrete high-level action space. The effectiveness is demonstrated on a simulated highway environment with human-driven vehicles modeled using classical car following and lane-change models. Therefore, the learned behavior is not based on human behavior but rather model parameterization. The authors claim that the methodology is transferable. In the evaluated scenarios introducing cooperation in the behavior planning process

greatly increased the success rate and decreased failures (such as collisions) compared to purely leaning to drive egoistically on the highway.

3.3 Planning Using Mixed-Integer Programming

Several competing technologies exist to in the end compute a interactive and cooperative behavior, one is optimal control. Complex situations with densely interacting agents cannot be solved to optimality using decomposition techniques from sequential planning and often handcrafted, scenario-specific solutions are applied. One possibility to overcome these limitations is MIP, that offers multiple benefits that are favorable for optimal behavior planning such as convergence to a global optimal solution and the alongside optimization of discrete and continuous variables. The work relies on MIP to solve the behavior-aware planning problem. Besides for motion planning (e.g., [Qia+16]) and behavior planning (e.g., [ES17]), MIP-based strategies are also applied for traffic management (e.g., [LR17]).

Motion and Behavior Planning

Schouwenaars et al. [Sch+01] model the multi-vehicle trajectory planning problem with obstacles as Mixed-Integer Linear Programming (MILP). All vehicles are assumed to be cooperative. The vehicle dynamics are incorporated into the optimization problem in a linear and discrete fashion. This indicates, that the applied vehicle model is only valid for holonomic motions. Safety distance constraints between pairs of vehicles are formulated separately in x- and y-direction, which will lead to infeasible problems in arbitrary on-road settings. Incorporating uncertain measurements, actuator disturbances and model uncertainties a MILP approach to optimize the trajectory of one vehicle in the presence of obstacles is formulated by Blackmore et al. [BOW11]. A chance constrained framework keeps the failure probability below a threshold. With bounds on the collision probability the chance constrained non-convex problem is converted into a linear convex one. Frese and Beyerer [FB11] propose a double integrator-based MILP formulation, with constraints ensuring a collision free motion inside the road boundaries. All vehicles are assumed to be cooperative. In the observed crossing and overtaking scenarios solving the MILP takes varying computation time, so the real-time applicability is questionable. Nevertheless the algorithm shows good results in terms of planning success, but tends to become infeasible due to conflicting constraints for more than three vehicles. The non-holonomy is assured by bounding an approximation of the lateral acceleration. However, that is only valid for small yaw angles, and yields a lot of invalid solutions [Fre12]. The collision checks are modeled with an arbitrary road polygon using a disjunctive collision check with convex polygons. They also propose to check for collision using rectangle-based vehicle-approximation of consecutive states, with each variant introducing a lot of invalid trajectories [Fre12]. Qian et al. [Qia+16] apply the double integrator to MIQP. Similar to Nilsson et al. [Nil+16], they model the non-holonomy by bounding the lateral velocity, which is only valid for straight roads and if the vehicle is road-oriented. Even when avoiding an obstacle, the yaw angle may exceed 20 degrees, which can yield bends in the optimized trajectories that cannot be executed with a real vehicle. The work is thus limited to straight roads and straight driving, whereas turning at intersections is not possible. With the quadratic cost function of a MIQP, differences to longitudinal or lateral values (such as acceleration) are only possible if the reference signal is zero (such as for acceleration). Thus, deviations from the absolute velocity cannot be expressed. Burger and Lauer [BL18] extend the work of Qian et al. [Qia+16] to the cooperative setting by extending the state space to multiple agents. A joint cost function is optimized using mixed-integer quadratic programming and avoids collision on a vehicle motion level. However, also the limitations of the formulation are inherited. Nilsson et al. [Nil+16] introduce two Quadratic Programmings (QPs) for longitudinal and lateral control based on a linear double-integrator model. To account for non-holonomy, the authors use a linear inequality constraint that couples lateral and longitudinal

Table 3.2: Comparison of MIP model approaches. We express linear functionals using ① , quadratic using ② , and nonlinear functionals using ⓝ . The counterparts ① , ② , and ⓝ express functionals that are not (and cannot be) implemented. We compare cost functionals that we assume to be desirable, with cost terms $j_\square$, curvature κ, velocity v, and acceleration a.

Source	Ziegler et al. [Zie+14a]	Gutjahr et al. [Gut+16]	Nilsson et al. [Nil+16]	Qian et al. [Qia+16]	Frese and Beyerer [FB11]
Problem formulation	SQP	QP for long, lat each	QP for long, lat each	MIQP	MILP
Model	Triple integrator	Frenet bicycle model	Double integrator	Double integrator	Double integrator
Reference frame	Cartesian, fixed	Frenet, street-wise	Cartesian, fixed	Cartesian, fixed	Cartesian, fixed
Non-Holonomy Constraint	κ, ⓝ	κ, ①	v_x, v_y are coupled, ①	v_x, v_y are coupled, ①	a_{lat}, ①
Acceleration Constraint $\lvert a \rvert$	②	①	ⓝ	ⓝ	approximated, ①
Collision Shape	circles	circles	road-aligned rectangle	road-aligned rectangle	road-aligned rectangle
Collision Check to	everything	everything	road-aligned rectangle	road-aligned convex polygon	non-convex road polygon
Collision Constraint	ⓝ	①	①	①	①
j_{velocity}	ⓝ	②	ⓝ	ⓝ	ⓝ
$j_{\text{acceleration}}$	②	②	②	②	① , ②
j_{jerk}	②	②	②	② , [BL18]	②
j_{yawrate}	ⓝ	② using $\dot{\kappa}$	ⓝ	ⓝ	ⓝ
$j_{\text{reference}}$	ⓝ	②	ⓝ	ⓝ	ⓝ
Validity	any road / orientation		straight road, aligned with road (orientation-wise)		
Multi-Agent	no	no	no	yes, [BL18]	yes

velocity. However, this is only valid for small yaw angles and will yield non-drivable trajectories at intersections and roundabouts for example, since the road curvature is not taken into account. Gutjahr et al. [Gut+16] introduce two longitudinal and lateral QP in the Frenet frame based on the bicycle model. The model yields good results for driving in static environments. Similar to Ziegler et al. [Zie+14a], this approach relies on a maneuver selection, as shown by Esterle et al. [Est+18]. However, the transformation of all obstacles (static environment and dynamic obstacles) to local coordinates is costly and with an increasing number of obstacles outweighs the benefits of the fast QP. Miller et al. [MPA18] base their work on [Gut+16] and formulate two consecutive longitudinal and lateral programs using MIQP similar to [Nil+16], but in local street-wise coordinates instead. However, the approach cannot account for any model-based prediction or multi-agent planning. Also separating longitudinal and lateral control yields sub-optimal solutions and limits the solution space. Reiter et al. [Rei+21] extend the motion planning problem by not only avoiding obstacles but also introducing reward regions to guide the optimization into certain states. As, e.g., Esterle et al. [Est+18], in a first step the horizon is split into homotopy classes representing different high-level maneuvers. A MILP is formulated to find the optimal homotopy with respect to the specified rewards. Inside this homotopy, a nonlinear optimization problem minimizing the deviation to the reference and jerk is formulated and solved using Sequential Quadratic Programming (SQP). The effectiveness is demonstrated in an autonomous racing scenario.

Problem Linearization for MIP

A major drawback of MIP is, that only linear constraints can be expressed as mixed-integer nonlinear programming problems with general constraints and cost functions cannot be solved

in real-time. Therefore, all constraints have to be expressed as a linear function which poses high requirements on the constraint formulation. Ziegler et al. [Zie+14a] propose a nonlinear optimal control problem, where they apply the cost functionals and constraints shown in Table 3.2. It analyzes the formulations in terms of linear functionals ① , quadratic ② , and nonlinear functionals ⓝ . We consider these as as desirable and compare other problem formulation based on QP and MIP with respect to the order of the constraint and cost functions. Ziegler et al. [Zie+14a] use a triple integrator as vehicle model, while bounding the curvature to account for non-holonomy. The Bertha-Benz drive showed the applicability of the triple integrator model if correctly constrained. Their approach yields a nonlinear optimization problem, which is solved using SQP but only finds a local optimum. Motivated by that, approaches for maneuver planning [Ben+15; AD18] have focused on finding the correct maneuver in a preliminary step. However, these approaches usually rely on a geometric partitioning of the workspace and thus scale poorly, and cannot be extended to account for any interactive or cooperative planning.

Traffic Management

Besides planning the behavior for a subset of vehicles on a road stretch, MILP is further applied successfully to optimize an overall traffic scenario taking into account every vehicle in a dedicated area. Considering a traffic scenario as a cooperative multi-agent system, the global optimum is optimal traffic flow, e.g., maximizing the throughput or avoiding deadlocks [Pap+03]. Usually these traffic management strategies provide too coarse or not kinematically invalid trajectories to be directly used as a behavior or motion plan but serve more as a reference trajectory planner.

Examining the global traffic network Lin et al. [Lin+10] reformulate a model predictive control problem of an urban traffic network as MILP and demonstrate the applied solution using a macroscopic traffic model. Levin and Rey [LR17] introduce an intersection manager that assigns reservations to each vehicle based on a MILP. The system is based on pre-computed conflict points and the mixed-integer optimizer coordinating the spatial trajectories of the vehicles to be collision free. The policy is in a receding horizon fashion. Ashtiani et al. [AFV18] extend the work of Fayazi et al. [FVL17] on optimal vehicle arrivals at intersections to networks of connected intersections. The applied MILP approach does not account for lateral motion and only uses the arrival time of a vehicle at an intersection as a vehicle-dependent decision variable and the vehicle dynamics are highly simplified. Highway scenarios are addressed in [RD12] trying to maximize the total traveled distance with subject to correct acceleration and overtaking behavior. An optimal vehicle to parking space assignment in a huge network is tackled by Geng and Cassandras [GC13] minimizing each driver's cost function and the proximity to the driver's destination. It is assumed that parking spaces can be supervised and reserved.

3.4 Automated Driving Vehicle Hardware

Despite the rich history of past research vehicles starting in the early nineties, the source code or system specification has rarely been made open-source. Apart from a collaborative approach by Levinson et al. [Lev+11] to summarize lessons learned on the algorithm side, only sparse knowledge has flown back to the community regarding how to build up an architecture including the inevitable pitfalls one will face. To display the different development stages of hard- and software, this section will compare the setup of several selected autonomous driving prototypes We chose to include the vehicles of the two winning teams of the DARPA Urban Challenge and the Grand Cooperative Driving Challenge (GCDC), respectively, as these competitions presented two milestones in autonomous driving research. Additionally, we added selected other vehicles, such as VaMP [TD96] or Bertha [Zie+14b]. Further experimental platforms are also discussed in [Bue+19]. An overview of historic and ongoing research programs is also given by Aramrattana

Table 3.3: Hardware comparison of selected research vehicles from an early demonstrator, vehicles from the DARPA challenges, and the Apollo reference vehicle. We use ♦ to indicate an existing component, ◊ to indicate its nonexistence and ? if such information is not public.

	VaMP [TD96] 1994	Junior [Mon+09] 2007	Boss [Urm+08] 2007	Bertha [Zie+14b] 2013	RACE [Bue+15] 2015	Halmstad [Ara+18] 2016	Bertha [Tas+18] 2016	Apollo [Bai17] 2018	Fortuna [Kes+19] 2019
Camera	front&rear	◊	front	stereo	front	◊	stereo/360°	front	360°
Lidar	◊	♦ 64L	♦	◊	♦ 4L	◊	♦ 4L	♦ 64L	♦ 32L
Radar	◊	♦	♦	♦	♦	♦ series	♦	♦	♦
GPS	◊	♦	♦	♦	♦	♦ rtk	♦ rtk	♦ rtk	♦ rtk
INS	♦	♦	♦	?	♦	♦	♦	♦	♦
RT comp.	♦	?	◊	?	♦	♦	♦	◊	♦
PC	♦	♦	♦	?	♦	♦	♦	♦	♦
GPU	◊	◊	◊	?	◊	◊	♦	♦	♦

et al. [Ara+18]. Table 3.3 depicts the hardware evolution of research prototypes in the past 15 years.

VaMP, developed by Thomanek and Dickmanns [TD96] in the 1990s, was limited to Adaptive Cruise Control (ACC) and lane change applications on highways due to reduced sensor capabilities with only four cameras and computation hardware of approximately 50 processing units. Fifteen years later, a successful choice for teams in the DARPA Urban Challenge was a fusion of high-end lidar and radar sensor data. However, back then, perception systems could not rely on GPU-based acceleration and deep neural networks. Since then, radar and lidar sensors for detection and camera-based systems for classification have been used in urban environments. Recent advances in multi-sensor data fusion using machine learning methods have encouraged to use more advanced sensors and setups. With the Apollo reference vehicle [Bai17], various scenarios have been demonstrated using different sensor setups. The DARPA Urban Challenge established high-precision Global Navigation Satellite System (GNSS) as a standard. Since 2016, systems with Real-Time Kinematic (RTK) have benefited from increased localization accuracy.

Before 2016, research platforms did not focus on inter-vehicle cooperation or communication devices. The first GCDC [Nun+12] in 2016 confronted researchers with cooperative ACC scenarios and introduced a V2V communication protocol. This challenge made appropriate communication devices necessary in the participating platforms. As fusing self-perceived sensor data and V2V data proved to be challenging, Nunen et al. [Nun+12] selected to use the communication interface and a Radar, whereas Tas et al. [Tas+18] selected to use the V2X inputs only. Similar competitors participated in the second GCDC [Ara+18], which also included lateral maneuvers with the need for communication and cooperation.

3.5 Open-Source Software Stacks for Automated Driving

Most research vehicles run a proprietary, individual and customized software stack and a decent comparison is impossible. Still, the setups share some software design properties, as we outline in Table 3.4. Considerations of functional safety are commonly neglected with research vehicles. Nevertheless, watchdogs and sanity checks usually handle algorithm and system errors. Only recently, open-source driving stacks got attention, potentially enabling research groups around the world to solve real-world problems. Driving stack generally denotes the set of all software components that are necessary for fully autonomous driving. A prominent example is Apollo, an open-source project funded and operated by Baidu [Bai17]. A joint project of various Japanese universities has initiated the open-source stack Autoware [Kat+18], that claims similar capabilities as Apollo. The company Apax.ai is working towards the industrialization of Autoware. Both projects expect deployment on a specific exclusive set of supported hardware architectures. To the best of our knowledge only these two comprehensive open-source software stacks with a sufficiently large community are available.

Table 3.4: Software stack characteristics of the discussed vehicle platforms.

	VaMP [TD96]	Junior [Mon+09]	Boss [Urm+08]	Bertha [Zie+14b]	RACE [Bue+15]	Halmstad [Ara+18]	Bertha [Tas+18]	Apollo [Bai17]
Application	German Highway	DARPA Urban Challenge		German Rural	Parking	Cooperative Driving Challenge		Various
Licensing	proprietary	partly open	proprietary	proprietary	proprietary	proprietary	proprietary	open
Middleware	?	publish/ subscribe	publish/ subscribe	?	RACE RTE	LCM	ROS	Cyber RT
OS on PC	?	Linux	?	?	PikeOS	Linux	Linux	Linux
Functional Safety	None	Watchdog module	Error Recovery	?	Supporting ASIL D	Trust System	?	System Health Monitor
Controller on	?	PC	?	?	RACE DDC	MicroAutobox	Real-time comp.	PC

Apollo Baidu's Apollo [Bai17] is an open-source autonomous driving stack with its own customized middleware. The framework has been adapted or used in various works. X. Wang et al. [WRA20] connected Apollo to the motion planning benchmark tool Commonroad [AKM17]. Fremont et al. [Fre+20] used Apollo to demonstrate the simulation to reality gap for a scenario-based testing approach. In Section 3.6, we will in detail describe the Apollo driving stack with a special focus on its motion planning capacities. Apollo has already been applied for various driving experiments in the US and China where algorithms or methodologies have been made publicly available, e.g., [Wan+20; Xu+19; Xu+20; He+21].

Autoware Autoware.AI (formally known as Autoware [Kat+18; Aut21a]) is an open-source software stack for autonomous driving based on ROS. It contains modules for localization, perception, prediction and planning. The localization is based on a High Definition Map (HD-Map), SLAM, and GNSS/Inertial Measurement Unit (IMU) sensors. The perception fuses data from camera and lidar. The planning is based on probabilistic robotics and rule-based systems and offers modules for open space (e.g., parking) and lane following.

Autoware.Auto [Aut21b] is a recent re-implementation based on ROS2 of Autoware.AI [Aut21a] with well-defined interfaces and an additional focus on functional safety. It is developed under the Autoware Foundation. Currently, the new stack supports valet parking but no on-lane driving. With Apax.Autonomy, Apax.ai develops a commercial fork of Autoware.Auto supporting real-time execution and offering production-grade software quality.

Autoware.IO aims to provide a unified interface layer between Autoware and third party software components and hardware-dependent software, such as device drivers. Also, reference implementations and platforms are provided including a test framework.

Apollo and Autoware in Contrast Apollo and Autoware have a similar scope, as both aim to provide a framework and ecosystem in conjunction with algorithms such as planning or data fusion. Raju et al. [RGL19] have compared the functional components between Apollo and Autoware.AI and state that the functional modules in Apollo are more robust and performant at the cost of leaving the generic ROS environment. Apex.AI offers a middleware called Apex.OS that extends ROS2 and promises hard real-time capability. However, studying and comparing potential advantages of the Autoware products over Apollo regarding hard real-time requirements were out of scope for this work.

Other Autonomous Driving Stacks Hardly any other publically available open-source comprehensive stacks exist. The Junior Driving Stack [Mon+09] from the DARPA Urban Challenge is available as open-source software, but not actively maintained since then.

Also commercial solutions are available, that is not discussed in detail here as we focus on open-source frameworks that are available for research. NVIDIA's DriveWorks [NVI21] is a framework for developing driving functions. They also provide reference implementations for perception,

Table 3.5: Comparison of concepts between Cyber RT and ROS2

Cyber RT	ROS2 equivalent	Description
Component	Package	modules
Node	Node	fundamental building block that can read/write messages
Channel	Topic	publish-subscribe concept
Message	Message	data unit for data transfer between modules
Task	Action	asynchronous computation task
Reader/Writer	Publisher/Subscriber	class within node to send/receive messages
Service	Service	synchronous communication between nodes
Discovery	Discovery	finds other service nodes
Coroutines	–	optimize thread usage and system resource allocation
Launch file	Launch file	start multiple modules at once
Record file	Bag file	recorded messages

localization, and planning. However, the stack is not open-source and only supports NVIDIA hardware such as the NVIDIA Drive platform. Further commercial solutions exist on the market, some being available for research. For example, EB robinos [Ele21] claims production-grade, hardware-agnostic implementation of various automated driving components. The Mathworks Automated Driving Toolbox [The21a] offers a set of software components and tools for autonomous driving components.

3.6 The Apollo Driving Stack

As we will discuss in Chapter 6, we chose to base the software setup on the Apollo driving stack. Baidu claims that the stack contains all necessary modules for full autonomy. The stack has a very modular structure and brings its custom-built middleware (Cyber RT) to exchange information. The Apollo stack is embedded in a growing open-source community and many companies have joined the Apollo board.

For a comprehensive description of the Apollo software design and architecture, the reader is referred to the documentation in the open-source repository [Bai21g]. Section 6.2 will give an overview how we used the Apollo driving stack in this work and will describe more software components in detail, such as the perception pipeline. In this section, we will first elaborate on the Cyber RT middleware in Section 3.6.1 and afterwards describe the capabilities of the planning (see Section 3.6.2) and control (see Section 3.6.3) components.

3.6.1 The Cyber RT Middleware

Apollo comes with Cyber RT [Bai21d], a customized middleware for autonomous driving. Like popular robotic middlewares such as ROS or its successor ROS2, it is based on a publish-subscribe principle to interconnect different software components. Table 3.5 compares relevant concepts of both and specifies equivalences. It offers easy-to-use C++ and Python Application Programming Interfaces (APIs) to integrate new components. Similar to ROS2, no central core component has to be launched; each component is launched as a separate task in the userspace. The system is fully distributed. Also, a distribution of components among multiple computers is possible. Cyber RT resolves the components' data input and output dependencies and correctly links the components in a directed acyclic graph. It is also possible to exchange data between components using a service-oriented API rather than the publish-subscribe channels. Components can be triggered from a callback or run at a predefined frequency. Furthermore, Cyber RT offers APIs for parameterization, timing, and logging. The messages can be monitored, recorded, and replayed easily, essentially offering comparable usability to ROS. Each Apollo module runs inside Apollo's Docker container. A bridge component enables communication with components outside this

container. Apollo offers a kernel patch [Bai21b] to Ubuntu to achieve real-time requirements when running the stack.

The message format of Cyber RT is based on Google Protocol Buffers. Messages and parameters have to be first specified in the Protocol Buffers format. From this domain-specific language, an appropriate implementation to access the values is created that is compiled alongside the component's source code. This allows creating components in Python, C++ , and other languages.

3.6.2 The Apollo Planning Module

Apollo's planning module offers several planning algorithms [Bai21a], that we will describe in this section. They are all based on the same interface and take the vehicle ego state and the predicted positions or trajectories of obstacles as an input. A new trajectory planning is triggered for each new prediction message. The code is barely documented but follows modern and understandable coding patterns.

Public Road Planner

The public road planner is designed as a modular and holistic approach to generate suitable trajectories for various kinds of distinct driving situations by decoupling the selection of the current situation from the actual planning algorithm and its parametrization. Based on the scenario the vehicle is currently in, a predefined set of tasks is executed. This modular and flexible implementation comes at the downside of various switching and blending layers in the planner. Four classes of situations can be handled:

1. **Lane Following and Overtaking**: Track the desired velocity in a particular lane or follow the vehicle in front. Also, lane changes and overtaking of slow vehicles are possible.
2. **Intersections**: Handle signalized and unsignalized intersections with or without traffic signs according to the (United States) traffic rules. When approaching a stop sign or performing an unprotected turn at a signalized intersection, the vehicle first comes to a complete stop and then creeps in the intersection to find a safe gap for passing.
3. **Parking**: Maneuver from a lane into a parking spot or reverse. Here the Open Space Planner is triggered.
4. **Emergency**: In case of software or hardware failure, two emergency maneuvers are possible. Either the Open Space Planner drives the vehicle to the curb or the vehicle stops in the current lane while preventing rear collisions.

The actual motion planning is done in a path-velocity decomposed setting. First, the vehicle state is translated into the Frenet frame. For this, a smooth reference line (up to the curvature derivative) is generated from the reference line using QP. Second, the lane boundaries and static obstacles in front of the vehicle are converted into the Frenet frame. Selecting a homotopy to either pass an obstacle on the right or left side is usually a non-trivial task [Est+18]. Apollo solves the homotopy class selection problem with a heuristic search. Third, a QP for the lateral deviation from the reference line is defined. The longitudinal interval on the reference line is set to a fixed resolution. The optimization problem holds constraints to be kinematically feasible, collision-free, and minimizes lateral reference deviation and lateral movement in a piecewise-jerk fashion. For longitudinal movement along the reference line, a similar QP is defined. In this step, moving obstacles are considered by setting an appropriate reference speed for following a vehicle or merging in. Longitudinal and lateral trajectories are then combined and transformed back from the Frenet frame to the Cartesian space [Zha+19; Zha+20]. The public road planner has proven to master various on-road driving scenarios and compute smooth and collision-free results. The computation time is comparably low on standard hardware, and the algorithms do not require specific accelerators.

Open Space Planner

The Open Space Planner is a dedicated module for maneuvering in free-space scenarios without a lane structure, such as parking. The relevant area for maneuvering and the objects possibly interfering with the planned trajectory are filtered in a pre-processing step. As a basis, the Hybrid-A* path planning algorithm is used. In a subsequent step, the curvature of this collision-free path is smoothed using the Dual-Loop Iterative Anchoring Path Smoothing algorithm [Zho+20]. Afterward, a speed profile along the path is generated, formulating a constraint QP that minimizes the jerk [He+21; Zho+20]. The planning algorithm shows a fast, reliable performance in both simulated and real-road scenarios.

State-Lattice Planner

Apollo provides an implementation of a state-lattice planner, that is capable of reference line tracking and obstacle avoidance. The methodology is similar to, e.g., the approach of Werling et al. [WZT10]. First, the reference line and the vehicle position is transformed into the Frenet frame with the origin of the new coordinate system on the nearest point of the reference to the initial point for planning. Then, longitudinal and lateral trajectory bundles are computed individually. These one-dimensional motions are then checked for feasibility with respect to the current vehicle dynamics and then combined to two-dimensional trajectories. These combined trajectories are then sorted by costs, a linear combination of jerk and acceleration costs. Beginning from the one with lowest cost, for each of the combined trajectories is checked, if also this two-dimensional trajectory is within acceleration, jerk, and curvature bounds. If yes, the trajectories are checked for collision with obstacles. No collision check with the environment is performed; it is assumed, that appropriate parameterization ensures that the trajectories stay on the lane. This assumption can yield safety problems and also limits the applicability for lanes with varying geometries.

Limitations of the Apollo Planner

The planning module produces good results in most of the scenarios we tested. Nevertheless, we identified the following shortcomings of Apollo's current planning module:

- Lack of interactive or cooperative planning.
- Lack of transferability to arbitrary situations.
- Trajectories are not C^3-smooth.

We now discuss them in more depth. The planners have an excellent performance in terms of evaluation time. A planning frequency of 100 ms is easily achieved. Due to this property, the planner is reactive to dynamic obstacles around the vehicle. However, it is barely interactive as it avoids the predicted trajectories of dynamic obstacles without the intention to influence or negotiate with others. The applied path-velocity decomposition in the planner also prevents the creation of interactive trajectories.

The public road planner is designed for a specific set of driving maneuver classes, such as lane following or lane changing. For each of those, the detected traffic participants are processed differently before being fed to the planner. Designing this and tuning the triggering and translating between those requires significant engineering effort. No template or generic methodology is yet available to enhance the number of supported maneuver classes.

Apollo's planning algorithms impose constraints on the controller stabilizing the trajectory. For example, the trajectories are not guaranteed to be C^3-smooth, and therefore, the tracking controller has to smooth the trajectory to achieve a comfortable driving behavior.

The shortcomings of the Apollo planner motivated the development of replacing it with a more generic planning implementation that can also plan interactively for multiple agents.

3.6.3 The Apollo Control Module

Apollo's control module [Wan+20] generates a steering and throttle/brake command from a trajectory with respect to the vehicle dynamic status and the localization. It consists of an outer control loop stabilizing the longitudinal and lateral vehicle dynamics and an inner control loop compensates the dynamic delay of the by-wire actuation. The longitudinal, lateral, orientation, and velocity tracking error is used for stabilization.

The outer control decouples longitudinal and lateral control. The longitudinal controller consists of a feedforward term and a sequence of proportional–integral–derivative (PID) controllers that uses the reference acceleration as input. A desired acceleration is then stabilized from the tracking errors and mapped to throttle and brake commands. The steering command is generated using a Linear Quadratic Regulator (LQR) stabilizing the lateral rotation, lateral translation, and tire-friction dynamics using a 4th order model. As an input, the desired heading angle change is used, output of the model is the desired steering angle.

Core of the inner control loop is a Model-Reference-Adaptive-Control (MRAC) algorithm to account for delays in the application of the steering commands. The controller stabilizes the steering angle command and change from the outer loop. The MRAC parameters are tuned online.

The chain of controllers relies on state-of-the-art standard approaches with various controller parameters and gains to be parameterized. Using a learning-based framework, these parameters can be computed and tuned in simulation or from real-world test drives. This include the dynamic model parameters of the vehicle (plant) model, the mapping from high level (acceleration) control commands to low-level (pedal position) commands, and the gains of the PID, LQR, and MRAC modules. The tuning procedure itself depends on parameters, that have to be selected manually. The control module also takes care of handling the vehicle state, such as switching between manual and automatic driving mode.

The framework shows a good performance in simulation without a high-fidelity vehicle model, but heavily relies on a good performance of the tuning framework. Also, the approach is not suited to be executed on a hard real-time platform. As we decided to run the trajectory tracking controller on a real-time device, we did not go through the process of developing a high-fidelity vehicle model for our experimental platform and generate suitable parameters from the auto-tuning toolchain. Therefore, this work chooses to implement a different control approach as described in Section 6.4.

4 ACCORD: Dynamic Games on a Discrete Action Space

Executive Summary of this Chapter This chapter introduces the Autonomous Car Coordination (ACCORD) planning approach, a formulation for the multi-agent behavior planning problem on a continuous state space using discrete actions forming a decision tree. The resulting optimization problem is solved using Mixed-Integer Linear Programming (MILP). We target applications where a coordinated group of Connected Autonomous Vehicles (CAVs) can realize an optimal resolution of a conflicting traffic scene viewed from a global perspective. We also include, non-communicating vehicles into the planning process and still achieve optimal behavior. The algorithm is based on the collaborative assumption, that each traffic participant shows cooperative behavior to some extend and others' actions can be influenced with own actions. Still, we plan appropriate behavior if the collaborative assumption is not fulfilled or the prediction of others' behavior differs much from the observed behavior. In a runtime and complexity analysis we show, that with a limited number of densely interacting agents the approach is real-time capable, but scales exponentially. In various demonstration scenarios, such as a highway, an intersection, and a negotiation scenario, we show that ACCORD is scenario-agnostic, handles different kinds of agents, and effectively leverages the level of cooperation between different agents.

Content and Structure of this Chapter We here introduce a novel approach to coordinate the behavior of multiple vehicles – CAVs and human-driven vehicles – and plan their cooperative behavior and motion in generic traffic scenes. Based on the observed situation and an estimated intention of other participants, their driving behavior is quantified, and an optimized trajectory plan is calculated. The algorithm is sketched in Section 4.1. Our three-step method generates motion options as a tree-like graph structure first, neglecting vehicle interactions (cf. Section 4.2). Each path in the tree corresponds to one possible behavior. Afterward, in Section 4.3, a symmetric MILP is generated and solved to find the optimally coordinated motion patterns. Suitable constraints avoid colliding behavior options and an optimal set of behavior motion options according to previously assigned costs is computed. In a final step, an online re-calibration of cost terms based on the observed behaviors in reality (cf. Section 4.4) is performed. We do not rely on a fixed computation order of all agents or other prioritization schemes. Section 4.6 describes algorithmic modifications to efficiently handle cooperative automated valet parking scenarios. Several implementation and tuning remarks are given in Section 4.5 together with a decent complexity analysis of the problem in Section 4.7. We can then show that cooperative behavior with conflict resolution, as well as egoistic driver intentions, can be handled safely and analyze the properties of the proposed solution. Section 4.8 presents the evaluation of various simulated scenarios and Section 4.9 concludes this chapter.

Contributions of this Thesis This chapter is based on two publications, Kessler and Knoll [KK17] and Kessler and Knoll [KK19]. Here the idea of cooperative optimization on discretized decision trees is introduced alongside with the generation of the behavior options. [KK17] is focused on automated parking scenarios, that are treated as a special case here. Alongside with the evaluation of more driving scenarios, this thesis further contributes several measures to apply the algorithms in a real-time fashion in an on-road setting. This includes the introduction of a strategy to ensure a collision-free motion in-between timesteps and the two-stage network flow optimization

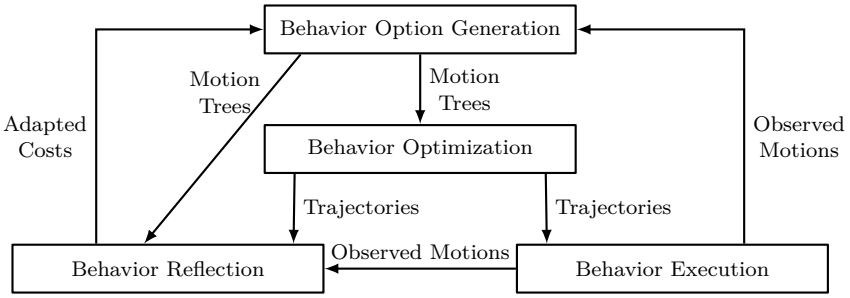


Figure 4.1: Components of the behavior coordination system (graphic from [KK19], ©2019 IEEE)

and iterative collision check solution refinement algorithm. Also, the behavior reflection is described in detail. Compared to [KK19], the optimization horizons for each vehicle have been chosen to have equal length, a tree expansion until a given depth in time in stead of a fixed number of steps, and a simplified cost term calculation. These implementation changes have proven to work more robust in practice and reduce the parameterization effort.

4.1 Algorithmic Idea and Architectural Overview

We develop a behavior-aware, integrated behavior and trajectory planning component that calculates a coordinated motion for all vehicles in the current traffic scene. As we already sketched in Section 2.3, ACCORD was mainly developed as a behavior coordination approach among connected vehicles that integrates into traffic with human-driven vehicles.

The behavior coordination approach consists of four major components. Their interaction is visualized in Figure 4.1 and described in detail in the following subsections. First, a decision tree of reasonable driving options is generated in the behavior option generation step, which is described in detail in Section 4.2. The result of this step is a tree of possible options, one for each agent. These trees are then passed to the behavior optimization step, where a cooperative optimal, collision-free trace for each agent is found as described in Section 4.3. The third and fourth steps are executed in parallel. For non-cooperating vehicles, a behavior reflection set is performed, see Section 4.4. Here, the deviation of the optimal trajectory to the observed trajectory of the vehicles is measured and translated into adapted costs for the optimization. For the ego vehicle and all other CAVs, the generated trace through the tree is transformed into a trajectory, smoothed in a post-processing step to ensure minimal jerk, and passed to a trajectory tracking controller for behavior execution on the vehicle. As our computed trajectories are discretized, we smooth the trajectories using the nonlinear Model-Predictive Control (MPC) introduced in Section 6.3.4 as a suitable low-level trajectory planner keeping as close to the original trajectory as possible. This way, we also minimize the jerk along the trajectory to ensure a comfortable motion. We execute ACCORD in a receding horizon fashion and, therefore, only execute the first steps of the optimized trajectory. We then prune the graph and drop all branches leading to past or unreachable states. The remaining graph is kept as a starting point for the next step. If e.g., all states in the graph turned invalid due to an emergency maneuver from the vehicle controller, the whole graph is dropped and re-initialized for the next planning step. Technically, the proposed algorithm does not distinguish between the ego vehicle and other vehicles. With only one vehicle present, it optimizes the trajectory choice. With more vehicles, it also resolves possible conflicts.

The main drawback of working in a discrete action space is the *curse of dimensionality* as the size of a decision tree grows exponentially with its depth. In Section 4.7 we will discuss how this property can also translate to exponential growth in runtime and in which situations a better scaling can be achieved. Also, discrete actions can exclude the true optimal solution that lies in between two discretization steps.

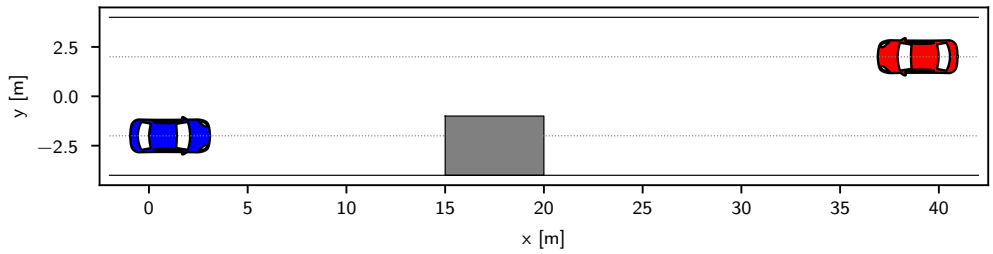


Figure 4.2: The example scenario used throughout this section: A two-lane scenario with a roadblock and two vehicles approaching from different directions with initial speed of 3 m/s, an obstacle (dark gray) on the right lane, and two reference lines (dashed).

On the other hand, discretization offers several benefits besides the lower implementation complexity compared to continuous formulations. First, the generative model for the agent dynamics can be chosen freely. We here chose a nonlinear vehicle model. Also, the infinite space of actions is manually bounded to a reasonable subset that guides the solution algorithm in a desired direction. Furthermore, graphs offer an efficient data structure, and parallelization can effectively be integrated into the algorithms.

As an illustrative example, we throughout this chapter use a two-vehicle roadblock scenario as depicted in Figure 4.2. The left (blue) ego vehicle 1 has to decide whether to pass the roadblock before or after an oncoming (red) vehicle 2, if the oncoming vehicle gives way, or if the oncoming vehicle creates space that both vehicles can pass the roadblock at the same time.

4.2 Behavior Option Generation

For each traffic participant, a tree-like graph structure of possible motions over time is calculated. Starting from the position of a vehicle at the current time as root state, we generate new future states by sampling new vehicle motion options. Per timestep, we sample a number of expansions resulting in new states. One central idea is to not recalculate the whole motion tree at every time instance but to reuse old information along the horizon. The graphs for each vehicle are independent.

The result of this step is a directed acyclic graph with vertices representing a possible state of a vehicle connected by edges representing how the translation between the two states is achieved. The graph is ”layered” in time, an edge always represents a connection between a state at time k to a state at time $k+1$. The graph is collision-free with the static environment.

Graph Nodes and Edges One node of the graph represents a dynamic state

$$s = (x,\ y,\ \theta,\ v,\ k,\ c_s) \tag{4.1}$$

with the pose x, y, orientation θ and velocity v of the traffic participant. Also, the time k and cost c_s of the state are stored. As root node of the tree, we chose the state closest to the actual vehicle state at the current time, as our computed states will not match the exact vehicle motions. Each time we add a new node to the motion tree, all possible motion patterns expanding the new node are added to an expansion list for future exploration.

These expansion list entries form the edges of the graph and encode how to transfer from one state to another and can be any physically admissible motion. We rely on a fixed input acceleration a and curvature κ for the time interval of one edge defined as

$$e = (a,\ \kappa,\ c_e) \tag{4.2}$$

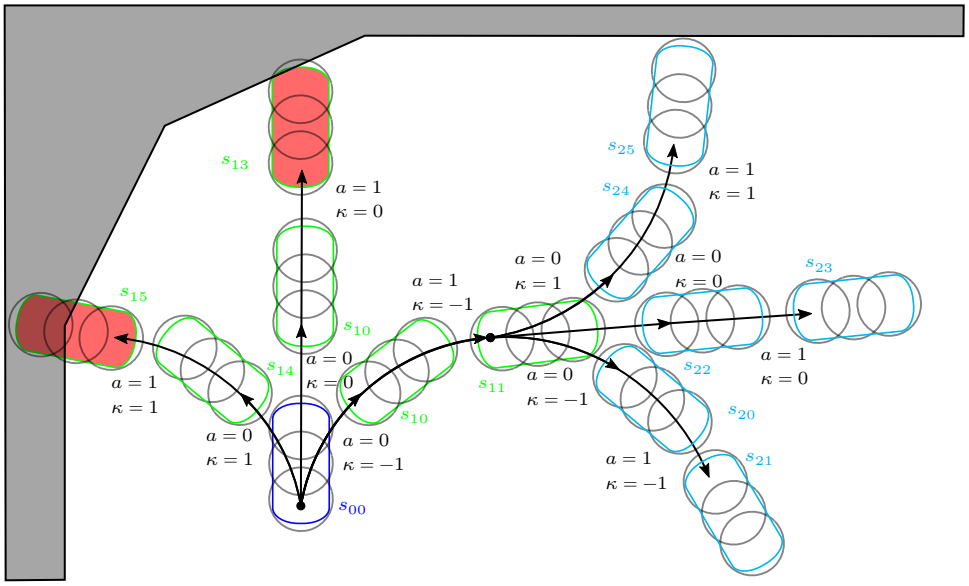


Figure 4.3: Schematic visualization of the state expansion starting in the behavior option generation step from state s_{00} (blue) with different edges corresponding to acceleration a and curvature κ values. Starting from the initial state s_{00} (blue), the states s_{1*} (green) indicate the first expansion step, s_{2*} (light blue) the second expansion step. The vehicle shape circle approximation is visualized. The states colliding with the static obstacle (gray) s_{13} and s_{15} (solid red) are dropped from the graph. (graphic from [KK19], ©2019 IEEE)

with costs c_e. The maximum available amount of curvature κ along on edge is decreased with growing velocity.

Graph Expansion Using the Euler discretization of the single track vehicle model,

$$\begin{pmatrix} x(k+1) \\ y(k+1) \\ \theta(k+1) \\ v(k+1) \end{pmatrix} = \begin{pmatrix} x(k) \\ y(k) \\ \theta(k) \\ v(k) \end{pmatrix} + \Delta t \begin{pmatrix} v(k)\cos\theta(k) \\ v(k)\sin\theta(k) \\ v(k)\kappa(k) \\ a(k) \end{pmatrix} \tag{4.3}$$

we derive motion primitives from these acceleration-curvature pairs with an appropriate stepsize Δt.

If two nodes result in the same dynamic state, the states are merged, and only one is kept in the graph. This process forms a directed acyclic graph. Figure 4.3 schematically shows how the tree is expanded and Algorithm 4.1 states the implementation of one planning step.

With adding a node to the (initially empty) graph by `addNode()` (line 14) also a list of possible next expansions is added by `appendNewEdges()` (line 15). This expansion list holds pairs of a state (parent nodes) and an edge definition leading to a possible new node. The function `calculateNextState()` computes a new possible node from one entry of the expansion list by stepping (4.3) forward by the predefined timestep (line 4). Afterward, the new state is checked for collisions with the static environment by `collide()` (line 5) and added to the graph if no collision is found. For the collision check, we approximate the vehicle shape with a set of circles [LKK15]. We used three circles per vehicle. If a similar node is already part of the graph (e.g., reached from another trace in the graph), checked by `stateAlreadyInGraph()` (line 6), we do not add the node again but only add a fitting edge in the graph. Note that the time k is part of the dynamic state which shrinks the search space. Consequently, we speak of a tree-like structure as there might be more than one path through the (directed acyclic) graph leading to a node following a different

Algorithm 4.1 One step of the behavior option generation algorithm

```
Input: T                                              ▷ Full expansion time
Input: s                                              ▷ Current vehicle state
Input: env                                            ▷ Static environment
Input: t                                              ▷ Motion tree graph
Input: e                                              ▷ Expansion list
 1: while time(front(e)) < T do
 2:     if not empty(t) then
 3:         nextExpansion ← popFront(e)
 4:         nextNode ← calculateNextState(nextExpansion)
 5:         if not collide(nextNode, env) then
 6:             if not stateAlreadyInGraph(nextNode, t) then
 7:                 addNode(nextNode, t)
 8:                 appendNewEdges(e)
 9:                 removeDeadEnds(e)
10:                 appendEmergencyStop(e)
11:             else
12:                 addConnections(t)
13:     else
14:         addNode(s, t)
15:         appendNewEdges(e)
16: return t, e
```

series of edges. If a node does not have any children (a dead end) or yields an immanent collision (e.g., node s_{25} in Figure 4.3), we remove this node from the graph and recursively all completely explored nodes that only connect to the dead end node by `removeDeadEnds()` (line 9). To make sure that for each node, an emergency trajectory is available, we by `appendEmergencyStop()` (line 10) create a stopping edge that brakes with maximum deceleration until zero velocity while keeping the current steering angle constant. The costs for this edge are set to a high value to make sure the optimizer only chooses a state sequence passing this edge if no other solutions are available. This strategy avoids infeasible optimization problems and increases the operational design domain, as sharp braking maneuvers are also handled. This algorithm is executed until the graph has been fully created until a given final maximum time-depth T. Figure 4.4 shows an example of one motion tree.

Cost Calculation The costs c_s of a state s calculate to

$$c_s = \xi_{\text{ref}}\text{dist}(s, \vec{s}_{ref}) + \xi_v\text{abs}(v_s - v_{ref}) + \xi_\theta\text{abs}(\theta_s - \theta_{\text{ref}}) \tag{4.4}$$

with scaling factors ξ and appropriate distance function dist(). For each vehicle, we either know the intention, and therefore the reference track and speed or the intention is estimated. We put positive cost scaling on the deviation from the reference track $\vec{s}_{ref}$ and the reference speed v_{ref}.We chose the Euclidean distance of s to the nearest point of the reference track from the static environment as the distance function. At this point, we also compute the reference orientation. As the distance from s to the root node we set the distance from the root node to the nearest point along the reference track.

The costs c_e of a edge e calculate to

$$c_e = \xi_a a + \xi_\kappa \kappa \tag{4.5}$$

with acceleration a, curvature κ and scaling factors ξ.

Graph Sample Time Choosing the sample time to expand the graph is crucial. It represents the time distance between two states in the graph. A small sample time is desirable to cover the true possibilities of actions a vehicle has but results in huge graphs that are computationally

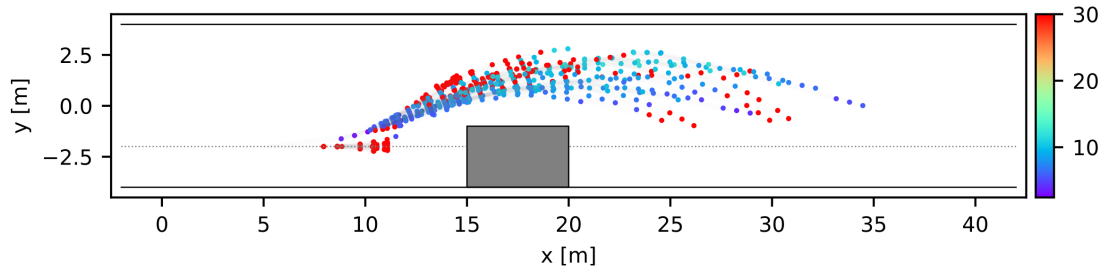


Figure 4.4: Example of one full expansion step: The states' x, y positions correspond to the vehicle positions. The color coding refers to the state costs. The gray obstacle is avoided as well as the solid black road boundaries; the reference track is depicted in dashed gray. Note that the edge curvatures are omitted as well as the vehicle orientation θ. (modified graphic from [KK19], ©2019 IEEE)

intractable. A larger sample time leaves out possible options but covers the high-level behavior better.

To avoid collisions in-between two states (either with the environment of with another vehicle), we subsample between two states and store these subsampled states leading to a new states alongside with this new, main state. The strategy is further elaborated in Section 4.3.2.

Handling of Dynamic Obstacles If a predicted trajectory for a fellow vehicle is available, the option generation can easily be adapted to reactively avoid the occupied area by this obstacle. In this case, we do not generate a set of options along the tree but just generate a single sequence of actions and states in the needed time discretization. This drastically reduces the problem size with the price of reactive instead of interactive planning.

4.3 Behavior Optimization

The purpose of this step is to find a conflict- and collision-free set of traces within the combined motion trees (cf. Section 4.2) of each vehicle minimizing the total costs in a joint cost function.

4.3.1 Graph Connectivity Constraints

Recall that we are calculating this motion tree individually for each vehicle in the scene. The trees are collision-free regarding the static environment but not regarding other vehicles. We formulate this search as mixed-integer linear optimization program, specifically as a network flow problem [CBD11] in terms of graph edges. n_e denotes the number of edges, n_s the number of nodes of the graph. Decision variables are the flows f on the edges of the graph

$$f_{st}^v \in \{0, 1\} \tag{4.6}$$

with the source node s and the target node t for each vehicle v. The problem consequently has n_e decision variables. Using the flow formulation, we ensure that an agent takes a unique trace through the graph. As nodes represent states in time, we make sure the agent is at exactly one state at each timestep. The flow conservation constraints are formulated to

$$\sum_{\text{Nodes j}} f_{ij}^v - \sum_{\text{Nodes j}} f_{ji}^v = \begin{cases} 1 & i \text{ source} \\ -1 & i \text{ sink} \\ 0 & \text{otherwise} \end{cases} \quad \forall \text{ Nodes } i, \text{ Vehicles } v \tag{4.7}$$

To make sure the optimizer does not get stuck at intermediate nodes, we connect all possible end nodes to a virtual sink node and force the flow formulation to start at the root node (source) and end at this sink node that is unique per vehicle. Possible end nodes are all nodes at the specified final optimization time or nodes with zero velocity. Each fully expanded node has at least one valid successor node that represents a vehicle state further in time or the agent has reached its destination. The root node of the graph represents the initial agent position at the current planning step.

4.3.2 Collision Avoidance Constraints

We implemented two ways to check the inter-vehicle collisions. In the first strategy, we check the collisions prior to the optimization for each pair of vehicles and store the result in a list, that we hand over to the optimization problem. The collision avoidance constraint is then formulated as

$$\sum_{\text{Nodes k}} f^v_{k,i} + \sum_{\text{Nodes k}} f^w_{k,j} \leq 1 \;\forall \text{ Nodes } i,j \text{ collide}, \quad \text{Vehicles } v \neq w \tag{4.8}$$

before calling the optimization program to constrain the usage of conflicting nodes. The collision calculation is again done by approximating both vehicle shapes as a series of circles and checking these for intersections. The naive implementation of looping through every node for all vehicles and checking the collisions is computationally intractable. Therefore, we first perform an approximate collision check based on the shape and position of the vehicles to mark definitely non-colliding states. To further boost the performance, we iterate through the graphs with increasing time and only compare nodes with the same timestamp. We stop to iterate deeper in the graph if a collision was found as future states cannot be reached anyway. Also, as the nodes are reused for future timesteps, the collision information is persisted.

With a large time discretization along the graph, collisions in-between two timesteps can occur. One obvious possibility is to reduce the time discretization, which is unfavorable. The solution time rises with a growing graph, and the graph contains too many equal or nearly-equal decision points. To still avoid collisions in between timesteps, we apply the following strategy: We add intermediate states along the graph edges connecting two states. These states are a by-product from the explicit Euler integration (cf. Section 4.2), where intermediate steps are needed anyway for sufficient numerical performance. As a timestep for these states, we do not set the correct time, in-between source and target node time, but the steptime or the target node. This implementation implies that when collision check is executed, not only the circles from each node have to be checked for intersection but also all circles from the introduced intermediate nodes. As the circle-to-circle collision check is efficiently implemented, this does not yield a huge overhead while simplifying the problem a lot. For the graph topology that is relevant for the flow-based optimization, the intermediate states are irrelevant and are omitted. This strategy comes at the price of introducing additional conservatism in the collision check. We ensure that time ranges are collision-free instead of checking each time instance within a range. All possible collisions are still covered, and this simplification is not a safety issue.

As a second method, we formulate the circle approximation collision checker as constraints internally within the optimization problem as

$$\begin{aligned} &(x_i + l_v\cos(\theta_i) - x_j - l_w\cos(\theta_j))^2 + (y_i + l_v\sin(\theta_i) - y_j - l_w\sin(\theta_j))^2 \\ \geq &(r_v + r_w)^2 - M(1 - \sum_{\text{Nodes k}} f^v_{i,k}) - M(1 - \sum_{\text{Nodes k}} f^w_{j,k}) \\ &\forall \text{ Nodes } i,j \;\forall \text{ Vehicles } v \neq w \end{aligned} \tag{4.9}$$

for each two nodes i, j with positions x, y and orientation θ for each pair of vehicles v, w. A vector l denotes the positions of the vehicle approximation circle centers, whereas 0 indicates a circle around the rear axle center point. r denotes the radius of these circles. If, for example, each

Algorithm 4.2 The two-stage network flow optimization and iterative collision check solution refinement algorithm

```
1: minimize (4.10) subject to (4.7)                          ▷ using a network flow solver
2: solutionFound ← false
3: while solutionFound = false do
4:     trajectories ← getLastSolution()
5:     if colliding(trajectories) then
6:         collidingNodes ← getCollidingPairOfNodes(trajectories)
7:         addCollisionConstraints(collidingNodes)
8:         Solve (4.12)                                      ▷ using a dual simplex solver
9:     else
10:        solutionFound ← true
11: return trajectories
```

vehicle is approximated by three circles (4.9) resolves to nine constraints. We use the well-known Big-M method with a sufficiently large number M.

Our experiments showed a better overall runtime performance using external collision constraints. We therefore rely on this strategy in the following.

4.3.3 Objective Function

The objective function minimizes the total costs along the combined graph traces which is equivalent to finding the minimal flow in the connected graphs. It calculates to

$$\sum_{\text{Vehicles } v} \lambda_v \sum_{\text{Nodes } i,j} f_{ij}^v (c_{ij} + c_i) \tag{4.10}$$

with the respective edge costs c_{ij} and the node costs c_i. We further introduce a vehicle *scaling factor* $\lambda \in [0, 1]$ to adjust the level of cooperation for each specific vehicle. All scaling factors shall sum up to one:

$$\sum_{\text{Vehicles } v} \lambda_v = 1 \tag{4.11}$$

Using these factors alongside with an appropriate adaption of the cost terms (cf. Section 4.4), we force the optimization problem to generate egoistic, symmetric, or altruistic solutions. For each agent A^i, we define a scaling factor λ^i with the following properties.

4.3.4 Optimization Problem and Solution Algorithm

The optimization problem is consequently formulated as

$$\begin{aligned} &\text{minimize } (4.10) \\ &\text{subject to } (4.7), (4.8). \end{aligned} \tag{4.12}$$

This formulation has the drawback that a relatively large number of collision constraints are calculated without a real need as the optimal flow through the graph does not interfere with these constraints at all. Also, as we will see in Section 4.7, the collision checks are a significant contribution to the overall runtime.

We therefore split the optimization problem (4.12) into two stages. First, we solve the flow problem without collision constraints using a specialized solver and afterward successively add the collision constraints as Algorithm 4.2 describes.

We first optimize the optimization problem without any collision constraints (line 1). We here chose a specialized solver that is efficient for network flow problems. Here the solver first extracts the network structure from the linear constraints definition solves the network with a

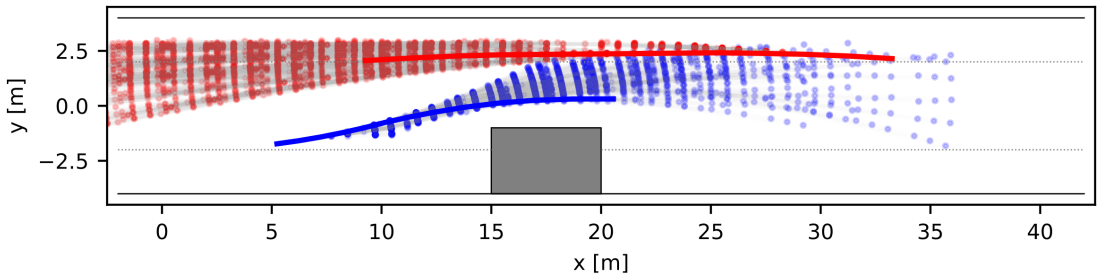


Figure 4.5: Example of two (blue, red) optimized trajectories (thick lines) in the motion trees avoiding the gray obstacle and staying within the road boundaries. Note that the edge curvatures are omitted as well as the vehicle orientation θ. (modified graphic from [KK19], ©2019 IEEE)

structure-adapted strategy to create a feasible starting base for the linear problem. The resulting linear problem is then solved using a simplex optimizer. We then obtain trajectories from the solution and check if these are collision-free (line 5). If yes, we have already found the optimum and return it. If not, we add collision constraints (lines 6, 7). Naively we could now only add a constraint for the colliding nodes. We found by empirical experiments that if two nodes collide, the optimizer in the next step tends to find trajectories using very similar nodes that also collide. This procedure yields an unnecessarily high number of optimization runs. Therefore we decide that if two nodes collide, we also check each node for collision with all nodes from the other agent at the respective time. This strategy is still an enormous speedup as not every node from every graph has to be checked against each other, but only one node from one graph against a subset of nodes from one other graph. The found collisions are then formulated as constraints and added to the problem. The problem structure is no longer a pure network flow, and we use the dual simplex algorithm to solve it (line 8) but with the base from the previous optimization run as a starting point. We iterate this procedure until the resulting trajectories are collision-free or the optimization fails if the problem is not feasible (omitted in Algorithm 4.2). The result of the optimization is an optimal edge sequence for each vehicle referring to a trajectory as depicted in Figure 4.5.

4.4 Behavior Reflection and Mixed-Traffic Integration

Besides coordinating connected vehicles, we also take non-cooperating vehicles into account by predicting and estimating their future motion and strategic plans. In the behavior reflection step, the cost scaling factors ξ are tuned to fit the individual vehicle intents and observations. Also, the intention of each non-communicating vehicle is re-estimated. Hence, the used reference track and reference speed can change.

CAV Setting In the case of a fully connected and cooperative scenario, cost terms and reference track and velocity are known for each vehicle. As each vehicle follows the optimized trajectory, there is only a need for adoption if the intention changes. Here, we assume these changing values to be explicitly exchanged among the vehicles to be easily adopted in the parameter settings. We do not consider adversarial communicating agents here.

Non-Communicating Settings For a non-communicating vehicle, we observe the deviation of the observed vehicle motion and the optimized vehicle motion. Based on this, the intention is estimated in terms of the reference track and the cost factors.

Algorithm 4.3 describes the procedure in pseudo-code. It is executed for each non-communicating agent independently. With the observed state s_{observed} of each agent, we check if any state in

Algorithm 4.3 The behavior reflection algorithm

1: **Input:** $s_{observed}$ ▷ Observed state of the agent
2: **Parameter:** $MaxDist_{\square}$ ▷ Maximum allowed distance between state in graph and observed state
3: $s_{graph} \leftarrow$ getNearestStateInGraph($s_{observed}$)
4: **if** distance($s_{observed}$, s_{graph}) > $MaxDist_{state}$ **then**
5: resetGraph($s_{observed}$)
6: **else**
7: $s_{planned} \leftarrow$ getPlannedState()
8: **if** speedDistance($s_{observed}$, $s_{planned}$) > $MaxDist_{speed}$ **or** speedDistance($s_{observed}$, desired velocity) > $MaxDist_{speed}$ **then**
9: UpdateDesiredVelocity($s_{observed}$)
10: UpdateVelocityAccelerationCosts()
11: **if** positionDistance($s_{observed}$, $s_{planned}$) > $MaxDist_{position}$ **or** positionDistance($s_{observed}$, reference line) > $MaxDist_{position}$ **then**
12: CheckAndUpdateReference($s_{observed}$)
13: UpdateDistanceCosts()
14: **if** orientationDistance($s_{observed}$, $s_{planned}$) > $MaxDist_{orientation}$ **or** orientationDistance($s_{observed}$, reference line orientation) > $MaxDist_{position}$ **then**
15: UpdateOrientationCosts()

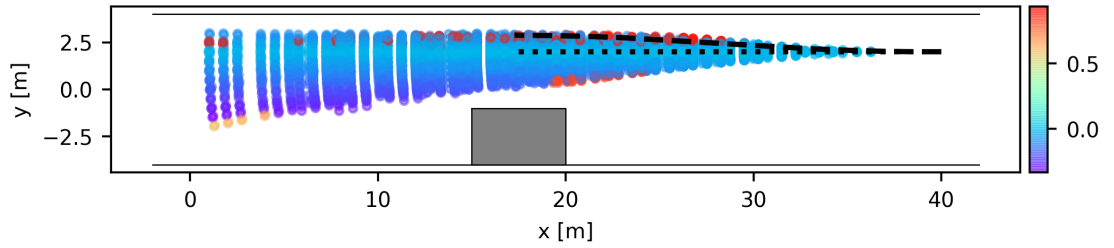


Figure 4.6: Example of how the behavior reflection modifies a graph, each node is one dot. The state costs of both graphs are subtracted and the difference is color-coded in the figure. The optimal trajectory before reflection (black, dashed) significantly differs from the optimal trajectory after reflection (black, dotted).

the graph matches this observed state (line 4). If not, the graph and the actual agent state have differed too much, and we cannot reuse the graph. It is, therefore, re-initialized at the observed position. If a matching state can be found, we perform three different checks. All use the optimized state for the agent $s_{planned}$. First, we check if the observed and planned speed deviates. Also, we check if the observed speed and the estimated reference speed deviate (line 8). If yes, we update the estimated desired velocity using the observed velocity of the agent (line 9). Also, we update the cost terms for velocity and acceleration costs (line 10). Second, we check the position difference of the observed state to the planned state and the reference line (line 12). If the thresholds are violated, we again update the respective cost term and check if another available reference line would fit the observed behavior better. For this calculation, we not only take the input state into account but also information collected from the prior motion of the agent (line 13). Third, a similar process is computed for the agent orientation. Often, observing the deviation of the orientation changes can be recognized prior to position changes.

Figure 4.6 displays the result of the reflection process. We subtract the state costs before and after the reflection step and color-code each state with this difference. In the example, it can clearly be observed that the costs for driving straight did not change, whereas the costs for steering maneuvers increased. Therefore, the optimal resulting trajectory also changed from performing a S-shaped curve to a straight line.

4.5 Real-Time Implementation

For a planning component to be executed in a real-world setting, it has to be ensured that a valid solution is always found before the end of the previous trajectory. Also, the planned motion has to be collision-free and kinematically feasible.

Solver Settings The optimization of the Mixed-Integer Progamming (MIP) is relatively fast, as we set up the problem in an efficient structure. Still, too many solver iterations have to be avoided to meet real-time requirements. As described in Section 4.3, the optimization problem is solved in a two-stage manner: In the first step, the problem is solved without collision constraints, and the solver is parameterized to solve the network flow problem efficiently. In the second stage with active collision constraints, a dual simplex solver is selected that is warmstarted with the solution from the previous step.

Before solving the linear program, the solver by default performs multiple presolve steps. This presolve tries to reduce the number of variables and constraints by substitution. Also, redundancies in the model are eliminated. Our model barely contains redundancies as we avoid adding redundant collision constraints, and we rarely include variables that can be substituted in the flow formulation. Still, there is room for model simplification, e.g., if a node has exactly one input and one output edge. Applying one presolving step proved to be beneficial for the overall runtime. More presolving steps, as the solver by default tries, are not expedient. We, therefore, limit the number of presolve steps to one.

Agent-to-Agent Collision Checks Regarding runtime, the collision check between two agents is most crucial. In the worst case, it scales exponentially if every node of every agent has to be checked for collision. In Section 4.3.2, we describe strategies to avoid this exponential scaling. Making use of the graph structures to avoid unnecessary collision checks and only performing collision checks just in time if needed effectively avoids exponential scaling, as we will discuss in Section 4.7. Checking two graph nodes for collision effectively reduces to the computation of the minimal distance of two pointsets. This calculation is performed with linearithmic complexity ($\mathcal{O}(n \log(n))$)[1].

Efficient Implementation The graph structure provides an efficient way to handle the receding horizon planning problem naturally. Persisting the graph and only pruning historic states reduces the computational time to build up the graph, and the information persisted in the nodes from scratch each timestep.

It is desirable to keep the graph as small as possible and avoid unnecessary states, such as duplicates. Therefore we make sure while inserting that the graph does not hold duplication if two traces through the graph are possible to reach one node. Also, we remove branches that do not lead to any reasonable states, such as only colliding with the environment or deadlocks. Whenever possible, we iterate through the graph structure instead of looping over all states to efficiently use the topology.

We use parallelization to distribute some of the heavy computations over multiple CPU cores. Every algorithm part that operates independently on the graphs of each agent is executed in a separate parallel task (such as the option generation and the reflection steps). Also, the collision checking between pairs of agents is parallelized. The solver also parallelizes the solution of the MILP over multiple threads.

Implementation Remarks and Software Design We implemented our approach in C++. The graph structures are represented using the boost graph library [SLL21] and the geometric

[1] `https://www.boost.org/doc/libs/1_72_0/libs/geometry/doc/html/geometry/reference/algorithms/distance/distance_2.html`

computations use the boost geometry library [Geh+21]. The optimization problem is formulated and solved using the commercial solver IBM ILOG CPLEX Optimization Studio [Int21b]. Option generation, collision checking, and reflection are parallelized using OpenMP [Ope21].

The graph of options for each agent is stored in a separate instance of the `StrategyGraph` class, which takes care of maintaining and expanding the graph, managing the list of future expansions, performing environment collision checks and executing the reflection. The class `OptimizationVehicleCoordination` combines multiple of these graphs and triggers the graph extension, generates and executes the optimization, including the agent-to-agent collision check, and triggers the reflection. It furthermore offers the high-level Application Programming Interfaces (APIs) for a simulator or the autonomous driving software stack.

Theoretic Limitations of the Approach The action space is discretized by design, which comes with two significant drawbacks. First, using discrete actions, not all nuances of motion can be achieved, and the true optimal action might not be part of the action space. So the resulting behavior can be suboptimal. Second, discretization suffers from the curse of dimensionality: Starting from one possible state at time 0, with n possible explorations, at time 1, n states are available, n^2 at time 2 and so on. Standard graph-based planning algorithms use heuristics to efficiently only explore the most relevant states [LaV06]. We, in our work, do not rely on heuristics but process only a subset of nodes in one step and leave the rest for the next receding horizon instance. Also, solely generating new states is not computationally expensive. It is the checking of states for collision and the selection of appropriate states. These two steps have to be implemented in a smart and performant manner, see Section 4.3.

When large actuation or perception errors occur, the problem exists that the current state of one agent does not match any state in the graph sampled in the past. Then the graph has to be deleted and computed from the current new agent position. While theoretically, this is not an issue, it requires an undesired additional amount of computation time.

4.6 ACCORD-P: Extending ACCORD to Valet Parking

ACCORD has been developed to coordinate vehicles on reference lines and not to navigate a vehicle to a specific point, such as a parking spot. In Kessler and Knoll [KK17], we discuss the special case of cooperative maneuvering multiple agents in a parking lot in a static environment with cooperating and non-cooperating agents present. It is based on the observation that for maneuvering on parking lots, only a limited set of standard strategies/trajectories is needed to cope with every specific spot. The proposed strategy – Autonomous Car Coordination for Valet Parking (ACCORD-P) – is a special case of ACCORD described above.

In contrast to ACCORD, we sample predefined trajectories including direction changes instead of actions and do not form a motion tree. This comes with the benefit that the final pose can be reached more accurately. Also, ACCORD is capable of including reverse driving segments, but the graph then grows significantly faster, resulting in slower runtimes. We use MIP to optimize the assignment of trajectories and agents.

In a highly structured environment, the set of possible actions and the set of goal poses are limited, and the options each vehicle can take are limited. In the parking case, a vehicle is either looking for a suitable parking spot, maneuvering into/out of a parking spot, or leaving the parking site. The trajectory a vehicle will take is among the set of pre-calculated ones or at least close. Most coordination approaches assume fixed goal positions for each vehicle. Here, the exact goal is a degree of freedom, and the goal choice is optimized as from an estimated vehicle intention, a set of possible goal positions is computed.

Table 4.1: Properties calculated for each trajectory of ACCORD-P

Name	Property	Scaling	Description
distance to end	p_{de}	φ_{de}	Distance from trajectory end to the chosen goal
driven distance	p_{dd}	φ_{dd}	Traveled distance on this trajectory if driven completely
ends at goal	p_{pl}	φ_{pl}	1 if the trajectory's goal is reached
acceleration amount	p_{aa}	φ_{aa}	Total amount of acceleration required
steering amount	p_{sa}	φ_{sa}	Total amount of steering required
direction changes	p_{dc}	φ_{dc}	Penalize paths changes the direction often
speed difference	p_{sd}	φ_{sd}	Deviation of desired and chosen vehicle speed

Trajectory Sampling

No effort was put into the generation of the vehicle trajectories. Following the implementation of Reed-Shepps curves [LaV06], a path drivable by a non-holonomic vehicle is generated and enriched with a linear velocity profile. Using multiple maximum speeds along the path, we generate a set of trajectories along the paths. The paths underlying the trajectories lead from the current vehicle position to a goal pose, such as a parking spot. The resulting trajectories track this path for a defined time interval with the given velocity profile.

For each pre-computed trajectory, we compute a set of characteristics as quantified properties. Each agent puts a different weight on the properties according to its high-level intention. With ideal knowledge of a vehicle's intention, the trajectory it is willing to choose could be matched exactly. However, the estimation of the intentions of non-communicating agents is a non-trivial task and will not be accurate. This results in the prediction of a wrong trajectory and a sub-optimal result. All properties and weights are zero or positive. Any value that can be computed from a trajectory can serve as property. We use the following set:

- Spatial distance to the targeted goal,
- traveled distance on this trajectory if it is driven from start to goal,
- flag if the trajectory reaches its goal in the current timestep,
- amount of acceleration needed to follow the trajectory as a rough measure for energy consumption,
- amount of steering input needed to follow the trajectory as a rough measure for comfort,
- number of direction changes of the trajectory,
- deviation from the desired top speed.

According to the intention, the weight set is biased towards (e.g.) minimum travel time, maximum smoothness, or low consumption. The properties are summarized in Table 4.1. For non-communicating agents, based on the observed actions of a vehicle, weights are extracted that fit the observed behavior most, as we described in Section 4.4. Interacting with other autonomous vehicles, weights are exchanged via Vehicle-to-Vehicle (V2V) communication. The weights will directly influence the cost function of the optimization. Weights of different magnitude will force the optimization to stronger incorporate the intentions of the corresponding vehicle and property. We perform a pair-wise collision check for each of the sampled trajectories and store the results for later use in the optimization.

Trajectory Assignment as Optimization Problem

We use MILP to assign a trajectory to each vehicle and to optimize the assignment due to given criteria. The problem is formulated symmetrically in the sense that no distinction is made between

the ego and other vehicles. To find an optimal assignment each trajectory is enhanced with properties, denoted by p, and each vehicle can set weights, denoted by φ, on these properties, cf. Table 4.1. Defining the decision variables x with $x(v,t) = 1$ if vehicle v takes trajectory t and 0 otherwise the objective function aiming to minimize the total costs can be written as

$$\sum_{t\in\mathcal{T}}\sum_{v\in\mathcal{V}} x(v,t)\Big(\sum_{i=1}^{N} p_i(t)\varphi_i(v)\Big) \tag{4.13}$$

with the set of vehicles $\mathcal{V}$ with a unique start pose and possibly several goal poses and set of trajectories $\mathcal{T}$ from vehicle start to goal poses. By $\mathcal{G}$ we denote the set of possible goal poses. By the constraints

$$x(v,t) \in [0,1] \quad \forall \quad t \in \mathcal{T}, v \in \mathcal{V} \tag{4.14a}$$

$$\sum_{t\in\mathcal{T}} x(v,t) = 1 \quad \forall \quad v \in \mathcal{V} \tag{4.14b}$$

we ensure each vehicle out of $\mathcal{V}$ is assigned exactly one trajectory out of $\mathcal{T}$. With the pre-calculated collision matrix $C(t_1, t_2)$ that indicates if trajectory t_1 and t_2 collide (2 if no collision, 1 if collision) we exclude colliding trajectory combinations by

$$\sum_{v_1\in\mathcal{V}} x(v_1,t_1) + \sum_{v_2\in\mathcal{V}} x(v_2,t_2) \leq C(t_1,t_2) \;\forall \quad t_1 \in \mathcal{T}, t_2 \in \mathcal{T}. \tag{4.15}$$

We further define an assignment matrix A ensuring a vehicle v can take trajectory t as start and goal poses match (1 if possible, 0 if not) and a corresponding constraint

$$x(v,t) \leq A(v,t) \quad \forall \quad v \in \mathcal{V}, t \in \mathcal{T}. \tag{4.16}$$

The matrix A is easily set up while computing the trajectories.

The resulting optimization problem computes an optimal vehicle/trajectory assignment for one step avoiding collisions and balancing individual interests is formulated as

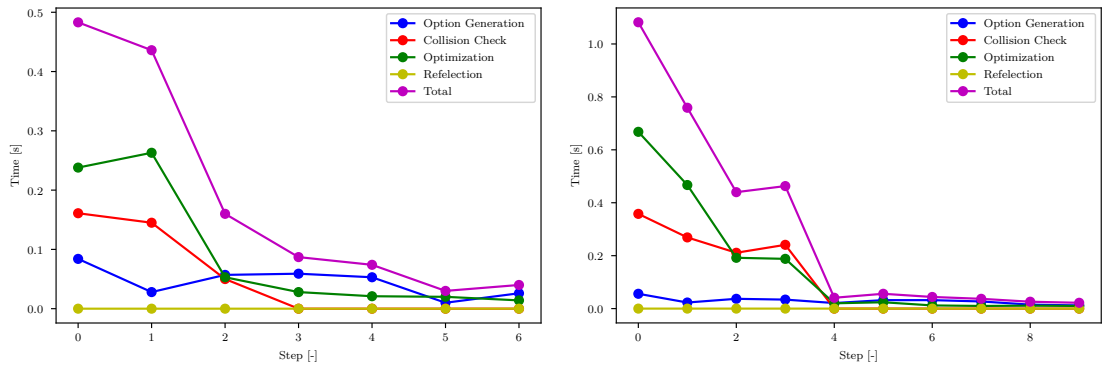
$$\begin{aligned} &\text{minimize } (4.13) \\ &\text{subject to } (4.14a),\ (4.14b),\ (4.15),\ (4.16). \end{aligned}$$

The constraint functions 4.14a and 4.14b ensure that each vehicle out of $\mathcal{V}$ is assigned exactly one trajectory out of $\mathcal{T}$. The collision avoidance constraint 4.15 relies on a precomputed collision matrix C to exclude colliding trajectory combinations. The objective function 4.13 aims to find the cheapest assignment of vehicles and trajectories. Valid assignments are ensured by (4.16). For each vehicle/trajectory pair the cost regarding the trajectory properties and corresponding vehicle scaling factors is computed.

4.7 Complexity Analysis

In this section, we will analyze how the problem complexity and overall runtime of ACCORD scales when increasing critical parameters such as the number of sampling steps. The experiments are all performed on the same machine with an Intel Core i7-9850H 2.60GHz CPU with 12 cores and 32GB of memory.

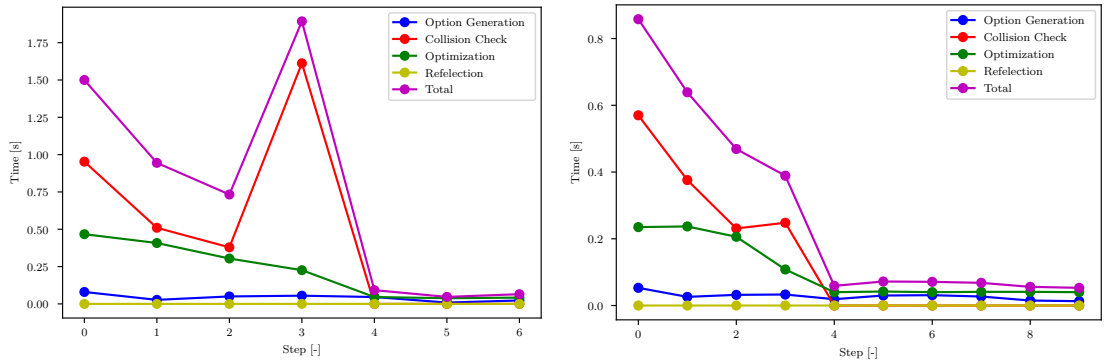
We will perform quantitative experiments in Section 4.7.2 in four environment settings. First an empty environment where the environment does not constrain the motion. The reference paths are chosen as parallel straight lines. Second, a two-lane road with oncoming directions with reference lines as the centerlines of each lane, third a wide one-lane road, and fourth narrow one-lane road. In Section 4.7.1, we motivate this section with a qualitative analysis of the road block scenario, we used as an illustrative example throughout this chapter.



(a) Cooperative setting

(b) Non-cooperative setting

Figure 4.7: Distribution of the overall solution time over the algorithm stages if collision checks are only performed if needed using the iterative algorithm Algorithm 4.2. The less dense the interaction is, the faster is the solution. The non-cooperative case requires significantly more iterations leading to a higher runtime.



(a) Cooperative setting

(b) Non-cooperative setting

Figure 4.8: Distribution of the overall solution time over the algorithm stages when checking all nodes for collision before starting the optimization.

4.7.1 Runtime Analysis of the Example Scenario

With two different scenario settings, one where the right (red) vehicle makes room for the left (blue) vehicle and both can pass the roadblock at the same time, see Figure 4.18, we observe a typical distribution of the runtime over the different stages of the cooperation algorithm. The more tight the interaction is, the higher the runtime. Also in an uncooperative setting, where the right (red) vehicle stays on the reference line and the (blue) ego vehicle has to stop and wait, see Figure 4.20, a similar timing pattern can be observed.

Contribution of the Different Algorithm Stages

Figure 4.7 shows that collision checking is one main runtime factor. The time for the optimization also scales with the necessary number of collision checks, as more optimization problems have to be solved. The time for creating the behavior options in the graph is compared to the other stages constant and non-time-consuming as expected. The time for behavior reflection is not significant.

In comparison to Figure 4.7, Figure 4.8 shows the distribution of the overall solution time if the collision checks are all carried out prior to optimization and solving only one optimization program

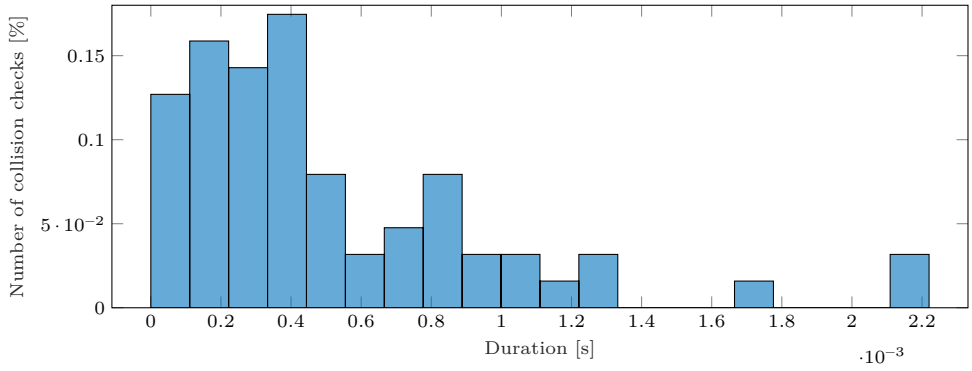


Figure 4.9: Histogram of the evaluation time one collision check takes.

instead of several. We observe several drawbacks when not iteratively solving the problem and performing the collision checks. First, more collision checks take more time, and therefore the overall solution time is generally higher. Second, when the initial solution is already conflict-free, a possibly large number of collision checks are still performed (such as in step 5 of the cooperative scenario). Third, the MIP is bigger and more complicated and therefore takes more time to solve. Nevertheless, as we can observe in the first steps non-cooperative scenario, a priori computing all collision checks can also be faster. The reason is that we have a lower overhead for creating and solving a multitude of small optimization problems.

Single Agent-to-Agent Collision Check Duration

One collision check of two-vehicle states in these scenarios takes at mean 5.1944×10^{-4}s with a variance of 2.2032×10^{-7}s averaged from 22306 runs. This number cannot be extrapolated linearly due to the use of parallelization in the collision checker and the different constellations of circle sets. The parallelization approach works most efficiently if the colliding nodes are decoupled into separate batches, which is a property of the specific situation. Figure 4.9 shows the histogram of the evaluation time of these collision checks. The duration of 90% of the checks takes below 1 ms.

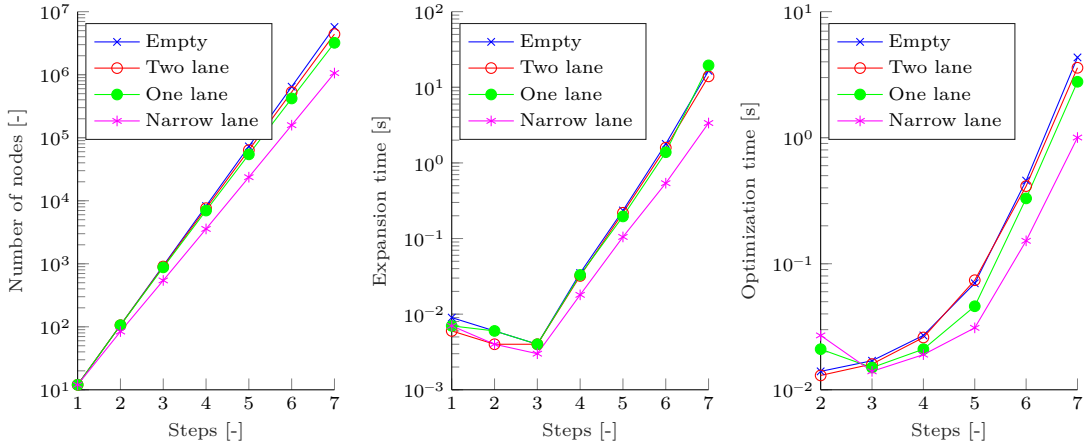
4.7.2 Quantitative Runtime Analysis

We analyze the effects on the overall runtime of the horizon length, the newly expanded nodes per node, and the collision check strategy.

Effect of Horizon Length

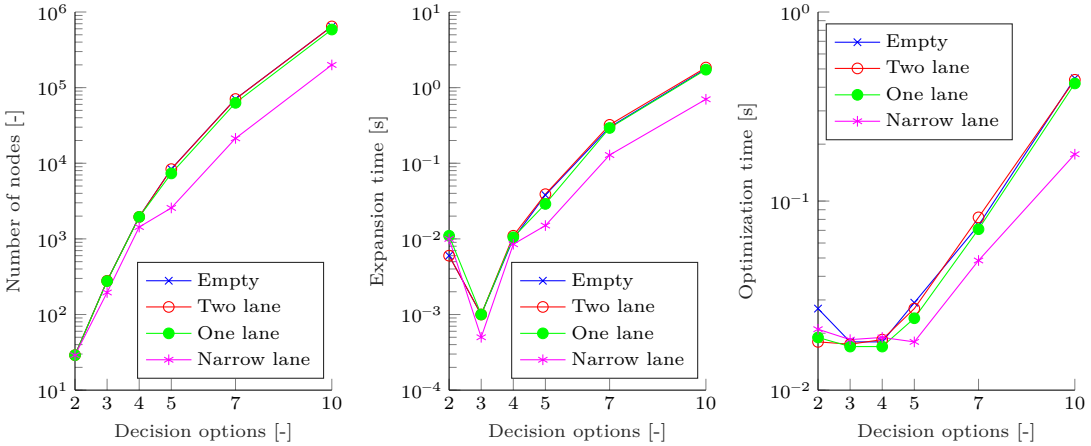
In Figure 4.10, we analyze how a longer horizon affects the time needed for behavior option generation. A longer horizon is equivalent to a growing number of sampling steps, resulting in a deeper graph. We expand ten new edges leading to new nodes at each graph node and analyze one receding horizon instance sampled into the future with a varying number of steps. Recall that new nodes are dropped if colliding with the environment. Therefore, the number of graph nodes varies in different environments and initial conditions.

Figure 4.10a displays the number of graph nodes and the time needed to sample these nodes with respect to the number of steps that are sampled into the future. The more open the environment is, the more nodes can be sampled, resulting in a bigger behavior graph. On a very narrow lane, nearly no steering options are valid. Therefore, the graph is comparably small. The maximum number of ten new edges per node can be observed easily in the graph. Also, the time for expanding a further timestep grows exponentially, cf. Figure 4.10b. The decreasing expansion time from



(a) Number of graph nodes (b) Expansion time (c) Optimization time

Figure 4.10: Analysis of the effect of a growing horizon time/number of sampling steps.



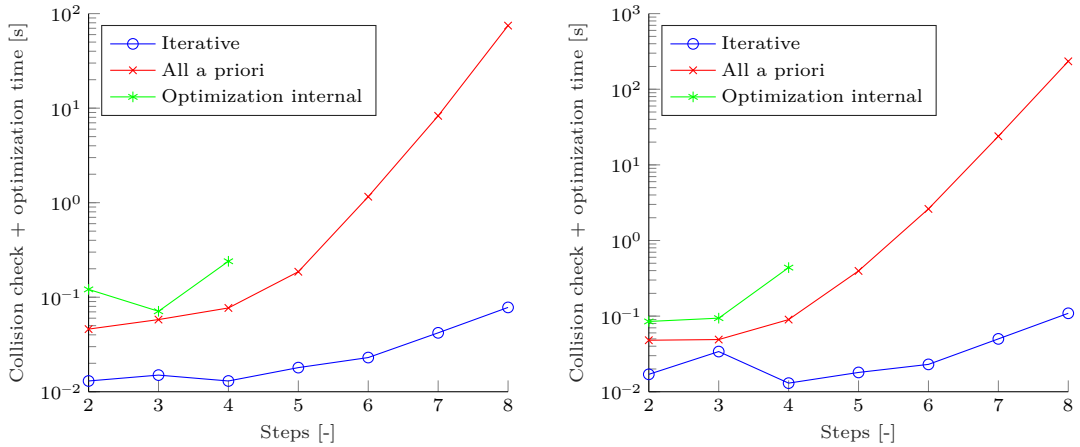
(a) Number of graph nodes (b) Expansion time (c) Optimization time

Figure 4.11: Analysis of the effect of a growing number of decision options per step.

one to three nodes corresponds to the overhead of code optimization and parallelization and is of minor practical relevance. Furthermore, the time for finding the optimal trace through the graph increases with the number of nodes, as we see in Figure 4.10c. From these experiments, we conclude that for real-time applicability expanding ten edges, sampling six steps into the future is a reasonable upper limit.

Effect of Decision Options per Step

Figure 4.11 depicts the effect of a growing number of decision options per horizon step. The experiment ranges from two options (keep the current acceleration/steering profile and emergency maneuver) to ten options. We observe an exponential growth in the number of nodes if the environment is not tightly constrained, which is expected. Also, for the expansion time, we observe exponential growth. The optimization time is relatively constant until five decision options. With more options, we observe exponential growth but with a moderate time increase. We from this experiment conclude that to not end up in exponential growth in the computation time, either the



(a) Two agents

(b) Three agents

Figure 4.12: Effect of different collision checking strategies in a scenario with barely conflicts.

environmental conditions have to be limiting or the possible next motion options at each horizon step have to be selected reasonably.

Effect of the Different Agent-to-Agent Collision Check Strategies

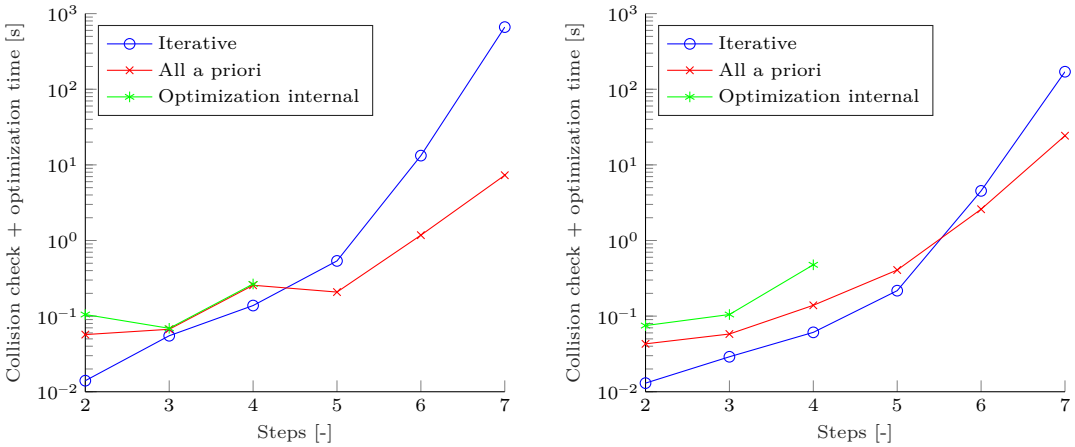
Let us now analyze the effect of different inter-agent collision checking strategies with respect to the number of horizon steps. We simulate three variants:

1. The two-stage network flow optimization and iterative collision check solution refinement algorithm as introduced in Algorithm 4.2,
2. pre-calculating all collision constraints (cf. (4.8)) recursively along the graphs, and
3. modeling collision avoidance as constraints internally in the optimization problem as formulated in (4.9).

All experiments are performed in an empty environment to not bias the result with an (un-)favorable start position of an agent. We expand five new edges at each node. We simulate two types of scenarios; first, a setting without dense interaction of all agents as the goals are barely conflicting. We place all agents on non-conflicting reference lines. Second, dense interaction is necessary as all agents are placed on the reference line of another agent, and the positions have to be swapped. We show the results of scenarios with two and three agents. The trend continues for more agents, and the exact results depend more on the initial configuration. Generally, the evaluation time mainly scales with the number of steps than with the number of agents. Starting from five steps, the optimization-internal collision constraints get computationally intractable.

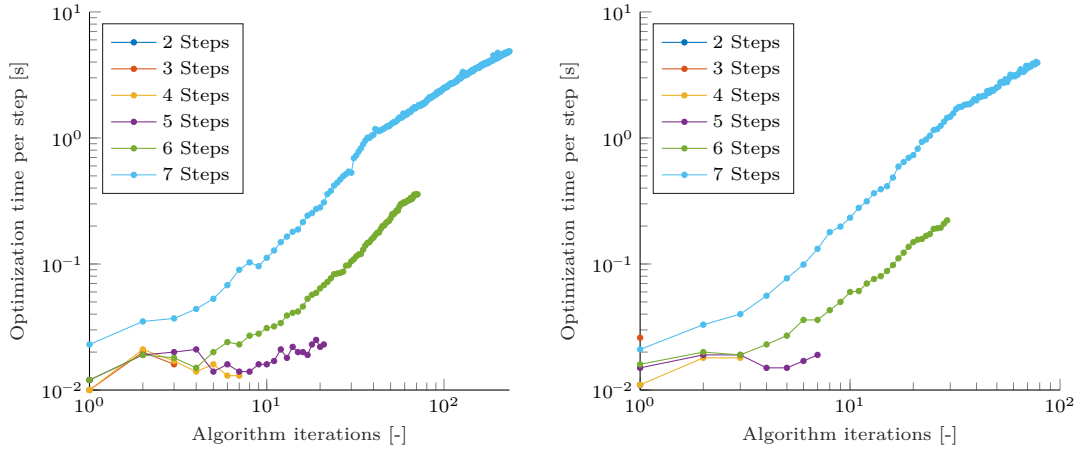
In the first case without conflicts, the two-stage network flow optimization and iterative collision check solution refinement algorithm clearly outperforms all others. This is expected, as few collision check computations have to be performed on the optimal traces through the graph. Until six horizon steps also the pre-calculation of all constraints is computationally tractable. With more horizon steps, the exponential growth of makes real-time application critical. These effects can be observed in Figure 4.12.

In Figure 4.13 we depict the scenario with conflicts. For a few horizon steps, the two-stage network flow optimization and iterative collision check solution refinement algorithm is faster or comparably fast than checking all collision a priori. With more horizon steps, this effect vanishes as the overhead of modification and re-optimization of the optimization problem is slower than computing all collision checks a priori using the graph structure and parallel computing.



(a) Two agents (b) Three agents

Figure 4.13: Effect of different collision checking strategies in a scenario with dense interactions.



(a) Two agents (b) Three agents

Figure 4.14: Development of the optimization time per step of the two-stage network flow optimization and iterative collision check solution refinement algorithm. The different curves represent trajectories with a different number of steps. The plots show how long one algorithm evaluation took with a growing number of iterations.

From these experiments, we conclude that with a high number of horizon steps, a pre-calculation of all possible collisions can be desirable in very interactive scenarios, whereas otherwise, the two-stage network flow optimization and iterative collision check solution refinement algorithm is preferable.

Effect of Increasing Number of Iterations

To understand why the two-stage network flow optimization and iterative collision check solution refinement algorithm can take longer in situations where agents densely interact, we analyze how many iterations the algorithm needs and how long one interaction of this algorithm takes in Figure 4.14. The more receding horizon steps are also needed, the longer the solution of one step takes and the solution time per step only grows. For the first few algorithm iterations, the

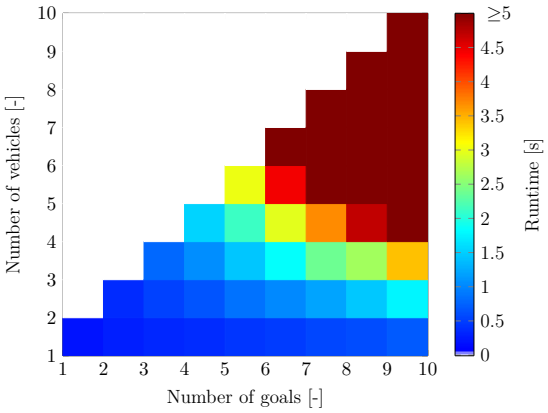


Figure 4.15: Total runtime in seconds with varying number of goals and vehicles in the same environment. White patches indicate infeasible combinations. (graphic from [KK17], ©2017 IEEE)

runtime does not significantly grow. For a few receding horizon steps, only a few algorithm steps are needed. For a bigger number of steps, we observe exponential scaling.

4.7.3 Runtime Analysis of ACCORD-P

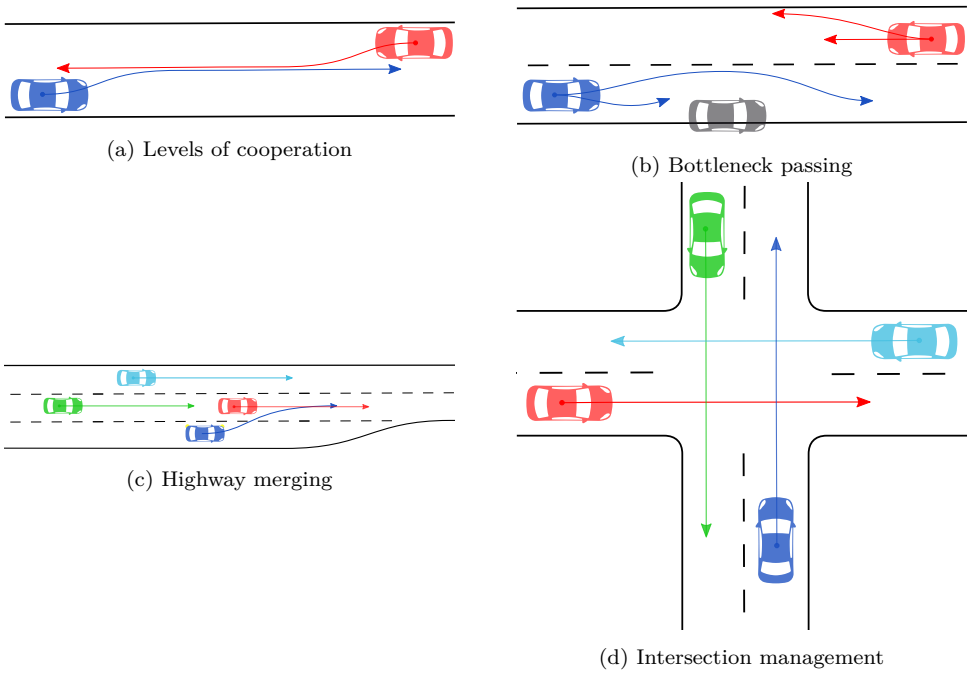
In Section 4.6, we presented Autonomous Car Coordination for Valet Parking (ACCORD-P), a specialized implementation for automated valet parking scenarios. Recall that the algorithm here coordinates the selection of complete, precomputed trajectories instead of finding an optimal trace through a tree of agent states. This algorithm scales differently in complexity and runtime, which we present in the section at hand. The experiments are executed on a standard PC with Intel Xeon 3.7GHz CPU with 4 cores and 32GB of memory.

The main runtime contributions are the trajectory generation and the optimization step. The number of generated trajectories is bounded by $|\mathcal{T}| = |\mathcal{V}| \times |\mathcal{G}| \times |\mathcal{S}|$, where $|\mathcal{S}|$ denotes the set of desired trajectory top speeds. The number of decision variables is bounded by $|\mathcal{T}| \times |\mathcal{V}|$. The trajectory generation time and the optimization runtime scale linearly with the number of trajectories. Collision checking of all trajectories is the most computationally expensive part of the trajectory generation. It scales quadratically with the number of trajectories. A quantitative analysis shows that the optimization step scales quadratically with the number of vehicles and goals. The trajectory generation step scales linearly in the number of vehicles. For few goals (≤ 10), this step scales quadratically but exponentially for a higher number of goals.

No effort has been devoted towards improving the absolute runtime; the implementation is, e.g., single-threaded. The illustrative two-vehicle example (see Section 4.8.5) with 2 vehicles and 6 goals has a maximum step time of 0.51 s. In the four vehicle example, the maximum cycle time is 2.02 s. In a benchmark scenario with a varying number of vehicles and goals, the total runtime of one iteration is displayed in Figure 4.15. The solver parameters are chosen to iterate to the optimum, which can be time-consuming, but a feasible, non-optimal solution is often found fast.

We conclude with the observed scaling effects that the proposed algorithm can be applied in real-time in scenarios with a moderate number of vehicles and goals. For a higher number of vehicles, heuristics have to be applied, which choose a subset of vehicles that shall be considered for interactions. This selection can lead to suboptimal solutions viewed from a global perspective.

If applied as an overall coordination approach in a fully autonomous parking garage, the algorithm can be executed on a dedicated server instead of a vehicle, which mitigates runtime effects. In such a use-case also longer trajectories can be optimized, and the cycle time can be chosen lower. Furthermore, a global infrastructure can easily choose reasonable splits of the overall optimization problem, e.g., location-based.



(a) Levels of cooperation

(b) Bottleneck passing

(c) Highway merging

(d) Intersection management

(e) Valet parking using ACCORD-P

Figure 4.16: Schematic sketches of the five evaluation scenarios using ACCORD

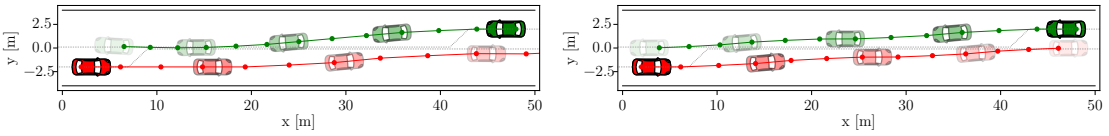
4.8 Demonstration of Benefits and Effectiveness in Simulated Scenarios

In this section, we will assess the performance of ACCORD in various simulated scenarios to demonstrate the universal applicability of the approach. Figure 4.16 sketches five different experiments, each demonstrating the capabilities of ACCORD or ACCORD-P. Table 4.2 gives an overview. By coordinated we denote a connected setting with CAVs that follow a global optimal plan. By reactive or proactive we denote the integration into a mixed-traffic scenario with the ego vehicle either showing passive behavior or actively enforcing own goals.

All parameters are set individually per vehicle. In the shown evaluations, we choose the same parameters for simplicity. Each vehicle's reference tracks and velocities are predefined but may change in a reflection step. With different cost terms, different behaviors are realized, as shown in the subsequent sections. The cost scaling parameters ξ were chosen to demonstrate the capabilities of the algorithm. We chose a discretized timestep δt of 2 s with a subsampling of 0.5 s and a horizon length of 6 s. Per node, we expand three acceleration and three steering options.

Table 4.2: Overview of the five different evaluation scenarios used in this section to showcase the capabilities of ACCORD and ACCORD-P. ◊ denotes not applicable, ♦ denotes applicable.

Section	4.8.1	4.8.2	4.8.3	4.8.4	4.8.5
Scenario	Levels of cooperation	Bottleneck passing	Highway merging	Intersection management	Valet parking
Schematic figure	4.16a	4.16b	4.16c	4.16d	4.16e
Number of agents	2	2	4	3... 4	2... 4
Coordinated	♦	♦	♦	♦	◊
Proactive	◊	♦	◊	◊	◊
Reactive	◊	♦	♦	◊	♦
Receding horizon	◊	♦	♦	♦	♦
Speed	low	low	high	medium	low
Planner	ACCORD	ACCORD	ACCORD	ACCORD	ACCORD-P



(a) Cooperative solution for $\lambda_2 = 0.2$

(b) Cooperative solution for $\lambda_2 = 0.5$

Figure 4.17: A scenario with conflicting references. The optimized solution is leveraged with the cooperation factor λ.

4.8.1 Levels of Cooperation in a Negotiation Scenario

We evaluate the effect of the cooperation factor λ in a negotiation scenario with two agents A^1 and A^2, the same scenario as we discuss in Section 5.21a. The scenario is fully symmetric with two vehicles placed on a two-lane road in oncoming direction with road boundaries. Both reference lines do not track the lane centers but the road center. This yields conflicting goals (Figure 5.21a). The solution can be balanced to favor one or the other agent. The vehicle trajectories change as λ changes (Figure 4.17a and Figure 4.17b). As costs, we only set the squared euclidean distance to the reference. Therefore the overall costs per agent are equivalent to the overall distance of the trajectory from the reference times λ.

In Table 4.3, we qualitatively show the effect of varying λ. We analyze a single run of the algorithm. The column Idx $A^{\square}$ indicates at which time index the respective agent has reached the reference trajectory. We also state the contribution of each agent to the global cost function, denoted by Cost $A^{\square}$. Both metrics show the same trend; by varying $\lambda_{\square}$, the respective agent is favored. We observe that for $\lambda_{\square} \approx 0.5$ all metrics are balanced, but also with a strong favor of one agent still, valid solutions are computed. Due to the chosen discretization, fewer scenario variants occur compared to the continuous formulation. Small changes in $\lambda_{\square}$, like from 0.2 to 0.3, do not yield a difference in the computed optimal maneuver as the same discrete motion primitives are chosen.

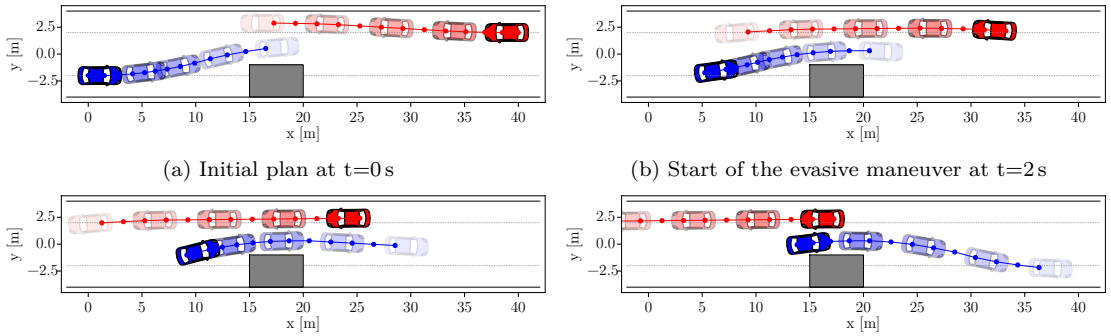
From this experiment, we conclude that the cooperation factor effectively favors one or the other agent. However, with a continuous scaling of λ, the discretization in the option graph still limits the number of possible solutions to a discrete set.

4.8.2 Cooperative Solutions of Two-Vehicle Scenario with Conflicts

We demonstrate the capabilities of ACCORD in a scenario with two vehicles and a roadblock that we already used as an illustrative example, see Figure 4.2. In a variety of different settings, a reasonable behavior is always achieved. In the following, we discuss three different examples. We consider the blue vehicle 1 on the left as the ego vehicle. Recall that ACCORD always assumes

Table 4.3: Quantitative evaluation of the negotiation scenario. We compare the time the reference is reached and the contributions to the cost function of each agent.

λ_2	Idx A^1	Idx A^2	Cost A^1	Cost A^2
0	-	8	0.0	222.05
0.1	12	8	38.46	199.85
0.2	12	8	76.93	177.64
0.3	12	8	115.39	155.44
0.4	12	8	153.86	133.23
0.5	11	11	150.63	150.62
0.6	8	12	133.25	153.85
0.7	8	12	155.46	115.39
0.8	8	12	177.67	76.92
0.9	8	12	199.88	38.46
1	8	-	222.09	0.0



(a) Initial plan at t=0 s

(b) Start of the evasive maneuver at t=2 s

(c) Evasive maneuver at t=4 s

(d) Both agents meet while passing the obstacle at t=6 s

Figure 4.18: Evolution of a two vehicle scenario with cooperative evasive behavior on a Cartesian plot. The lane of vehicle 1 in blue is blocked and vehicle 2 in red makes enough space to safely overtake. Dark to light colors depict increasing time. The trajectories start at the current vehicle positions. The reference lines are dashed gray.

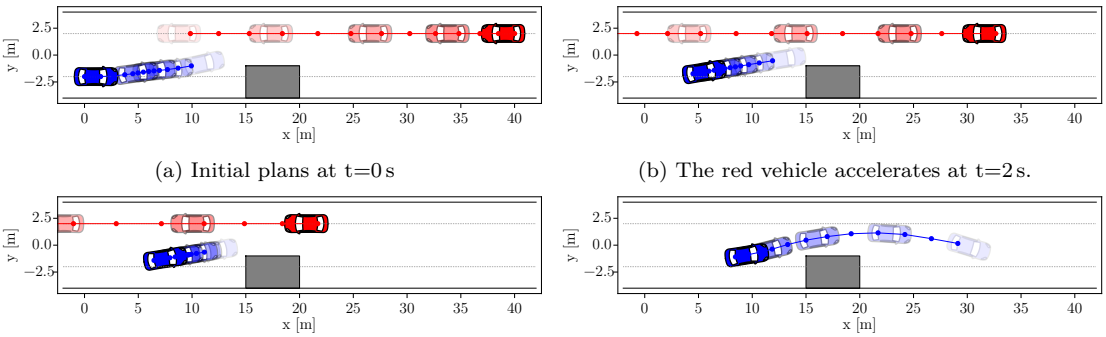
collaboration of all agents but in the reflection step can also counteract egoistic behavior and prediction errors.

Coordinated Evasive Maneuver

Figure 4.18 shows the evaluation of the traffic scene over time at selected time instances, including the currently planned trajectories and occupied spaces. In this behavior coordination scenario, the optimal solution for vehicle 2 is to perform an evasive maneuver so both vehicles can pass the obstacle at the same time. With constant acceleration control inputs per timestep, the motion is continuous but jerky due to the linear velocity profiles.Vehicle 1 also deviates from the reference speed to pass each other at an optimal space-time location.

Coordinated Speed-change Maneuver

By adapting the parameters, different cooperative solutions can be enforced. With a lower focus on the reference speed tracking and a higher focus on keeping the reference track, vehicle 2 accelerates, and vehicle 1 does not have to stop in front of the obstacle but can accelerate while vehicle 1 is passing the obstacle. The evolution of this scenario is depicted in Figure 4.19. In contrast to the uncooperative case, neither vehicle has to come to a complete stop. As both vehicles have agreed on the cooperative motion plan, vehicle 1 can start the overtaking maneuver earlier.



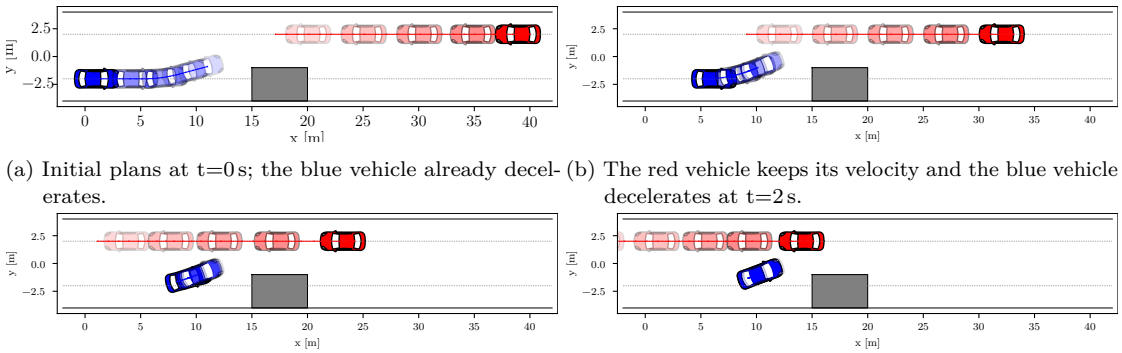
(a) Initial plans at t=0 s

(b) The red vehicle accelerates at t=2 s.

(c) The blue vehicle decelerates but does not stop completely at t=4 s.

(d) The blue vehicle passes the obstacle after the red vehicle at t=6 s.

Figure 4.19: Evolution of a two vehicle scenario with cooperative velocity change behavior on a Cartesian plot. Dark to light colors depict increasing time. Vehicle 2 accelerates to let vehicle 1 pass the obstacle earlier without stopping. At t=6 s, vehicle 2 does not interfere with the ego vehicle 1 any more and is therefore omitted.



(a) Initial plans at t=0 s; the blue vehicle already decelerates.

(b) The red vehicle keeps its velocity and the blue vehicle decelerates at t=2 s.

(c) The blue vehicle creeps forward to eventually come to a complete stop at t=4 s.

(d) The blue vehicle stops at the obstacle to first let the red vehicle pass at t=6 s.

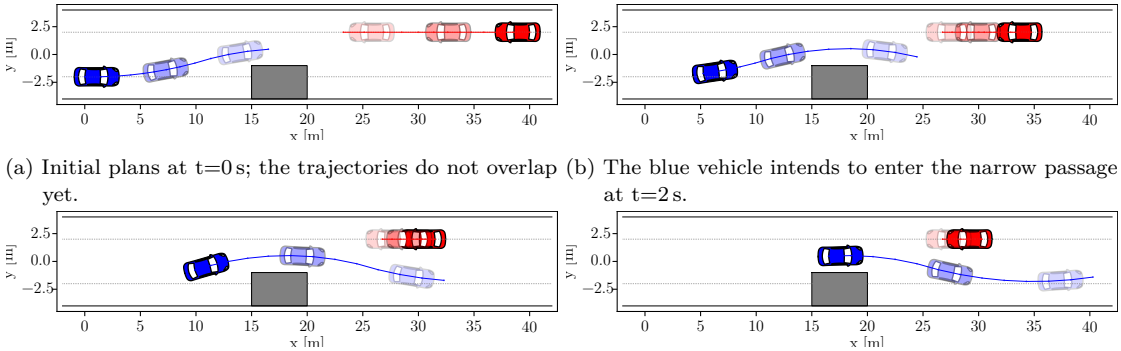
Figure 4.20: Evolution of a two vehicle scenario with the oncoming vehicle 2 keeping constant velocity on a Cartesian plot. Dark to light colors depict increasing time. As the behavior of vehicle 2 is unclear a feasible solution is chosen if vehicle 2 accelerates or not. The evasive maneuver is started after vehicle 2 has passed the obstacle.

Uncooperative Behavior

In case the oncoming vehicle 2 is not cooperating and keeps a constant speed of 4 m/s, vehicle 1 has to stop and pass the obstacle after the oncoming vehicle 2. The evolution of this scenario is depicted in Figure 4.20. Vehicle 1 decelerates and awaits if vehicle 2 gives way. As vehicle 2 does not show cooperative behavior, vehicle 1 brakes and continues at low speed until vehicle 2 has passed the obstacle after t=10 s. Note that for vehicle 2 we show the planned, and not the actual trajectory here, the deviations from the true speed of vehicle 2 of 4 m/s reflects the prediction error in the unknown true motion. We observe that also in the presence of a non-cooperating vehicle, our approach yields a safe solution.

Aggressive Ego Behavior

Giving vehicle 1 a more aggressive parameterization, it accelerates into the gap even with vehicle 2 approaching. In this experiment, visualized in Figure 4.21 we model vehicle 2 to stay on the reference line and decelerate once its trajectory intersects with the constant velocity and steering



(a) Initial plans at t=0 s; the trajectories do not overlap yet.

(b) The blue vehicle intends to enter the narrow passage at t=2 s.

(c) The red vehicle creeps forward to eventually come to a complete stop at t=4 s.

(d) The blue vehicle passes the narrow passage and makes room for the red vehicle at t=6 s.

Figure 4.21: Evolution of a two vehicle scenario with with aggressive parameterization of the ego vehicle 1 on a Cartesian plot. Dark to light colors depict increasing time. As vehicle 1 proactively enters the narrow passage with its desired velocity, vehicle 2 is forced to stop in front of the passage and wait for vehicle 1 to pass.

Table 4.4: Time needed for both vehicles to finish the scenario.

	Vehicle 1	Vehicle 2
Cooperative evasive	12 s	10 s
Cooperative speedchange	14 s	12 s
Uncooperative other	22 s	9 s
Aggressive ego	11 s	16 s
Vehicle 1 only	11 s	-
Vehicle 2 only	-	9 s

prediction of the state of another vehicle, resulting in passive behavior. While at t=0 s, the solution appears balanced as the trajectories do not overlap yet, starting from t=2 s vehicle 2 brakes and vehicle 1 can enter the passage first, achieving its own goal to not deviate from the reference speed.

Discussion

The time to pass the obstacle can serve as a measure of cooperation. Table 4.4 states the time it takes for both to pass the obstacle and to travel with the targeted speed on the respective reference line again in the three discussed scenarios. As a baseline, we simulate the scenario for each vehicle individually. By cooperation, clearly, a balanced behavior is achieved.

To quantify how cooperative a maneuver has been performed, we evaluate the normalized contribution of both vehicles to the total costs [DP14]. The lower the absolute cost values are for a vehicle, the more desirable is the solution for the respected vehicle. Figure 4.22 shows the individual cost contributions of vehicle 1 divided by the cost contributions of vehicle 2 per step. Cooperative solutions should be close to the ideal value of 1.0 as no vehicle is favored according to the cost function. Deviations from 1.0 indicate favor towards one vehicle. In our example, for the uncooperative case the cost contribution of vehicle 1 is significantly higher than for vehicle 2. Hence the behavior of vehicle 2 can clearly be classified as egoistic. For the cooperative solution, the absolute cost values per vehicle are close, and parameter adaptions can leverage a slight favor towards one vehicle. This example illustrates how we distinguish between cooperative and uncooperative behavior and that we can generate a safe behavior in either setting.

Summing up, we observed in this scenario that using a modified parameterization, different reasonable, cooperative behaviors can be achieved. Also, if an agent does not show cooperation, our algorithm still finds a safe solution.

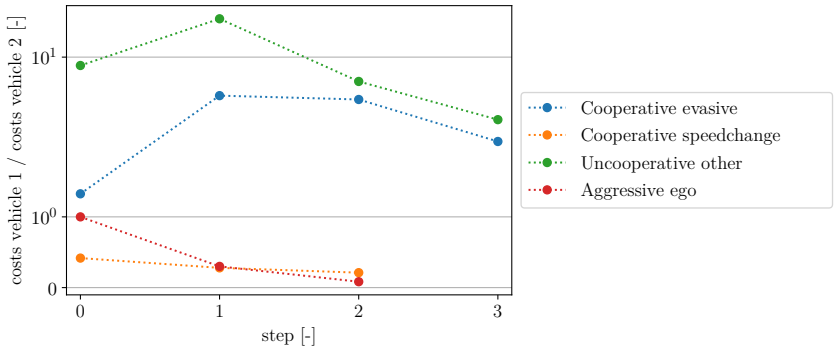
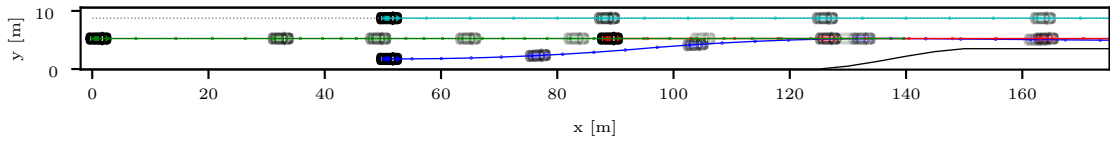
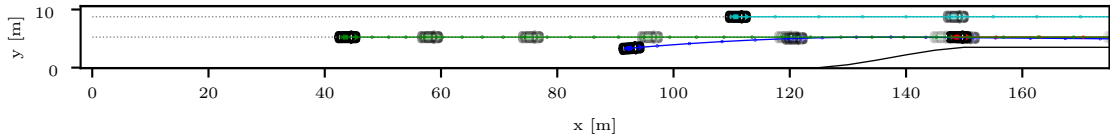


Figure 4.22: Relative contributions of both vehicles to the total costs in the different settings in the two-vehicle scenario. 1.0 is an ideal symmetric cost contribution of each vehicle, greater values indicate high cost contribution of vehicle 1, lower values indicate high cost contribution of vehicle 2. Once both vehicles do not influence each other any more we stop the evaluation



(a) The blue vehicle plans a merging trajectory and the green vehicle allows this merge by decelerating at t=0 s.



(b) The blue vehicle can successfully merge at t=2 s without stopping the green vehicle.

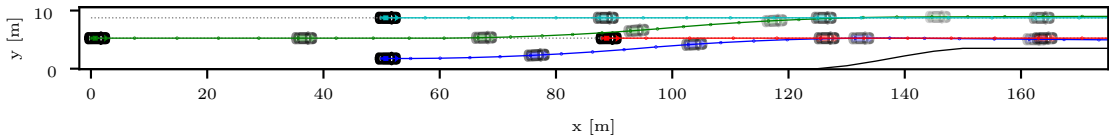
Figure 4.23: Evolution of a cooperative highway scenario. The ego vehicle (blue) on the ramp needs to merge into the mainline. The rear vehicle (green) decelerates to let the ego vehicle merge. Dark to light colors depict increasing time. The trajectories start at the current vehicle positions. The reference lines on the two lanes are shown in dashed gray.

4.8.3 Cooperative Planning in Highway Traffic

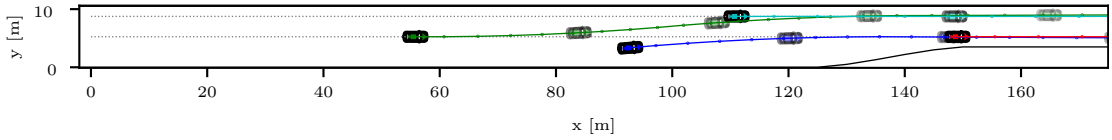
In this experiment, we will show the performance of the algorithm in a highway scenario where the ego vehicle has to merge from a ramp on the mainline into moving traffic. The scenario is defined in a way that if all vehicles greedily follow their desired velocity on the optimal reference track, collisions would occur or the ego vehicle would have to come to a complete stop. To avoid collisions at high speeds, the subsampling stepsize is set to 0.25 s.

Solutions with Coordination

First, we show two settings where all agent cooperate. According to the parameterization different reasonable cooperative behaviors can be achieved. In the first setting (Figure 4.23) we chose the cost terms and the cooperation factor λ that the green vehicle behind the blue ego vehicle slows down to let the ego vehicle merge safely after the red vehicle has passed. By putting less weight on tracking a specific lane, the green vehicle behind the blue ego vehicle performs a lane change on the left lane after the cyan vehicle on this lane has passed to let the ego vehicle merge. The evolution of the scenario is depicted in Figure 4.24.

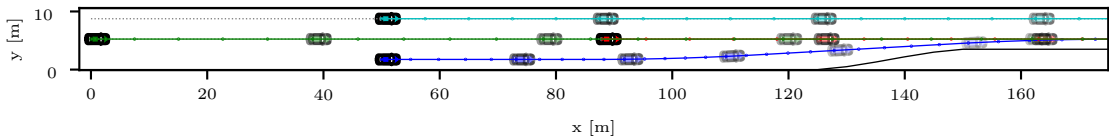


(a) The blue and the green vehicle plan a coordinated lane change at t=0 s.

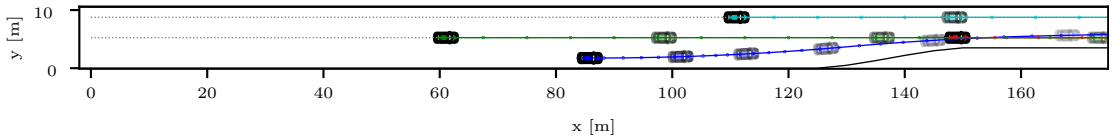


(b) The coordinated lane change is safely executed at t=2 s.

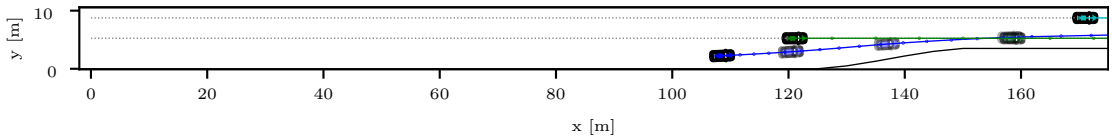
Figure 4.24: Evolution of a cooperative highway scenario. The ego vehicle (blue) on the ramp needs to merge into the main line. The rear vehicle (green) performs a lane change to the left lane taking into account the cyan vehicle there to let the ego vehicle merge. Dark to light colors depict increasing time. The trajectories start at the current vehicle positions. The reference lines on the two lanes are shown in dashed gray.



(a) The blue vehicle plans an initial merging trajectory at t=0 s.



(b) The blue vehicle awaits cooperation from the green vehicle to perform the lane change later at t=2 s.



(c) The blue vehicle decelerates to merge after the green vehicle at t=4 s.

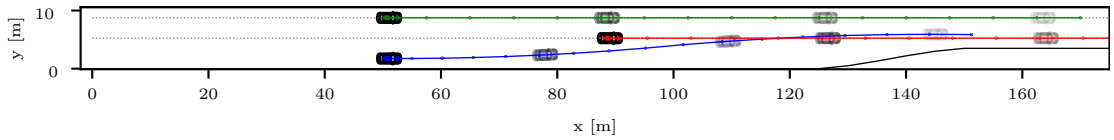
Figure 4.25: Evolution of a non-cooperative highway scenario. The ego vehicle (blue) on the ramp needs to merge into the main line but neither the rear (green) nor the front (red) vehicle shows cooperation. The ego vehicle therefore decelerates and merges after the rear vehicle has passed. Dark to light colors depict increasing time. The trajectories start at the current vehicle positions. The reference lines on the two lanes are shown in dashed gray.

Non-cooperative Solution

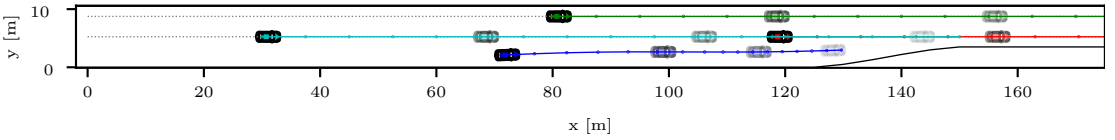
Second, we show a non-cooperative setting in Figure 4.25. Here the three vehicles on the mainline keep their constant velocity. Still, the (blue) ego vehicle awaits cooperation. After the first timestep t=0 s, the reflection adapted the cost terms that the ego vehicle merges after the green vehicle.

Safe Behavior with Limited Sensor Sight

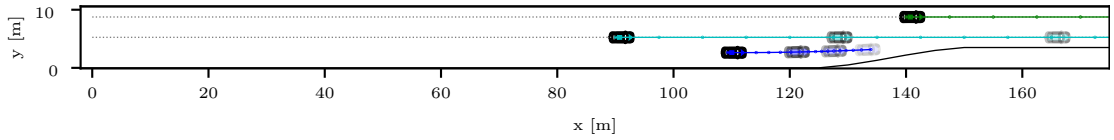
As a third experiment, we model an insufficient sensor sight of the (blue) ego vehicle to the rear. The approaching vehicle on the right lane is not visible at t=0 s, only if it approached closer than 50 m, it is observed and included in the planning. This approaching vehicle does not show cooperation. In Figure 4.26 we observe that first, the ego vehicle initiates a lane change on the



(a) The blue vehicle plans a lane change without being aware of the green vehicle in the back at t=0 s.



(b) At t=1 s, a new, fast vehicle (green) is perceived behind the blue vehicle.



(c) The lane change is aborted and the blue vehicle decelerates heavily at t=3 s. Still, the behavior is safe.

Figure 4.26: Evolution of a highway scenario with limited sensor sight. The ego vehicle (blue) on the ramp needs to merge into the main line but is unaware of the approaching vehicle from rear (green) at time t=0 s. After having initiated the lane merge trajectory, the vehicle from rear is observed and the maneuver is aborted by braking and steering back on the ramp. Dark to light colors depict increasing time. The trajectories start at the current vehicle positions. The reference lines on the two lanes are shown in dashed gray.

mainline that it aborts once the additional vehicle is perceived. By decelerating and staying on the ramp, the ego vehicle can still achieve safe behavior. While this behavior is not desired, it is still necessary for a planner to also cope with such situations. When the in-vehicle sensors are not sufficient to observe the situation precisely could be resolved by infrastructure sensors and Vehicle-to-Everything (V2X) communication [Krä+22].

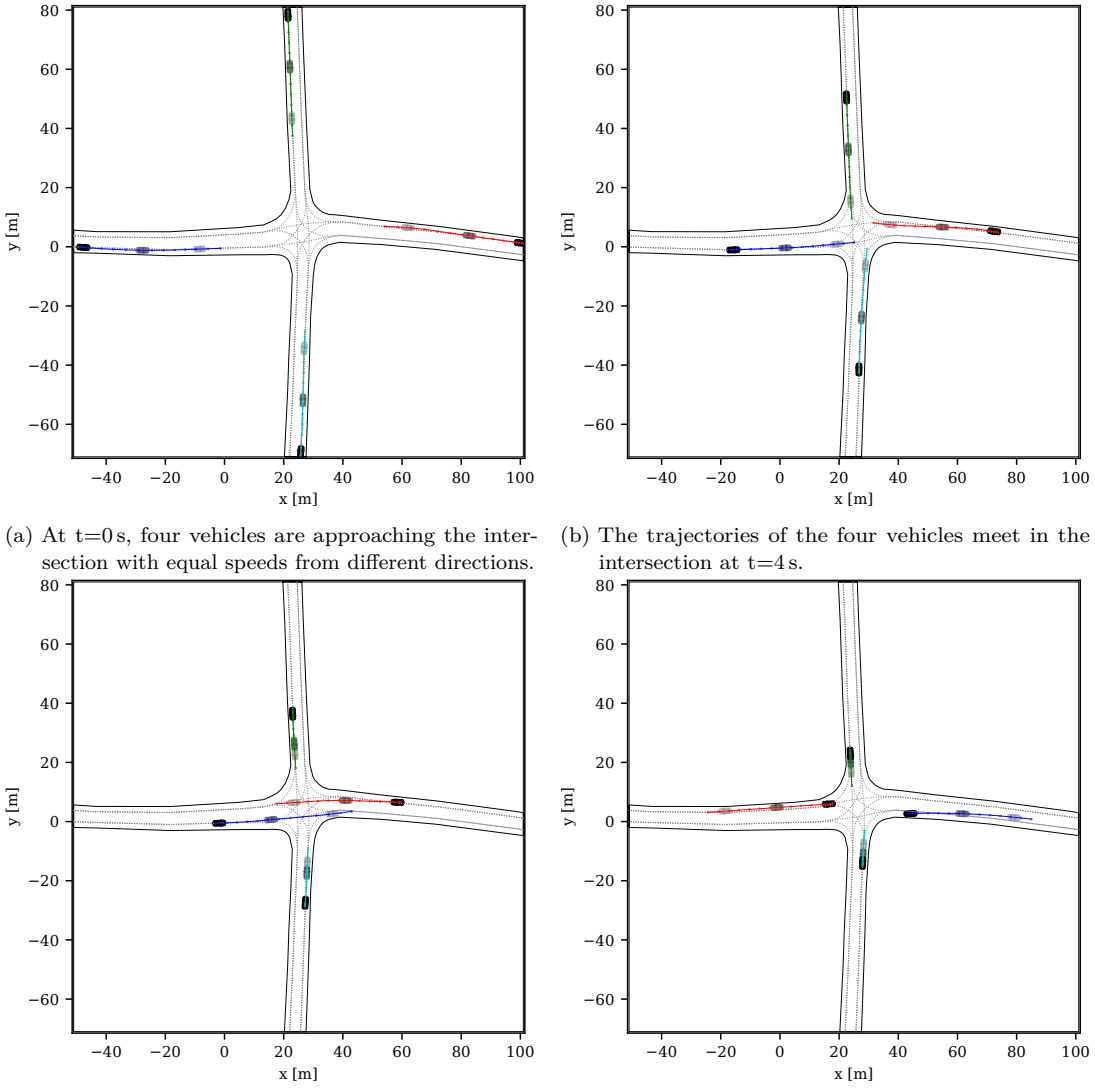
Discussion

These experiments exemplarily show several benefits of ACCORD: Due to the collision checking strategy in-between decision states, it can operate safely at high speeds without introducing over-conservative behavior. Also, it can cope with unexpected situations and low-quality estimations of other traffic participants. Most importantly, we observe that our approach creates cooperative behavior like human drivers do today. Changing a lane on a highway to let vehicles merge from a ramp is, for example, a common driving strategy. So we have transferred human cooperation strategies to algorithms making the agents safely exchangeable.

4.8.4 Cooperative Intersection Management

This example demonstrates how traffic at an unsignalized intersection can be coordinated safely using cooperative behavior. All vehicles communicate and coordinate their behavior. Ignoring classical right-of-way rules, the algorithm generates a solution such that each vehicle tracks its reference speed as close as possible and reaches its desired destination as fast as possible without collisions. Note that collisions occur if all vehicles keep a constant reference speed on their reference tracks, or vehicles would have to brake harshly. The intersection in this example is asymmetric as in the east-west road, vehicles have to take a slight S-curve and is located near Munich [KA17; AKM17].

We generate a conflicting situation by placing four vehicles with the same initial speed on each road leading to the intersection with the same distance from the intersection center. Figure 4.27



(a) At t=0 s, four vehicles are approaching the intersection with equal speeds from different directions.

(b) The trajectories of the four vehicles meet in the intersection at t=4 s.

(c) The North/South driving vehicles (green and cyan) decelerate at t=6 s.

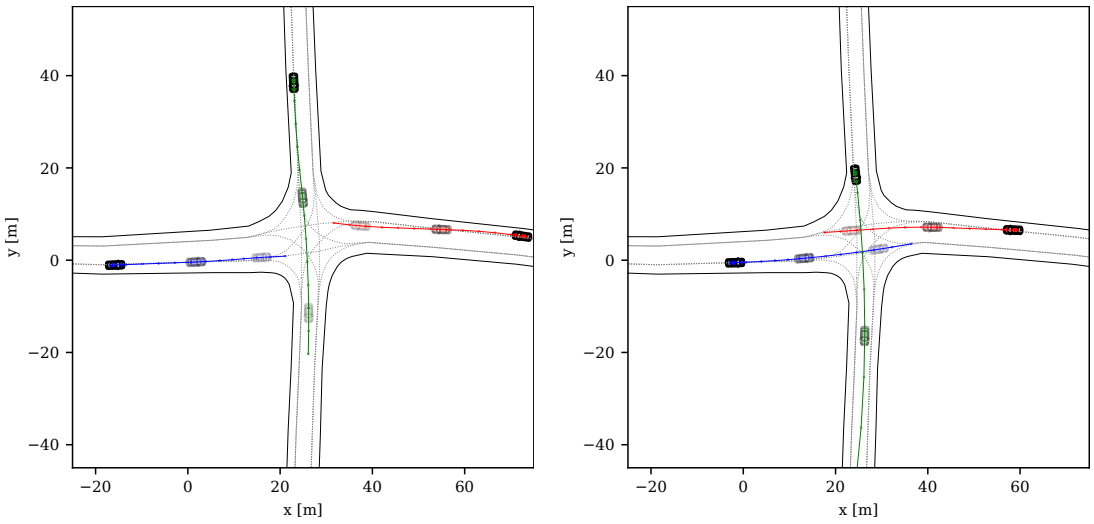
(d) The East/West driving vehicles (red, blue) pass the intersection and the others accelerate at t=12 s.

Figure 4.27: Evolution of a cooperative intersection scenario with four vehicles with equal priority approaching an intersection. Dark to light colors depict increasing time.

shows the scenario. We choose an equal cooperation factor λ and equal cost terms for each vehicle. Due to the slight asymmetry of the intersection, the optimizer favors letting the vehicles moving on the east-west road pass first. Both vehicles on the north-south road decelerate to let the other vehicles pass.

When varying the cooperation factor λ and the initial states of the vehicles, the evolution of the scenario varies. We here show an extreme case in a three-vehicle scenario with $\lambda = [0.01, 0.01, 0.98]$. An application could be to assign high priority to an emergency vehicle. In the example in Figure 4.28, a faster, high-priority vehicle approaches the intersection from the northern direction, and both vehicles on the east-west road brake to let the high priority vehicle pass. As the motion is globally coordinated, both low priority vehicles do not have to brake to a standstill until the high priority vehicle has passed but can pass the intersection right after it has been cleared.

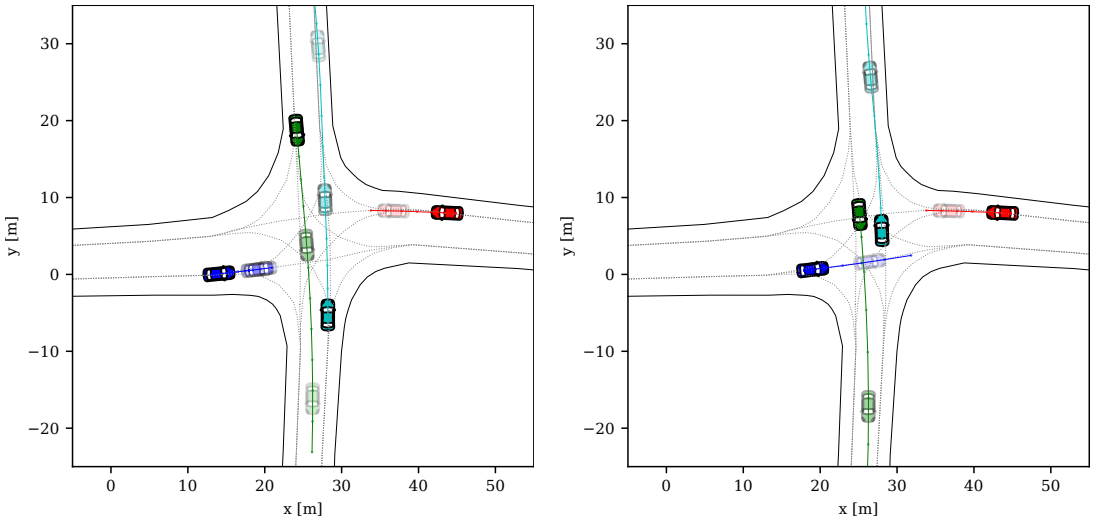
As a slight modification of the scenario, we consider a four-way stop as it is common, e.g., in the USA. The (US) legal rules that the vehicles cross the intersection in the order of arrival at



(a) The trajectories of all vehicles meet on the intersection at t=4 s.

(b) The green vehicle accelerates and the red and blue vehicle decelerate at t=6 s.

Figure 4.28: Evolution of a cooperative intersection scenario with three vehicles approaching an intersection, one (green, north) having higher priority. Both other vehicles give way so the high priority vehicle can pass first. Dark to light colors depict increasing time.



(a) The north/south direction goes first, but in parallel at t=2 s.

(b) The red and blue vehicle creep towards the intersection at t=4 s and await the other vehicle to pass.

Figure 4.29: Evolution of a cooperative intersection scenario with four vehicles at a four-way stop intersection. The coordination algorithm finds an optimal order for the vehicles to pass the intersection. Dark to light colors depict increasing time.

the intersection are not efficient in terms of throughput. In Germany, the right before left rule results in similar efficiency drawbacks and leaves ambitiousness with four vehicles at an intersection. Cooperative driving offers the possibility to resolve such situations fairly. With equal cooperation factors, λ the asymmetry in the scenario decides on the most efficient resolution. On the one hand, this takes up the driver's behavior to let the vehicle go first, which is considered the most effective. On the other hand, the enhanced knowledge from communication can be used for a more efficient

resolution, as non-conflicting vehicles can go in parallel regardless of right-of-way rules. Figure 4.29 depicts an exemplary scene.

Summing up, even at a fairly simple intersection, cooperative behavior can help to reduce waiting times and therefore increase throughput. The balance between the intents of the different vehicles is achieved by appropriately setting the cooperation factor λ. If each vehicle communicates and cooperates, no infrastructure or fixed traffic rule set is needed.

4.8.5 Cooperative Valet Parking

In Section 4.6, we introduced a trajectory coordination approach adapted for the specific needs of automated valet parking in the presence of non-autonomous vehicles. We will, in this section, show the performance of the algorithm in two examples. Each trajectory has a horizon length of 10 s. The chosen parameterization rewards choosing short and fast trajectories and penalize direction changes. This section only shows two settings without connected vehicles. As we work with pre-calculated paths in a connected setting, the optimal setting is easily reached if all vehicles follow the optimized plan.

Performance Measures

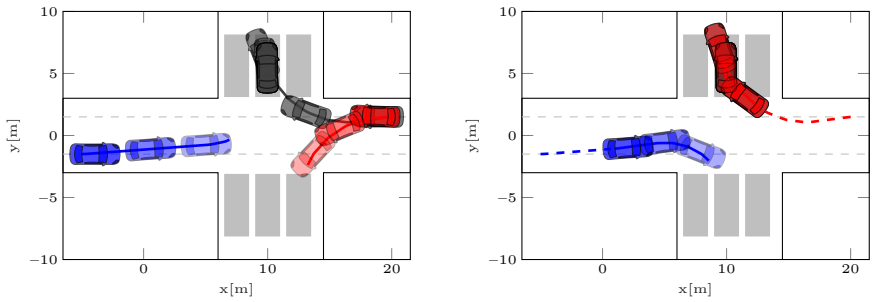
As a ground truth and performance benchmark, we resolve the scene with all vehicles following the optimized plan. We further compare the solution of the coordination algorithm to the scenario as if each participating vehicle would be alone in the scene. Non-cooperating vehicles follow a predefined motion without taking others' behavior into account. In the case of such a non-cooperating vehicle, we a posteriori compute the costs of this potentially aggressive and globally sub-optimal behavior. Most natural evaluation factors are already part of the optimization's objective function and hence can be leveraged with appropriate parameterization. Therefore, these may not serve as evaluation criteria; we instead define the following: By $J(v, i)$ we denote the cost function value contributed by vehicle v in cycle i that was computed in the optimization. $\hat{J}(v, i)$ is defined as the actual cost contribution of vehicle v in cycle i that is calculated a posteriori from the taken trajectory's properties and the estimated intentions. We define the fraction of both as $\iota = J/\hat{J}$. An overline denotes the respective mean. We choose the following performance measures:

- The time T until a set of trajectories is found so that each vehicle reaches its desired position without conflicts,
- the total $\sum \hat{J}$ and average $\overline{\hat{J}}$ cost function values giving a performance measure of the coordination algorithm in relation to the reference scenarios, and
- the ratio of observed costs to optimized costs ι as an indicator how accurate the intention has been and how aggressive/defensive non-cooperative vehicles behave in relation to the ego-vehicle.

An Illustrative Two-vehicle Example

We consider an easy coordination situation with two vehicles on a parking lot (see Figure 4.30). The autonomous ego vehicle 1 (blue) wants to park and vehicle 2 (red) has the fixed but unknown intention to park at its right. Both vehicles have various reasonable driving options. Note that in contrast to the scenarios above, reverse driving is allowed.

From the cost function values presented in Table 4.5, we observe that the global optimal solution leverages the interests of both vehicles. In the non-cooperative case, the total optimization costs are roughly the costs of both vehicles if they were alone in the scene. This is expected, as the optimal solution is straightforward after the first iteration. $\sum_{v,i} \iota(v, i) = 0.96$ and $\overline{\iota(2, i)} = 0.79$ indicate that with a non-cooperative vehicle present a worse solution is found compared to the



(a) Initial plans with a wrong estimated intention of the red vehicle at t=0 s.

(b) Updated plan with corrected intention of the red vehicle at t=2 s.

Figure 4.30: Evaluation of a two-vehicles parking lot maneuvering scenario. The free parking spaces and goal poses are depicted as gray patches, the reference lines along the lane centerlines as dashed gray. The autonomous left ego vehicle 1 is colored blue and the non-communicating right vehicle 2 is colored red. In blue and red the respective planned trajectories are shown, in gray the (unknown) trajectory vehicle 2 is going to take. Already traveled trajectories are dashed. (modified graphic from [KK17], ©2017 IEEE)

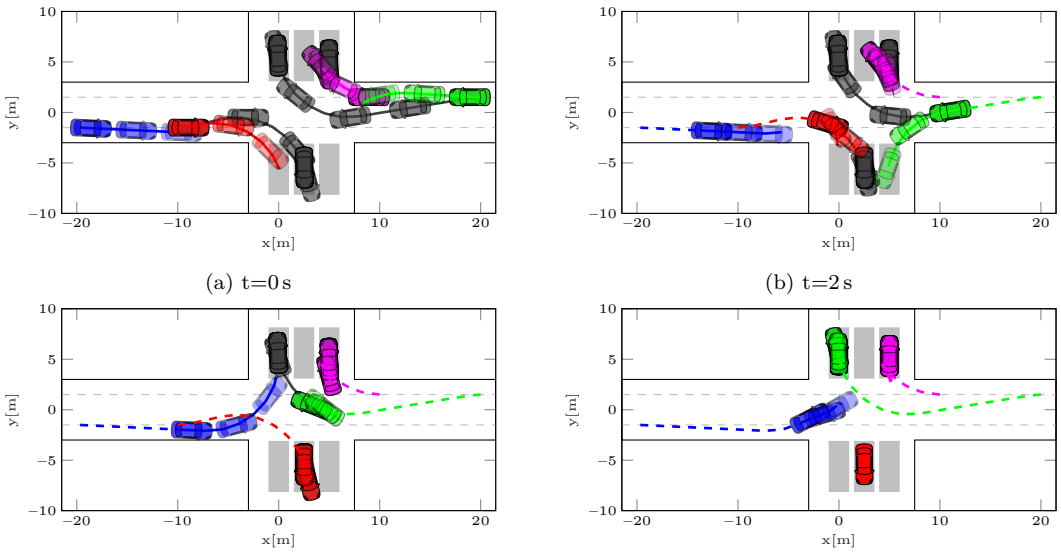
Table 4.5: Cost and time evaluation of the illustrative two-vehicle example.

	T	$\sum_{v,i} \hat{J}(v,i)$	$\overline{\hat{J}(v,i)}$
Vehicle 1 alone	8 s	421.8	105.5
Vehicle 2 alone, cooperative traj.	8 s	296.5	74.1
Vehicle 2 alone, non-cooperative traj.	10 s	635.4	127.1
Global Optimum	8 s	716.1	69.5
Non-Cooperative	10 s	1178.9	117.9

global optimum. The qualitative evolution of the scene is depicted in Figure 4.30. Comparable results are achieved if vehicle 2 has the intention to go straight instead of choosing a parking spot.

Safe Maneuvering Into a Parking Spot With Inaccurate Initial Knowledge

As a second example scene, we consider a parking lot scenario with 3 free parking spaces to the left and 3 to the right of a two-lane road. These are the possible goal positions. Vehicle 1 (leftmost, blue) is the autonomous ego vehicle under test. All other vehicles 2-4 act non-cooperative. All four vehicles want to park. The evolution of the scenario is depicted in Figure 4.31. We observe that the ego vehicle first (until t=4 s) targets a parking spot that is then occupied by another vehicle and the ego vehicle has to maneuver to navigate into a neighboring parking spot. Also, initially, the estimations on the vehicle intentions are incorrect. The initial scene is depicted in Figure 4.31a including the vehicle plans in the first step (colored) and the unknown actual plans (gray). The evolution of the scene is depicted in Figure 4.31, this solution is compared to the references in Table 4.6. The higher total costs in the non-cooperative scenario are mainly due to the higher number of iterations to finish the scenario. The mean values have a comparable magnitude. The mean cost function values $\overline{\hat{J}}$ of the vehicles 2-4 in the optimized and egoistic cases indicate that the estimations on the trajectories differ from the actually taken trajectories. As these egoistic trajectories also may have non-optimal goals, more iterations and higher costs occur. Regarding the cost function ratios ι, a deviation of optimized and non-cooperative value of $\sum_{v,i} \iota(v,i) = 0.9$ is obtained. The mean values per vehicle v_j, $\overline{(\iota(v_j,i))}$ vary between 0.84 and 0.97 depending on how much the optimized and the predefined egoistic trajectories deviate. Regarding the evolution of the costs over iterations, both the egoistic and the optimal trajectories show similar trends.



(a) t=0 s (b) t=2 s

(c) t=4 s (d) t=6 s

Figure 4.31: Evaluation of a parking lot scenario where three non-cooperative vehicles maneuvering into a parking spot without showing reaction to the ego vehicle. The blue (leftmost) vehicle 1 is the ego vehicle, all others are non-communicating vehicles. The respectively colored trajectories are the optimized ones, the dark gray ones the (unknown) trajectories the vehicles are going to take. The history trajectories are dashed. The available parking spots and the reference lines are depicted in light gray. In the subsequent steps, the ego vehicle maneuvers into the spot between the green and the magenta vehicle. The ego vehicle reactively lets the other vehicles chose a parking spot first and plans a non-conflicting behavior moving towards a non-occupied parking spot. (modified graphic from [KK17], ©2017 IEEE)

Table 4.6: Cost and time evaluation of the parking lot maneuvering scenario.

	#It	$\sum_{v,i} \hat{\Phi}(v,i)$	$\overline{\hat{\Phi}(v,i)}$
Vehicle 1 alone	8 s	658.4	164.6
Vehicle 2 alone, optimal traj.	8 s	338.2	84.6
Vehicle 2 alone, non-cooperative traj.	6 s	322.7	107.6
Vehicle 3 alone, optimal traj.	6 s	335.2	111.7
Vehicle 3 alone, non-cooperative traj.	8 s	259.2	64.8
Vehicle 4 alone, optimal traj.	10 s	582.7	116.5
Vehicle 4 alone, non-cooperative traj.	8 s	513.4	128.4
Global Optimum	10 s	1852.9	118.8
Non-Cooperative	14 s	2105.6	117.0

We conclude that safe maneuvering into a parking spot is possible, even with inaccurate initial estimations of the other agents' intentions and motions.

4.9 Conclusion

We, in this chapter, introduced the ACCORD approach, which solves the multi-agent behavior planning problem by spanning a tree of possible, discrete actions of each agent and finding an optimal trace through these trees using MILP. By taking into account the leveraged interest of each agent in the optimization, the approach is inherently cooperative and aims to coordinate the motion of each agent in a global optimal fashion. To account for non-optimal behavior of

non-autonomous agents, we, in each receding horizon step, adapt the weights of the optimization to fit the observations best.

By leveraging own and others' interests, the coordination approach generates driving behavior inspired by current traffic. For example, small changes in the ego vehicle behavior can result in significant benefits for other agents. An example is performing a lane change on a highway to let others merge from a ramp. Also, unbalanced situations can be handled cooperatively, such as letting an emergency vehicle pass. Nevertheless, the behavior is not human-like: Often, cooperative behaviors evolve that make optimal use of the available space without accounting for traffic rules of conventions, which is necessary to include in the constraints. The approach is scenario-agnostic, as demonstrated in a low-speed negotiation scene, a high-speed highway scenario, and an intersection scenario. For parking scenarios, we presented the adapted ACCORD-P algorithm with fewer degrees of freedom. It reflects the observation that the variety of possible behaviors on parking lots is limited and produces optimally coordinated motions, also with non-cooperating vehicles present. We showed that the worst-case runtime scales exponentially with densely interacting agents, mainly as all agents have to be checked for collision pairwise. This effect is critical for real-time applications. With two or three agents in interactive scenarios, the runtime is tractable in the number of possible decisions per step, and the optimization steps are limited.

5 MINIVAN: Dynamic Games on a Continuous Action Space

Executive Summary of this Chapter This chapter introduces the planning algorithm Mixed Integer Interactive Planning (MINIVAN), solving the continuous-time formulation of the multi-agent planning problem to optimality using Mixed-Integer Quadratic Programming (MIQP), based on a linear differential game definition. This requires a problem formulation with only linear constraints that will be derived here. We introduce a generic, globally valid linearization approach with correct, non-holonomic vehicle dynamics and show the approximation error introduced is velocity-dependent with positional errors up to 0.3 m at low velocities decreasing to less than 0.1 m at higher velocities, depending on the parameterization. Most state-of-the-art Mixed-Integer Progamming (MIP)-based planning algorithms lack this global validity. We introduce the methodology to compute all model parameters by linear least-square fits. Comparing the proposed method to a nonlinear reference implementation, we study the feasibility of the proposed model. We further introduce linear, over-approximating collision constraints for road boundaries, static and dynamic obstacles, and fellow (vehicle) agents and show that these effectively avoid collisions. Introducing soft constraints for collision avoidance handles inaccurate models, perception, and prediction of agents and obstacles. To introduce cooperation into the approach, we formulate a joint cost function, leveraging the interests of each agent using a cooperation factor and show that it effectively scales the computed behavior between egoistic, cooperative, and altruistic. MINIVAN can hence safely interact with other agents while still achieving its own goals.

Besides showing the validity of the computed trajectories and evaluating the effectiveness of the planner, we also show the real-time capability of the approach. We elaborate which restrictions apply based on the analysis of test drives and a thorough complexity analysis. We show that providing a warmstart solution to the solver is essential for real-time operation, as it lowers the runtime by 30 % to 50 % on average. By choosing a parameterization fitting problem structure and solution requirements, again 15 % to 30 % of the runtime can be saved on average. These numbers are even increased when choosing a scenario-specific parameterization. We further show that a realistically high number of static and dynamic obstacles can be handled in real-time, and the combination of our performance improvements effectively stops the (theoretically expected) exponential growth in runtime with respect to the number of obstacles, horizon length, or further properties increasing the model size and complexity. In a multi-agent setting, however, we cannot avoid the exponential growth, and therefore three interacting agents are an upper limit for real-time application. In environments where the solution of the single-agent MIQP already requires a majority of the available solution time, such as narrow, sharp curves, the computation time of a multi-agent MIQP is not tractable. In various demonstration scenarios, such as overtaking, intersection handling, and competitive racing, we show that MINIVAN effectively leverages the level of cooperation between different agents being robust against perception and prediction errors.

Content and Structure of this Chapter MIP has the ability to solve the multi-agent behavior problem to optimality, but requires a linear (vehicle) model. Based on a triple integrator model formulation, we compute the orientation of the vehicle and model it in a disjunctive manner for arbitrary vehicle states and orientations. That allows us to formulate linear constraints to account for the non-holonomy and collision avoidance. These constraints are approximations, for which we introduce the theory in Section 5.1 and Section 5.2.

We further formulate the optimization program to compute trajectories for multiple agents. By taking into account the intent of every agent, the ego agent can incorporate future interactions with human-driven vehicles into its planning. In Section 5.3 we define the MIQP-formulation, based on the definition of a linear differential game and solve this problem to optimality. We then define a joint cost function, where a cooperation factor can adapt between altruistic, cooperative, and egoistic behavior.

Section 5.4 shows, that the formulated linear model produces curvature-correct trajectories. These can be tracked by a non-holonomic vehicle model without causing collisions. As the real-time applicability of MIP-based planning algorithms is challenging, Section 5.5 elaborates on the measures we implemented to meet real-time requirements. We assess how these strategies help to lower the runtime in simulated and real-road driving scenarios. The complexity analysis in Section 5.6 analyzes how the solution time of the program scales with an increased number of horizon steps, objects, or agents present.

Section 5.7 shows how MINIVAN can resolve straight forward driving situations such as highway driving including overtaking, as well as more complex applications such as negotiating limited spaces or dense merging. Section 5.8 concludes this chapter.

Contributions of this Thesis The models and results of this chapter are based on the joint work with Klemens Esterle [EKK20; KEK20]. The model formulation in a single-agent context is introduced in [EKK20] and the extension to a multi-agent setting in [KEK20]. The main contribution of this chapter is to elaborate on the strategy to make the model applicable in real-time and evaluate the effect of different measures. We further extend the publications by a complexity analysis of the problem formulation, a broader variety of simulated scenarios including the evaluation of the solution quality and a detailed assessment on the introduced approximation errors.

5.1 Region-Based Linearization Approach

Following Qian et al. [Qia+16], we model the vehicle as a third-order point-mass system with positions $p_x(t)$, $p_y(t)$, velocities $v_y(t)$, $v_y(t)$, and accelerations $a_x(t)$, $a_y(t)$ as states. Jerk in both Cartesian coordinates $j_x(t)$ and $j_y(t)$ are inputs of the model. As we aim to formulate the vehicle model as linear constraints, all nonlinearities have to be eliminated. In the following, we will introduce how we guarantee the validity of the model for all orientations and perform collision checks.

5.1.1 Discretized and Disjunctive Modeling of the Orientation

Although the vehicle's orientation θ is not part of the state space, we will need it for a sufficient collision check in Cartesian coordinates within the optimization problem. We assume perfect traction and therefore neglect vehicle and tire slip. It can be calculated using $\theta = \text{atan}(v_y/v_x)$. However, this equation is nonlinear, as are the trigonometric operations

$$\sin(\theta) = \sin(\text{atan}(v_y/v_x)) \tag{5.1a}$$

$$\cos(\theta) = \cos(\text{atan}(v_y/v_x)) \tag{5.1b}$$

are necessary to calculate the front axle position. To formulate collision constraints, we thus need to linearize (5.1). For our model to be valid for orientations $\theta \in [0, 2\pi]$, we discretize the orientation by introducing *regions* in the (v_x, v_y) plane, see Figure 5.1. The regions are defined by

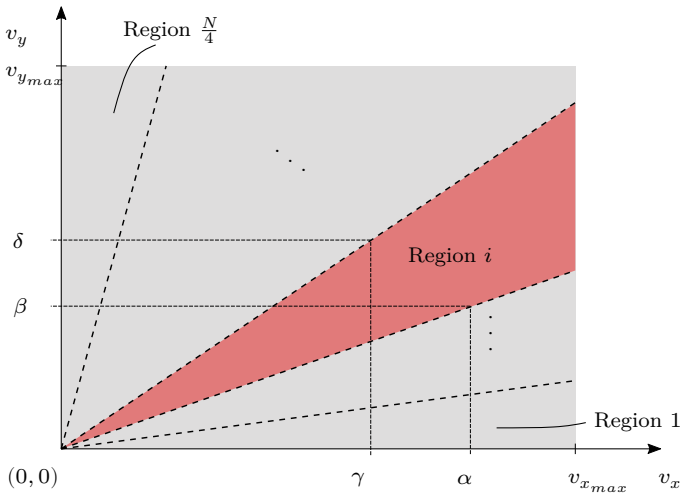


Figure 5.1: Construction of region i described through two lines $(0,0)-(\alpha,\beta)$ and $(0,0)-(\gamma,\delta)$ following (5.2). (graphic from [EKK20], ©2020 IEEE)

Table 5.1: Parameters from fitting or preprocessing used throughout this work. Region dependency is denoted by $\square^r$.

Parameter	Description
α^r	x value of region r lower region borderline
β^r	y value of region r lower region borderline
γ^r	x value of region r upper region borderline
δ^r	y value of region r upper region borderline
$\overline{\mathcal{P}^r_{\cos}}$	Polynome linear in v_x, v_y upper-bounding $\cos(\theta)$
$\underline{\mathcal{P}^r_{\cos}}$	Polynome linear in v_x, v_y lower-bounding $\cos(\theta)$
$\overline{\mathcal{P}^r_{\sin}}$	Polynome linear in v_x, v_y upper-bounding $\sin(\theta)$
$\underline{\mathcal{P}^r_{\sin}}$	Polynome linear in v_x, v_y lower-bounding $\sin(\theta)$
$\underline{\mathcal{P}^r_{\kappa}}$	Polynome linear in v_x, v_y lower-bounding the κ
$\overline{\mathcal{P}^r_{\kappa}}$	Polynome linear in v_x, v_y upper-bounding the κ
$\underline{u^r_x}$, $\overline{u^r_x}$	Lower and upper jerk limit in direction x
$\underline{u^r_y}$, $\overline{u^r_y}$	Lower and upper jerk limit in direction y
$\underline{a^r_x}$, $\overline{a^r_x}$	Lower and upper acceleration limit in direction x
$\underline{a^r_y}$, $\overline{a^r_y}$	Lower and upper acceleration limit in direction y
ϱ^r	Region that r is allowed for the current scenario

the area between two line segments starting at the origin. Consequently, for every (v_x, v_y) point within a region i, the following inequalities hold with region-dependent line parameters:

$$\alpha v_y \geq \beta v_x \tag{5.2a}$$

$$\gamma v_y \leq \delta v_x \tag{5.2b}$$

Having done this, we will subsequently formulate model equations that are valid in each region with different parameters. These are listed in Table 5.1.

For driving comfort reasons, most motion planners limit the maximum longitudinal and lateral acceleration, deceleration, and jerk. From these desired values in the driving direction of the vehicle, we compute region-specific bounds in global x, y coordinates. This is done by rotating the original longitudinal and lateral limits along with the vehicle orientation. As the orientation angle, we chose the mean angle of the respective region. This ensures that we comply with the original bounds in driving direction in terms of absolute values and directions.

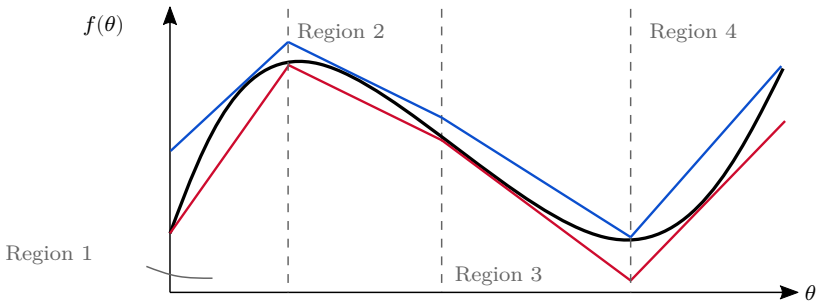


Figure 5.2: Exemplary nonlinear function and the respective piecewise linear upper (blue) and lower (red) bounds (graphic from [EKK20], ©2020 IEEE)

5.1.2 Over-Approximating the Collision Shape

If the orientation is known, a common strategy to approximate the vehicle shape is by using three circles with radius r for the rear axle, middle position, and front axle similar to [Zie+14a], as this allows for efficient collision checking to arbitrary polygons. We aim for a linear vehicle model, so the true (highly nonlinear) orientation is unknown. As we only have an approximated orientation but do not want to underestimate any collisions, we choose to compute the upper and lower bound of the sine and cosine of the orientation. Figure 5.2 illustrates this idea. With that and the vehicle's wheelbase l, we then calculate upper and lower bounds for the x and y position of the front axle.

In practice, we in the fitting process increase the true wheelbase by some centimeters which introduces an additional safety margin at the front left and front right corner of the vehicle.

$$\overline{f_x} := p_x + l\ \overline{\cos(\theta)} \tag{5.3a}$$
$$\underline{f_x} := p_x + l\ \underline{\cos(\theta)} \tag{5.3b}$$
$$\overline{f_y} := p_y + l\ \overline{\sin(\theta)} \tag{5.3c}$$
$$\underline{f_y} := p_y + l\ \underline{\sin(\theta)} \tag{5.3d}$$

Permuting $\overline{f_x}, \underline{f_x}$ with $\overline{f_y}, \underline{f_y}$ yields four circles for the front axle, which represent an over-approximation of the true front axle circle, as shown in Figure 5.3. For now, we chose to not model the middle point of the vehicle, as this increases the complexity of the model. However, a similar approach can be applied to the mid axle. We now present two methods for obtaining bounds for the sine and cosine.

Constant Approximation

We propose a constant approximation of the sine using the maximum and minimum orientation for each region.

$$\overline{\sin(\theta)} \tilde{=} \max[\sin(\operatorname{atan}(\delta^r/\gamma^r)), \sin(\operatorname{atan}(\beta^r/\alpha^r))] \tag{5.4a}$$
$$\underline{\sin(\theta)} \tilde{=} \min[\sin(\operatorname{atan}(\delta^r/\gamma^r)), \sin(\operatorname{atan}(\beta^r/\alpha^r))]. \tag{5.4b}$$

The cosine is calculated accordingly. With a higher number of regions, the error for this type of approximation will decrease. Note that this is only valid as long as a region is not defined over multiple quadrants, since sine and cosine are only monotonic functions within a quadrant.

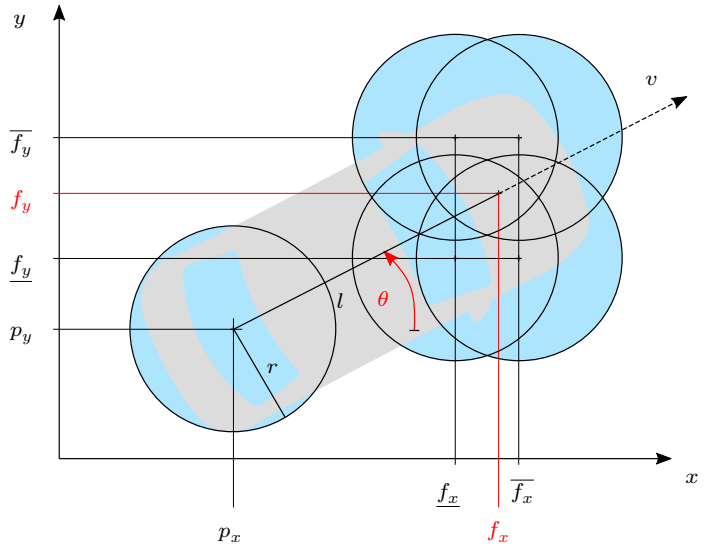


Figure 5.3: Vehicle model with wheelbase l and circle-based collision-shape (cyan) of radius r. The variables in red are unavailable in the MIQP model formulation. The orientation θ is defined clockwise. (graphic from [EKK20], ©2020 IEEE)

Velocity-Dependent Approximation

To preserve the linearity of the constraints, only a linear combination of the state variables can be computed. We chose to upper bound the $\sin(\theta)$ term by a first order polynomial depending on region r with three parameters $p_\square$

$$\overline{\sin(\theta)} \tilde{=} p_{00} + p_{10} v_x + p_{01} v_y := \overline{\mathcal{P}^r_{\sin}}(v_x, v_y) \tag{5.5}$$

and analogously fit such linear polynomials for the lower bound of the sinus $\underline{\mathcal{P}^r_{\sin}}$ function and the bounds of the cosine $\underline{\mathcal{P}^r_{\cos}}$ and $\overline{\mathcal{P}^r_{\cos}}$. The methodology how we compute these parameters $p_\square$ is introduced in Section 5.2. This approximation will lead to a front axle position that depends on the respective velocity terms. However, that is not the case if the orientation can be calculated analytically and leads to high errors for low velocities. The constant approximation is thus intended to be a complementary to the velocity-dependent approximation. Note that in this region the maximum errors differ where on the v_x, v_y grid the bounds are evaluated and the upper bound is much tighter for higher velocities than close to zero velocities.

5.1.3 Modeling the Non-Holonomics

Previous MIQP formulations [Qia+16; ES17; BL18] have approximated the non-holonomics by bounding acceleration in x and y direction and by coupling the velocities via

$$v_y \in [v_x \tan(\theta_{\min}), v_x \tan(\theta_{\max})], \tag{5.6}$$

with $\theta_{\min}$ and $\theta_{\max}$ being the valid orientation range of that model. However, decoupled acceleration bounds cannot yield a non-holonomic behavior. Ziegler et al. [Zie+14a] calculate the curvature using

$$\kappa = \frac{v_x a_y - v_y a_x}{\sqrt[3]{v_x^2 + v_y^2}} \tag{5.7}$$

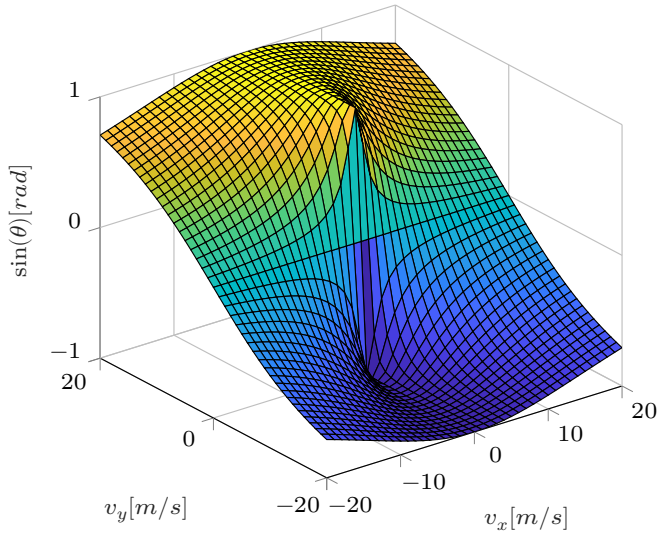


Figure 5.4: Plot of $\sin(\theta) = \sin(\text{atan}(v_y/v_x))$ with respect to v_x and v_y. The function is obviously highly nonlinear but can be approximated in a piecewise linear fashion. (graphic from [EKK20], ©2020 IEEE)

and formulate the bound constraints $\kappa \in [\kappa_{min}, \kappa_{max}]$. However, (5.7) is highly nonlinear and thus curvature constraints cannot be expressed as a linear constraint for MIQP. To obtain constraints dependent on a_x, a_y, we transform $\kappa_{max}, \kappa_{min}$ using (5.7) to

$$\frac{\kappa_{max}}{v_x} \sqrt[3]{v_x^2 + v_y^2} + \frac{v_y}{v_x} a_x \geqslant a_y \tag{5.8a}$$

$$\frac{\kappa_{min}}{v_x} \sqrt[3]{v_x^2 + v_y^2} + \frac{v_y}{v_x} a_x \leqslant a_y. \tag{5.8b}$$

We can then apply the concept of region-wise linearization described in Section 5.1.1 to obtain linear constraints and fit two linear polynomials for upper $\overline{\mathcal{P}_\kappa^r}$ and lower $\underline{\mathcal{P}_\kappa^r}$ bounding the curvature as shown in Section 5.2.

5.2 Fitting Method

In this section, we will briefly describe how we fit the parameters in 5.1.2 and 5.1.3. All fits are done on a region-wise basis and yield polynomials of the form of (5.5).

Fitting the Front Axle Position

In Section 5.1.2, we motivated the need to linearize the trigonometric functions (5.1a) and (5.1b), which depend on v_x and v_y. Both sine and cosine are highly nonlinear, in Figure 5.4 we show the sine function. We formulate the problem to find a piecewise upper bound to a two-dimensional nonlinear function of v_x and v_y as a linear least-squares problem with linear constraints.

To avoid infeasible fits or overconservative bounds, we do not chose the fitting support points beginning from 0 s but from a low speed $\underline{V}$. This significantly improves the quality of the bounds but yields infeasible optimization problems if the vehicle velocity drops below this speed. In Section 5.3.3 we elaborate how we cope with that issue. Also, the maximum velocity has an influence on the tightness of the bounds. We therefore chose a fit with a suitable speed range for the scenarios we evaluate. Adaptive selection and blending between different fitting velocities is possible but was not evaluated in this context.

Table 5.2: Decision variables used throughout this work, with discrete time dependency k, region dependency r, environment dependency λ, and obstacle dependency o

Variable	Description	Range
$p_x(k)$, $p_y(k)$	Vehicle rear axle center position	free
$v_x(k)$, $v_y(k)$	Vehicle velocity in x, y direction	$[\underline{v}, \overline{v}]$
$a_x(k)$, $a_y(k)$	Vehicle acceleration in x, y direction	$[\underline{a}, \overline{a}]$
$u_x(k)$, $u_y(k)$	Vehicle jerk in x, y direction	$[\underline{u}, \overline{u}]$
$\overline{f_x}(k)$, $\overline{f_y}(k)$	Upper bound of the vehicle front axle center position	free
$\underline{f_x}(k)$, $\underline{f_y}(k)$	Lower bound of the vehicle front axle center position	free
$\rho(k, r)$	Region r the vehicle is in at time k	binary
$\Psi_x^+(k)$	No region change allowed helper variable: true if $v_x \leq \underline{V}$	binary
$\Psi_x^-(k)$	No region change allowed helper variable: true if $v_x \geq \underline{V}$	binary
$\Psi_y^+(k)$	No region change allowed helper variable: true if $v_y \leq \underline{V}$	binary
$\Psi_y^-(k)$	No region change allowed helper variable: true if $v_y \geq \underline{V}$	binary
$\Psi(k)$	No region change allowed	binary
$e(k, \lambda)$	Vehicle is not inside the environment sub-polygon λ at time t	binary
$o_p(k, o)$	Vehicle does not collide with obstacle o at time t at the reference point p	binary
$o_{\underline{f}\underline{f}}$, $o_{\underline{f}\overline{f}}$,$o_{\overline{f}\underline{f}}$,$o_{\overline{f}\overline{f}}$	Vehicle does not collide with obstacle o at time t at the respective front upper and lower bounds $\overline{f}$, $\underline{f}$	binary
ξ_x, ξ_y	Slack variables for agent-to-agent collisions	$[0, D(k)]$
ξ_o	Slack variables for dynamic obstacles avoidance	binary
$\theta(k)$	True (unknown) vehicle orientation	$[0, 2\pi]$
$f_x(k)$, $f_y(k)$	True (unknown) vehicle front axle center position	free

Fitting the Curvature

As we discussed in Section 5.1.3, we approximate the curvature constraints by (5.8). We choose to fit the polynomials $\mathcal{P}_\kappa^r$ on

$$\frac{\kappa_{\max}}{v_x}\sqrt[3]{v_x^2 + v_y^2} \equiv \overline{\mathcal{P}_\kappa^r} \geq a_y - \frac{v_y}{v_x}a_x \tag{5.9a}$$

$$\frac{\kappa_{\min}}{v_x}\sqrt[3]{v_x^2 + v_y^2} \equiv \underline{\mathcal{P}_\kappa^r} \leq a_y - \frac{v_y}{v_x}a_x \tag{5.9b}$$

and use these polynomials in inequality constraints bounding a_x and a_y as (5.9) indicates. We approximate the term $\frac{v_y}{v_x}$ in the MIQP formulation by the mean orientation within the respective region, using the region boundaries, see (5.2). The larger the regions (the fewer number of regions), the higher the error will be. We then solve two unconstrained linear least-square problems by minimizing the error to (5.9a), yielding the linear polynomial $\overline{\mathcal{P}_\kappa^r}$, and (5.9b), yielding $\underline{\mathcal{P}_\kappa^r}$.

5.3 Mixed-Integer Formulation of the Planning Problem

This section introduces the optimization problem formulation as a quadratic minimization problem subject to a set of linear equality and inequality constraints.

5.3.1 Notation

We model the planning problem as MIQP without restricting the validity scope of the model. The vehicle motion model is formulated as discrete linear constraints. Using binary variables, we ensure collision-freeness and a correct non-holonomic motion of the vehicle. A quadratic objective function keeps the solution close to the reference.

Subsequently, we use the nomenclature for the decision variables introduced in Table 5.2 and Table 5.3 states the parameters of the model. The subscript $\square_{\text{ref}}$ denotes the respective reference. We optimize a discrete time range from k_2 to k_N with Δt increment. By $\mathcal{K}$, we denote the time interval $[k_1, \ldots, k_N]$. All decision variables are initialized with the current state of the vehicle

Table 5.3: Time- and region-independent parameters used throughout this work

Parameter	Description
q_p	Objective gain position
q_v	Objective gain velocity
q_a	Objective gain acceleration
q_u	Objective gain jerk
q_ξ	Objective gain agent slack
q_o	Objective gain obstacle slack
N	Number of states
Δt	Time increment
$\mathcal{K}$	Time interval: $[k_1, \dots, k_N]$
l	Vehicle wheel base
M	Big-M constant
r	collision circle radius
$\underline{V}$	Minimum speed to change a region
R	Number of regions
$\mathcal{R}$	Set of regions
$\underline{v}$, $\overline{v}$	Region, time, and direction independent lower/upper velocity limit
$\underline{a}$, $\overline{a}$	Region, time, and direction independent lower/upper acceleration limit
$\underline{u}$, $\overline{u}$	Region, time, and direction independent lower/upper jerk limit
Λ	Set of convex, polygonal environments
λ	Set of lines representing a convex polygonal sub-environment
$\mathcal{O}$	Set of obstacles
D	Maximum slack safety distance

at k_1. We bound the speed by the minimal and maximal values $\underline{v}$ and $\overline{v}$ from the fitting, as our approximations are only valid there. The following constraints are applied, if not indicated differently, to each agent individually. We omit the agent-dependency for each variable when not leading to ambiguousness.

5.3.2 Formulating the Vehicle Model as Constraints

The vehicle dynamics are defined by:

$$p_x(k_{i+1}) = p_x(k_i) + \Delta t v_x(k_i) + \frac{\Delta t^2 a_x(k_i)}{2} + \frac{\Delta t^3 u_x(k_i)}{6} \tag{5.10a}$$

$$v_x(k_{i+1}) = v_x(k_i) + \Delta t a_x(k_i) + \frac{\Delta t^2 u_x(k_i)}{2} \tag{5.10b}$$

$$a_x(k_{i+1}) = a_x(k_i) + \Delta t u_x(k_i) \tag{5.10c}$$

$$p_y(k_{i+1}) = p_y(k_i) + \Delta t v_y(k_i) + \frac{\Delta t^2 a_y(k_i)}{2} + \frac{\Delta t^3 u_y(k_i)}{6} \tag{5.10d}$$

$$v_y(k_{i+1}) = v_y(k_i) + \Delta t a_y(k_i) + \frac{\Delta t^2 u_y(k_i)}{2} \tag{5.10e}$$

$$a_y(k_{i+1}) = a_y(k_i) + \Delta t u_y(k_i) \tag{5.10f}$$

$$\forall k_i \in [k_1, \dots, k_{N-1}]$$

With this linear model correct non-holonomic motion as well as correct acceleration and steering angle limits are only valid around a small reference orientation. We overcome this property by introducing validity *regions*, see Section 5.1.1. The set of regions covering the full orientation of 360 degrees is denoted by $\mathcal{R}$. By the superscript $\square^r$, we denote region-dependent parameters in the following. All region-dependent parameters are summarized in Table 5.1.

We set the binary decision variable $\rho(k, r)$ defining in which region r the vehicle is in at time k according to the two lines defining the region, cf. (5.2). We force the solution to lie in exactly one region.

$$\sum_{r \in \mathcal{R}} \rho(k, r) = 1 \ \forall k \in \mathcal{K} \tag{5.11}$$

Since with an increasing number of regions, the evaluation time of the model increases as well, we a-priori restrict the optimization to only use a set of *allowed* regions and pre-compute a parameter ϱ^r accordingly. Non-allowed regions are those that cannot physically be reached within the planning horizon. To implement logical constraints, we use the technique of introducing a big constant M to switch inequalities. The intuitive explanation of the now often used term $M(1-\rho(k,r))$ is "this equation is active if and only if region r is active at time-step k". At low velocities, switching off the constraints enforcing the region is necessary as we forbid changing the region at these low velocities. This is achieved by the helper decision variable Ψ, that will be introduced in (5.17). Due to transformation errors from the absolute velocity to velocities separated in x and y direction and small changes in the input values could change the region the model can get infeasible if the regions are enforced. At low velocities, it is not that crucial that a non-holonomic motion occurs as acceleration and jerk are still bounded. The active region is set by the following set of constraints:

$$\alpha^r v_y(k) \geq \beta^r v_x(k) - M(1-\rho(k,r)) - M\Psi(k) \tag{5.12a}$$
$$\gamma^r v_y(k) \leq \delta^r v_x(k) + M(1-\rho(k,r)) + M\Psi(k) \tag{5.12b}$$
$$\forall k \in \mathcal{K}, \ \forall r \in \mathcal{R}, \ \text{if } \varrho^r(r) = 1$$

Regions marked as non-possible by ϱ^r may not be selected. With this implementation, the model formulation is generic for all scenarios.

$$\rho(k, r) = 0 \ \forall k \in \mathcal{K}, \ \forall r \in \mathcal{R}, \ \text{if } \varrho^r = 0 \tag{5.13}$$

We impose region-dependent limits on acceleration and jerk to make sure we always meet the correct absolute possible acceleration and jerk. As acceleration a_x, a_y and jerk u_x, u_y are defined in a global system, the limits are rotated with the regions. Note that the speeds are naturally bounded correctly by (5.12).

$$a_x(k) \leq \overline{a_x^r} + M(1-\rho(k,r)) \tag{5.14a}$$
$$a_x(k) \geq \underline{a_x^r} - M(1-\rho(k,r)) \tag{5.14b}$$
$$a_y(k) \leq \overline{a_y^r} + M(1-\rho(k,r)) \tag{5.14c}$$
$$a_y(k) \geq \underline{a_y^r} - M(1-\rho(k,r)) \tag{5.14d}$$
$$u_x(k) \leq \overline{u_x^r} + M(1-\rho(k,r)) \tag{5.14e}$$
$$u_x(k) \geq \underline{u_x^r} - M(1-\rho(k,r)) \tag{5.14f}$$
$$u_y(k) \leq \overline{u_y^r} + M(1-\rho(k,r)) \tag{5.14g}$$
$$u_y(k) \geq \underline{u_y^r} - M(1-\rho(k,r)) \tag{5.14h}$$
$$\forall k \in \mathcal{K}, \ \forall r \in \mathcal{R}, \ \text{if } \varrho^r = 1$$

5.3.3 Modeling the Non-Holonomy as Constraints

To ensure a correct non-holonomic movement of the vehicle, we limit the lateral acceleration using the maximal available curvature per region as motivated in Section 5.1.3. We fit lower and upper

linear approximation polynomials $\overline{\mathcal{P}^r_\kappa}$ and $\underline{\mathcal{P}^r_\kappa}$ dependent on v_x, v_y, and the region r. The following constraints model the inequalities bounding the curvature κ as stated in (5.8).

$$a_y(k) - \frac{\beta^r + \delta^r}{\alpha^r + \gamma^r} a_x(k) \leq \overline{\mathcal{P}^r_\kappa}(v_x(k), v_y(k)) + M(1 - \rho(k,r)) + M\Psi(k) \tag{5.15a}$$

$$a_y(k) - \frac{\beta^r + \delta^r}{\alpha^r + \gamma^r} a_x(k) \geq \underline{\mathcal{P}^r_\kappa}(v_x(k), v_y(k)) + M(1 - \rho(k,r)) + M\Psi(k) \tag{5.15b}$$

$$\forall k \in \mathcal{K},\ \forall r \in \mathcal{R},\ \text{if } \varrho^r = 1$$

At very low vehicle speeds, too tight curvature constraints limit the acceleration so that other accelerations than zero are not possible. In contrast, too loose curvature constraints will violate the non-holonomy. We therefore introduce a minimum speed limit. If both $|v_x|$ and $|v_y|$ are below that limit, regions changes are not allowed, which we indicate by setting the binary helper variable Ψ to true. We model this by a set of four helper variables Ψ^+_x, Ψ^-_x, Ψ^+_y, Ψ^-_y indicating if the velocity v_x or v_y exceeds or falls below the range defined by $\underline{V}$.

$$v_x(k_i) - \underline{V} \geq -M\Psi^+_x(k_i) \tag{5.16a}$$
$$v_x(k_i) - \underline{V} \leq M(1 - \Psi^+_x(k_i)) \tag{5.16b}$$
$$-v_x(k_i) - \underline{V} \geq M(1 - \Psi^-_x(k_i)) \tag{5.16c}$$
$$-v_x(k_i) - \underline{V} \leq -M\Psi^-_x(k_i) \tag{5.16d}$$
$$v_y(k_i) - \underline{V} \geq -M\Psi^+_y(k_i) \tag{5.16e}$$
$$v_y(k_i) - \underline{V} \leq M(1 - \Psi^+_y(k_i)) \tag{5.16f}$$
$$-v_y(k_i) - \underline{V} \geq M(1 - \Psi^-_y(k_i)) \tag{5.16g}$$
$$-v_y(k_i) - \underline{V} \leq -M\Psi^-_y(k_i) \tag{5.16h}$$
$$\forall k_i \in [k_2, \dots, k_N]$$

If all four helper variables evaluate to true, we set Ψ to true:

$$\Psi(k_i) \leq \Psi^+_x(k_i) \tag{5.17a}$$
$$\Psi(k_i) \leq \Psi^-_x(k_i) \tag{5.17b}$$
$$\Psi(k_i) \leq \Psi^+_y(k_i) \tag{5.17c}$$
$$\Psi(k_i) \leq \Psi^-_y(k_i) \tag{5.17d}$$
$$\Psi(k_i) \geq \Psi^+_x(k_i) + \Psi^-_x(k_i) + \Psi^+_y(k_i) + \Psi^-_y(k_i) - 3 \tag{5.17e}$$
$$\forall k_i \in [k_2, \dots, k_N]$$

If both v_x and v_y are within the $\underline{V}$ range, the combined decision variable Ψ is set true. In this case we set the current region equal to the region in the last optimization step assuming that region changes are not necessary at low velocities.

$$\rho(k_i, r) - \rho(k_{i-1}, r) \leq 1 - \Psi(k_i) \tag{5.18a}$$
$$\rho(k_i, r) - \rho(k_{i-1}, r) \geq -1 + \Psi(k_i) \tag{5.18b}$$
$$\forall k_i \in [k_2, \dots, k_N],\ \forall r \in \mathcal{R}$$

5.3.4 Approximating the Front Axle Position as Constraints

In Section 5.1.2 we introduced the concept of how to approximate lower and upper bounds for the position of the front axle of the vehicle. Using this idea, we formulate constraints performing an

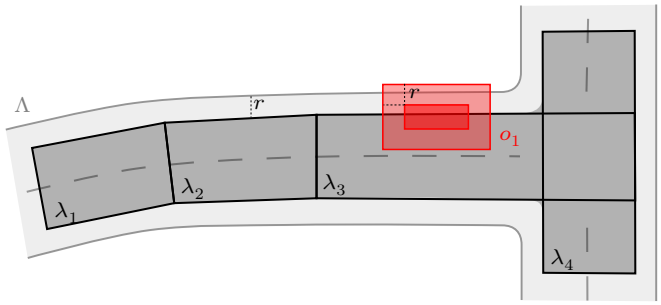


Figure 5.5: Schematic sketch showing how the environment polygon Λ is shrinked by the collision circle radius r and split into several convex polygons $\lambda_\square$ (gray). Obstacles $o_\square$ (red) are inflated with r. (graphic from [EKK20], ©2020 IEEE)

(over-)approximative collision check for the front axle instead of computing the intersection or distance of the actual vehicle shape with obstacles or the environment. We calculate the lower bounds $\underline{f_x}$, $\underline{f_y}$ and upper bounds $\overline{f_x}$, $\overline{f_y}$ of the true, unknown, front position (f_x, f_y) with respect to the current region r where l denotes the wheelbase of the vehicle.

$$M(\rho(k,r)-1) \leq \overline{f_x}(k) - p_x(k) - l\overline{\mathcal{P}^r_{\cos}}(v_x(k), v_y(k)) \tag{5.19a}$$
$$M(1-\rho(k,r)) \geq \overline{f_x}(k) - p_x(k) - l\overline{\mathcal{P}^r_{\cos}}(v_x(k), v_y(k)) \tag{5.19b}$$
$$-M(1-\rho(k,r)) \leq \underline{f_x}(k) - p_x(k) - l\underline{\mathcal{P}^r_{\cos}}(v_x(k), v_y(k)) \tag{5.19c}$$
$$M(1-\rho(k,r)) \geq \underline{f_x}(k) - p_x(k) - l\underline{\mathcal{P}^r_{\cos}}(v_x(k), v_y(k)) \tag{5.19d}$$
$$M(\rho(k,r)-1) \leq \overline{f_y}(k) - p_y(k) - l\overline{\mathcal{P}^r_{\sin}}(v_x(k), v_y(k)) \tag{5.19e}$$
$$M(1-\rho(k,r)) \geq \overline{f_y}(k) - p_y(k) - l\overline{\mathcal{P}^r_{\sin}}(v_x(k), v_y(k)) \tag{5.19f}$$
$$-M(1-\rho(k,r)) \leq \underline{f_y}(k) - p_y(k) - l\underline{\mathcal{P}^r_{\sin}}(v_x(k), v_y(k)) \tag{5.19g}$$
$$M(1-\rho(k,r)) \geq \underline{f_y}(k) - p_y(k) - l\underline{\mathcal{P}^r_{\sin}}(v_x(k), v_y(k)) \tag{5.19h}$$
$$\forall k_i \in \mathcal{K},\ \forall r \in \mathcal{R},\ \text{if } \varrho^r(r) = 1$$

5.3.5 Constraints Limiting the Model to Stay on the Road

This section introduces how we enforce the vehicle to stay within an environment modeled as an arbitrary, potentially non-convex closed polygon Λ [FB11]. The environment is deflated with the radius of the collision circles, see Figure 5.5. Non-convex environment polygons are split into several convex sub-polygons λ. We enforce the vehicle to be in at least one of these convex sub-polygons. These sub-polygons are represented by a set of line segments, denoted by l, between two points a^l and b^l. With this strategy, the polygon-to-polygon collision check narrows down to a point-to-polygon check.

We enforce the rear axle position (p_x, p_y) and the lower and upper bounds of the front axle position, namely the four points $(\underline{f_x}, \underline{f_y})$, $(\underline{f_x}, \overline{f_y})$, $(\overline{f_x}, \underline{f_y})$, and $(\overline{f_x}, \overline{f_y})$, to be within the environment polygon. Each set of constraints is formulated in a similar manner. The decision variable e models

that all five points do not collide with the environment sub-polygon λ at time k. By $l_\square$, we here denote $b^l_\square - a^l_\square$ for x and y respectively.

$$l_x(p_y(k) - a^l_y) - l_y(p_x(k) - a^l_x) \leq -Me(k,\lambda) \tag{5.20a}$$
$$l_x(\underline{f_y}(k) - a^l_y) - l_y(\underline{f_x}(k) - a^l_x) \leq -Me(k,\lambda) \tag{5.20b}$$
$$l_x(\underline{f_y}(k) - a^l_y) - l_y(\overline{f_x}(k) - a^l_x) \leq -Me(k,\lambda) \tag{5.20c}$$
$$l_x(\overline{f_y}(k) - a^l_y) - l_y(\underline{f_x}(k) - a^l_x) \leq -Me(k,\lambda) \tag{5.20d}$$
$$l_x(\overline{f_y}(k) - a^l_y) - l_y(\overline{f_x}(k) - a^l_x) \leq -Me(k,\lambda) \tag{5.20e}$$
$$\forall k \in \mathcal{K},\ \forall l \in \lambda,\ \forall \lambda \in \Lambda$$

To ensure that all vehicle points are at least within one of the environment polygons, we set

$$\sum_{\lambda \in \Lambda} e(k,\lambda) \leq |\Lambda| - 1\ \forall k \in \mathcal{K}, \tag{5.21}$$

where $|\Lambda|$ denotes the number of convex sub-environments.

5.3.6 Formulating Obstacle Collision Avoidance as Constraints

We enforce the vehicle to not collide with an arbitrary number of static or dynamic convex obstacle polygons $\mathcal{O}$. Non-convex obstacle shapes as described in Section 5.3.5 split into several convex ones. The obstacles are inflated with the radius of the collision circles, cf. Figure 5.5. Note that with the following formulation as well static obstacles as dynamically moving obstacles are avoided.

l again denotes one line segment of one obstacle o within the set of obstacles $\mathcal{O}$ with startpoint a^l and endpoint b^l. We enforce the vehicle rear axle position (p_x, p_y) and the four permutations of the front axle bounds to be collision-free. In contrast to the environment, which we assume as constant over time, the obstacle polygons may vary their position and shape over time, but preserve the polygon topology. By decision variables $o_\square$, we indicate whether none of the five points collide with the obstacle o. (5.22) states the inequalities for the point (p_x, p_y) constraining o_p and respective decision variables for the front axis lower and upper bounds. The four points representing the collision shape approximation of the front axle (f_x, f_y) are each taken into account by four more sets of similar decision variables $o_\square$ and sets of inequalities.

$$l_x(p_y(k) - a^l_y) - l_y(p_x(k) - a^l_x) \leq Mo_p(k,l) \tag{5.22a}$$
$$l_x(\underline{f_y}(k) - a^l_y) - l_y(\underline{f_x}(k) - a^l_x) \leq Mo_{\underline{f}\underline{f}}(k,l) \tag{5.22b}$$
$$l_x(\underline{f_y}(k) - a^l_y) - l_y(\overline{f_x}(k) - a^l_x) \leq Mo_{\underline{f}\overline{f}}(k,l) \tag{5.22c}$$
$$l_x(\overline{f_y}(k) - a^l_y) - l_y(\underline{f_x}(k) - a^l_x) \leq Mo_{\overline{f}\underline{f}}(k,l) \tag{5.22d}$$
$$l_x(\overline{f_y}(k) - a^l_y) - l_y(\overline{f_x}(k) - a^l_x) \leq Mo_{\overline{f}\overline{f}}(k,l) \tag{5.22e}$$
$$\forall k \in \mathcal{K},\ \forall l \in o,\ \forall o \in \mathcal{O}$$

Denoting the number of line segments in a sub-polygon by $|o|$, we enforce each of the five points to not lie within an obstacle.

$$\sum_{l \in o} o_p(k,l) \leq |o| - 1 \tag{5.23a}$$
$$\sum_{l \in o} o_{\underline{f}\underline{f}}(k,l) \leq |o| - 1 \tag{5.23b}$$
$$\sum_{l \in o} o_{\underline{f}\overline{f}}(k,l) \leq |o| - 1 \tag{5.23c}$$
$$\sum_{l \in o} o_{\overline{f}\underline{f}}(k,l) \leq |o| - 1 \tag{5.23d}$$
$$\sum_{l \in o} o_{\overline{f}\overline{f}}(k,l) \leq |o| - 1 \tag{5.23e}$$
$$\forall k \in \mathcal{K},\ \forall o \in \mathcal{O}$$

With the object collision avoidance constraints as formulated in (5.23) the exact shape of an obstacle is used for collision avoidance. If the object is a dynamically moving obstacle, such as a fellow car, the exact position and size is only know with some uncertainty due to the errors introduced from the perception and prediction components. Also, coping with human traffic participants, the predicted trajectory will only be true to a certain extent. Wrong perception and prediction can make the optimization problem infeasible. To avoid such situations, we soften the set of constraints (5.23) for dynamic obstacles by introducing slack variables $\xi_o \in \{0, 1\}$ that also will influence the objective function. These slack variables disable the respective inequality and allow the optimizer to violate the respective constraint. We modify the constraint set (5.23) for dynamic obstacles to

$$\sum_{l \in o} o_p(k,l) - \xi_o^1 \leq |o| - 1 \tag{5.24a}$$
$$\sum_{l \in o} o_{\underline{f}\underline{f}}(k,l) - \xi_o^2 \leq |o| - 1 \tag{5.24b}$$
$$\sum_{l \in o} o_{\underline{f}\overline{f}}(k,l) - \xi_o^3 \leq |o| - 1 \tag{5.24c}$$
$$\sum_{l \in o} o_{\overline{f}\underline{f}}(k,l) - \xi_o^4 \leq |o| - 1 \tag{5.24d}$$
$$\sum_{l \in o} o_{\overline{f}\overline{f}}(k,l) - \xi_o^5 \leq |o| - 1 \tag{5.24e}$$
$$\forall k \in \mathcal{K},\ \forall o \in \mathcal{O}$$

This formulation in practice has the consequence, that all dynamic obstacles are duplicated. A hard constraint avoids collision with the estimated position of the obstacles and a soft constraint formulation makes sure a sufficient safety distance is kept to robustly account for perception and prediction errors.

5.3.7 Multi-Agent Collision Constraints

We approximate the shape of each vehicle to formulate an agent-to-agent collision constraint. In our linear model, the actual orientation of the vehicle is not available, but with the region-based formulation, we can compute a lower- and upper-bounding rectangle for the center of the front axle.

In the following, the superscript $\square^i$ refers to the respective variable of the agent i. For collision avoidance, we approximate the vehicle shape by circles with radius R^i, one around the rear axle

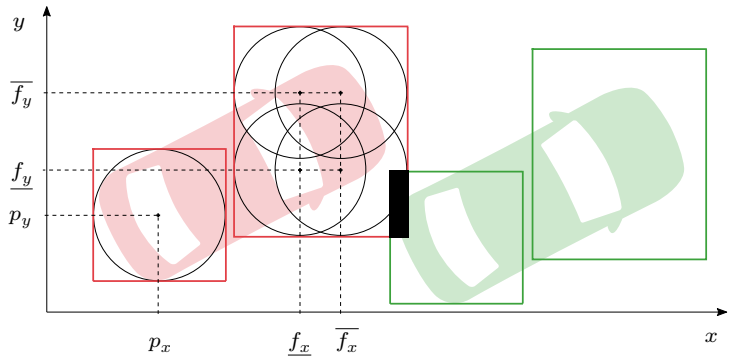


Figure 5.6: Approximation of the vehicle shape to formulate the agent-to-agent collision check based on the rectangles of the respective agents. The black area indicates a collision. (graphic from [KEK20], ©2020 IEEE)

center, and four for the front axle approximation. To avoid agent-to-agent collisions, we choose to over-approximate these circles with axis-aligned squares again, as sketched in Figure 5.6. A better approximation of the circles, for example, with two rectangles, yields more constraints and binary variables, lowering the runtime without providing a huge benefit. We formulate four sets of constraints, the first prevents collisions between the rear parts of two agents, the second prevents collisions between the rear part of the first and the front part of the second agent, the third vice versa, and the fourth prevents collisions between the front parts of both agents for each pair of agents.

The rear part-to-rear part collision constraint of two agents A^i and A^j is based on the following logical formula. We define the sum of both radii $R^{i+j} := R^i + R^j$. A collision occurs at one timestep k if and only if

$$\begin{aligned} &p_x^i(k) \geq p_x^j(k) - R^{i+j} \wedge p_x^i(k) \leq p_x^j(k) + R^{i+j} \\ \wedge\ &p_y^i(k) \geq p_y^j(k) - R^{i+j} \wedge p_y^i(k) \leq p_y^j(k) + R^{i+j}. \end{aligned} \tag{5.25}$$

Intuitively, (5.25) states that a collision occurs if both the absolute distance in x-direction $|p_x^i(k) - p_x^j(k)|$ and y-direction $|p_y^i(k) - p_y^j(k)|$ is smaller R^{i+j}. Logical negation yields that two agents do not collide at timestep k if and only if

$$\begin{aligned} &p_x^i(k) \leq p_x^j(k) - R^{i+j} \vee p_x^i(k) \geq p_x^j(k) + R^{i+j} \\ \vee &p_y^i(k) \leq p_y^j(k) - R^{i+j} \vee p_y^i(k) \geq p_y^j(k) + R^{i+j}. \end{aligned} \tag{5.26}$$

We formulate this as linear constraints using a set of four decision variables $\alpha_{\square}^{ij}$, one for each inequality and an appropriately chosen big constant M.

$$p_x^i(k) \leq p_x^j(k) - R^{i+j} + M\alpha_1^{ij}(k) \tag{5.27a}$$
$$p_x^i(k) \geq p_x^j(k) + R^{i+j} - M\alpha_2^{ij}(k) \tag{5.27b}$$
$$p_y^i(k) \leq p_y^j(k) - R^{i+j} + M\alpha_3^{ij}(k) \tag{5.27c}$$
$$p_y^i(k) \geq p_y^j(k) + R^{i+j} - M\alpha_4^{ij}(k) \tag{5.27d}$$
$$3 \geq \sum_{a=1}^{4} \alpha_a^{ij}(k) \ \forall\ k \in \mathcal{K}. \tag{5.27e}$$

(5.27e) represents the logical formulas (5.26) by coupling the four constraints (5.27a) - (5.27d) and makes sure no more than three are active, and hence no rear part-to-rear part collision occurs.

As we aim to cope with agents that are not controlled by our algorithm (e.g., human-driven vehicles), the computed motion will not exactly match the reality. This yields prediction errors

for the uncontrolled agents, which then can lead to infeasible optimization problems or imminent collisions. To account for these prediction errors, we introduce an additional safety distance for the ego agent to all uncontrolled agents as a *soft constraint*. Inspired by the formulation in [GGW17], we introduce *slack variables* ξ to the agent-to-agent collision constraints. With the desired safety distance of both agents $D(k)$ and a set of slack variables $\xi_x(k), \xi_y(k) \in [0,\ D(k)]$, we modify (5.27) to

$$p_x^i(k) \leq p_x^j(k) - R^{i+j} - D(k) + \xi_x^{ij}(k) + M\alpha_1^{ij}(k) \tag{5.28a}$$
$$p_x^i(k) \geq p_x^j(k) + R^{i+j} + D(k) - \xi_x^{ij}(k) - M\alpha_2^{ij}(k) \tag{5.28b}$$
$$p_y^i(k) \leq p_y^j(k) - R^{i+j} - D(k) + \xi_y^{ij}(k) + M\alpha_3^{ij}(k) \tag{5.28c}$$
$$p_y^i(k) \geq p_y^j(k) + R^{i+j} + D(k) - \xi_y^{ij}(k) - M\alpha_4^{ij}(k) \tag{5.28d}$$
$$\xi_x(k) \leq D(k) \tag{5.28e}$$
$$\xi_y(k) \leq D(k) \tag{5.28f}$$
$$3 \geq \sum_{a=1}^{4} \alpha_a^{ij}(k)\ \forall\ k \in \mathcal{K}. \tag{5.28g}$$

The slack variables will be included in the cost function (see Section 5.3.8). The optimizer will then seek to keep the slack variables as small as possible. Consequently, in (5.28), the additional safety distance D will be as high as possible. With this concept, fatal prediction errors (immanent collisions) are mitigated. Note that adding a hard safety margin only leads to more conservative behavior and does not avoid infeasible optimization problems.

To prevent collisions between the rear part of agent A^i and the front part of agent A^j, we again formulate logical constraints that a collision occurs at k if and only if

$$\begin{aligned} p_x^i(k) \geq \underline{f_x}^j(k) - R^{i+j} \wedge p_x^i(k) \leq \overline{f_x}^j(k) + R^{i+j} \\ \wedge\ p_y^i(k) \geq \underline{f_y}^j(k) - R^{i+j} \wedge p_y^i(k) \leq \overline{f_x}^j(k) + R^{i+j}. \end{aligned} \tag{5.29}$$

Here we force the point $(p_x^i,\ p_y^i)$ to be outside the front axle approximation rectangle enlarged by the sum of the collision circle radii. The set of constraints is formulated as described for the rear part-to-rear part collision case. This can again be formulated as a set of five constraints with appropriate decision variables:

$$p_x^i(k) \leq \underline{f_x}^j(k) - R^{i+j} + M\alpha_1^{ij}(k) \tag{5.30a}$$
$$p_x^i(k) \geq \overline{f_x}^j(k) + R^{i+j} - M\alpha_2^{ij}(k) \tag{5.30b}$$
$$p_y^i(k) \leq \underline{f_y}^j(k) - R^{i+j} + M\alpha_3^{ij}(k) \tag{5.30c}$$
$$p_y^i(k) \geq \overline{f_y}^j(k) + R^{i+j} - M\alpha_4^{ij}(k) \tag{5.30d}$$
$$3 \geq \sum_{a=1}^{4} \alpha_a^{ij}\ \forall\ k \in \mathcal{K} \tag{5.30e}$$

Another analog set of constraints prevents collisions between the rear part of A^j and the front part of A^i. Both front-to-rear collision checks are not softened with slack variables as this only showed a minor effect on the performance with increased runtime.

We avoid collisions between the fronts of agents A^i and A^j applying the same strategy but forcing the center point of the front axle approximation rectangle of A^j to retain sufficient distance to the front axle approximation rectangle of A^i. Concretely, we define the sufficient distance as

R^{i+j} plus the size of the approximation rectangle of agent A^j. Hence, no front part-to-front part collision occurs at timestep k if and only if

$$\begin{aligned}
&\frac{1}{2}(\overline{f_x}^j(k) + \underline{f_x}^j(k)) \leq \underline{f_x}^i(k) - R^{i+j} - \frac{1}{2}(\overline{f_x}^j(k) - \underline{f_x}^j(k)) \vee \\
&\frac{1}{2}(\overline{f_x}^j(k) + \underline{f_x}^j(k)) \geq \overline{f_x}^i(k) + R^{i+j} + \frac{1}{2}(\overline{f_x}^j(k) - \underline{f_x}^j(k)) \vee \\
&\frac{1}{2}(\overline{f_y}^j(k) + \underline{f_y}^j(k)) \leq \underline{f_y}^i(k) - R^{i+j} - \frac{1}{2}(\overline{f_y}^j(k) - \underline{f_y}^j(k)) \vee \\
&\frac{1}{2}(\overline{f_y}^j(k) + \underline{f_y}^j(k)) \geq \overline{f_y}^i(k) + R^{i+j} + \frac{1}{2}(\overline{f_y}^j(k) - \underline{f_y}^j(k)).
\end{aligned} \tag{5.31}$$

The set of constraints then calculates to

$$0 \leq \underline{f_x}^i(k) - \overline{f_x}^j(k) - R^{i+j} + \xi_x^{ij}(k) + M\alpha_1^{ij}(k) \tag{5.32a}$$

$$0 \geq \overline{f_x}^i(k) - \underline{f_x}^j(k) + R^{i+j} - \xi_x^{ij}(k) - M\alpha_2^{ij}(k) \tag{5.32b}$$

$$0 \leq \underline{f_y}^i(k) - \overline{f_y}^j(k) - R^{i+j} + \xi_y^{ij}(k) + M\alpha_1^{ij}(k) \tag{5.32c}$$

$$0 \geq \overline{f_y}^i(k) - \underline{f_y}^j(k) + R^{i+j} - \xi_y^{ij}(k) - M\alpha_2^{ij}(k) \tag{5.32d}$$

$$3 \geq \sum_{a=1}^{4} \alpha_a^{ij}(k) \;\forall\; k \in \mathcal{K} \tag{5.32e}$$

including an appropriate set of slack variables ξ and decision variables α.

As an alternative, we also formulated tighter front part-to-front part collision constraints by excluding the interval overlap of $[\underline{f_x}^i - R^i, \overline{f_x}^i + R^i]$ with $[\underline{f_x}^j - R^j, \overline{f_x}^j + R^j]$ and in y-direction vice versa. This formulation is slower as it needs more binary decision variables and does not provide huge benefits. In dense and coupled scenarios, (5.31) can lead to harsh braking or acceleration in front of narrow but still drivable passages due to the over-conservative approximation.

5.3.8 Joint Cost Function

As the objective of the optimization problem, we chose to minimize a weighted sum of individual cost functions per agent. This approach if referred to centralized planning in a model predictive control setting.

Formulating Reference Tracking as Objective Function We formulate the individual cost function term of agent A^i as a weighted sum of position and velocity distance to the reference. For acceleration and jerk, we aim to minimize the squared values to avoid changes. Suitable cost terms $q_\square$ balance the solution.

$$\begin{aligned}
J^i = \sum_{k \in \mathcal{K}} &(q_p^i(p_x^i(k) - p_{x,ref}^i(k))^2 + q_p^i(p_y^i(k) - p_{y,ref}^i(k))^2 \\
&+ q_v^i(v_x^i(k) - v_{x,ref}^i(k))^2 + q_v^i(v_y^i(k) - v_{y,ref}^i(k))^2 \\
&+ q_a^i a_x^i(k)^2 + q_a^i a_y^i(k)^2 + q_u^i u_x^i(k)^2 + q_u^i u_y^i(k)^2)
\end{aligned} \tag{5.33}$$

To formulate a MIQP problem, the objective function has to be a sum of squared or linear terms. Due to the separation of x- and y-directions, absolute terms, such as a reference path, cannot be included directly in the cost function. As we still consider it desirable to track a reference speed along a reference path, we compute the trajectory reference, a sequence of x and y coordinates along the discrete time k, from path and speed reference values in a pre-processing step. Also, we cannot use the absolute velocity (or acceleration) in the objective, since with $|v| = \sqrt{(v_x^2 + v_y^2)}$,

the term $(|v|^2 - |v_{ref}|^2)^2$ is not quadratic. Similarly, costs on the angular velocity would result in non-quadratic terms. Furthermore, as we do not calculate a distance to objects in the model, we cannot use these distance terms in the cost function. If the problem is formulated in a fully nonlinear way such terms are possible in the objective function [Zie+14a]. Note that with this formulation of the objective function we track a reference trajectory and not a reference path.

Combination Into a Joint Cost Function To leverage the interests of the agents, we introduce *scaling factors* $\lambda \in [0, 1]$ in the overall cost function. With these, we can push the optimization problem to generate egoistic, symmetric, or altruistic solutions. For each agent A^i, we define a scaling factor λ^i with the following properties. All scaling factors shall sum up to one

$$\sum_{i \in \mathcal{A}} \lambda^i = 1. \tag{5.34}$$

The scaling factor of the ego agent λ^{ego} is chosen freely within the interval $[0, 1]$. We then set

$$\lambda^i = \frac{1 - \lambda^{\text{ego}}}{|\mathcal{A}| - 1} \ \forall \ i \neq \text{ego}. \tag{5.35}$$

The intuitive explanation of (5.35) is that high values for λ^{ego} will lead to egoistic ego behavior, all values equal a symmetric solution, and for small values of λ^{ego} the ego agent will not enforce its own goals, only trying to fulfill the constraints.

The overall cost function is then defined as

$$J = \sum_{i \in \mathcal{A}} \lambda^i J^i + q_\xi \sum_{k \in \mathcal{K}} \sum_{i \in \mathcal{A},\ j \in \mathcal{A}i} \left(\xi_x^{ij}(k) + \xi_y^{ij}(k)\right)^2 + q_o \sum_{o \in \mathcal{O}} \sum_{i=1}^{4} \left(\xi_o^i\right)^2 \tag{5.36}$$

with the second term penalizing high values of the slack variables and a weighting factor q_ξ.

5.3.9 Optimization Problem

Collecting all constraints from above, the final optimization problem can be written as

$$\begin{aligned} &\text{minimize } (5.36) \\ &\text{subject to } (5.10), (5.11), (5.12), (5.13), (5.14), (5.15), (5.16), (5.17), (5.19), \\ &\qquad (5.18), (5.20), (5.21), (5.22), (5.23), (5.24), (5.28), (5.30), (5.32). \end{aligned} \tag{5.37}$$

The formulation (5.37) is a standard MIQP model that can be solved with an off-the-shelf solver. For a faster optimization solution, we additionally bound acceleration a_x, a_y and jerk u_x, u_y with constant values $\underline{a}$, $\overline{a}$ and $\underline{u}$, $\overline{u}$ respectively. We chose these values as minima and maxima of the region-dependent limits $\underline{a_x^r}$, $\underline{a_y^r}$, $\overline{a_x^r}$, $\overline{a_y^r}$ and respectively for the jerk. For the velocities v_x, v_y we bound the the values by the upper and lower bound of the fit $\underline{v}$ and $\overline{v}$ increased by a small ϵ to avoid numerical difficulties at the boundaries.

We execute the algorithm in a receding horizon fashion and, therefore, only execute the first steps of the optimized trajectory. As initial conditions, we set the states of each agent and the current region of each agent. Adding the latter helps to avoid infeasible problems on the region boundaries. Selecting this region correctly is crucial, as otherwise the constraints are violated at the first step.

5.4 Validation of the Model

As the model formulation for MINIVAN introduces linearizations and approximations, we in this section show that the model generates valid trajectories.

5.4.1 Reference Implementations

MINIVAN uses a linearized vehicle model and constraints assuring a correct non-holonomic motion. To assert that the planned motion is indeed correct, we implemented two nonlinear optimization problems of a vehicle model to compare the MINIVAN solution against. We use Sequential Quadratic Programming (SQP) to solve the problem. The model only computes the agent's motion over a given horizon without checking for collision with the environment, obstacles, or other agents. We perform a post-optimization collision check afterwards to filter out invalid solutions. The model formulation contains no binary or integer variables. It was also not tuned for performance, therefore we omit evaluations on the computation time but focus on the quality of the solution. The objective function and the vehicle model are implemented following the the solution of MINIVAN as much as possible.

The comparison of the MIQP solution and the SQP solution shall answer the following questions:

- Is the MIQP solution correct in terms of holonomy so that a nonlinear bicycle model produces a comparable result? For this, we use the solution trajectory from the MIQP model as reference for the SQP optimization and monitor the deviations.
- Is the linearized curvature constraint of the MIQP model correctly overapproximating the true curvature? For this, we optimize a triple integrator model with nonlinear curvature constraint using the SQP optimizer.

Third Order Integrator With Nonlinear Coupling

We use the same vehicle dynamics constraints as the MIQP model, namely (5.10). To ensure that we compute a non-holonomic motion, we couple longitudinal and lateral values using the side slip angle

$$\kappa_{\min} \leq \frac{v_x(k)a_y(k) - a_x(k)v_y(k)}{(v_x(k)^2 + v_y(k)^2)^{3/2}} \leq \kappa_{\max} \ \forall \ k \in \mathcal{K} \tag{5.38}$$

We further bound the absolute acceleration and jerk by

$$a_{\min} \leq \sqrt{a_x^2(k) + a_y(k)^2} \leq a_{\max} \ \forall \ k \in \mathcal{K} \tag{5.39a}$$

$$u_x(k)^2 + u_y(k)^2 \leq u_{\max}^2 \ \forall \ k \in \mathcal{K} \tag{5.39b}$$

As an objective function, we chose the same as for an MIQP model with only one agent (5.33). The resulting nonlinear optimization problem can then be written as

$$\begin{aligned} &\text{minimize } (5.33) \\ &\text{subject to } (5.10), (5.38), (5.39) \end{aligned} \tag{5.40}$$

Bicycle Model

We implement a bicycle model integrated with forward Euler integration with steering angle δ and acceleration a as input and the coordinates p_x, p_y, orientation θ and velocity v as state.

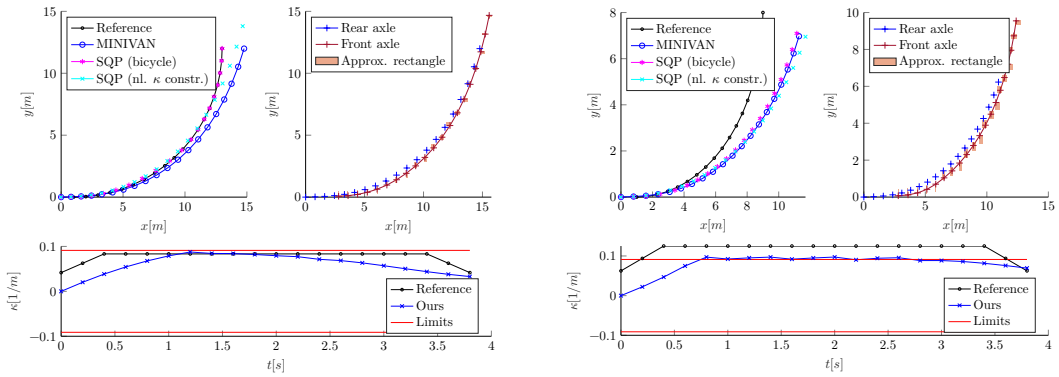
$$p_x(k_{i+1}) = p_x(k_i) + \Delta t v(k_i)\cos(\theta(k_i)) \tag{5.41a}$$

$$p_y(k_{i+1}) = p_y(k_i) + \Delta t v(k_i)\sin(\theta(k_i)) \tag{5.41b}$$

$$\theta(k_{i+1}) = \theta(k_i) + \Delta t v(k_i)\tan(\delta(k_i)/l) \tag{5.41c}$$

$$v(k_{i+1}) = v(k_i) + \Delta t a(k_i) \tag{5.41d}$$

$$\forall k_i \in [k_1, \dots, k_{N-1}]$$



(a) Feasible reference

(b) Infeasible reference

Figure 5.7: MINIVAN stays within the curvature bounds (lower plot) and shows valid trajectory tracking behavior as the reference implementations using a SQP optimizer does (upper left plot). The true front axle position is always within the lower/upper bound approximation rectangles of the front axle (upper right plot). All three optimization solutions cannot track the reference despite operating at maximum curvature for an infeasible reference. (graphic from [EKK20], ©2020 IEEE)

We further limit the model inputs by

$$a_{\min} \leq a(k_i) \leq a_{\max} \tag{5.42a}$$
$$\delta_{\min} \leq \delta(k_i) \leq \delta_{\max} \tag{5.42b}$$
$$\forall k_i \in [k_1, \dots, k_N].$$

As a cost function we implement a pure reference path tracking by

$$J = \sum_{k \in \mathcal{K}} \left((p_x(k) - p_{x,ref}(k))^2 + (p_y(k) - p_{y,ref}(k))^2 \right). \tag{5.43}$$

The resulting nonlinear optimization problem can then be written as

$$\begin{aligned} &\text{minimize } (5.43) \\ &\text{subject to } (5.41), (5.42). \end{aligned} \tag{5.44}$$

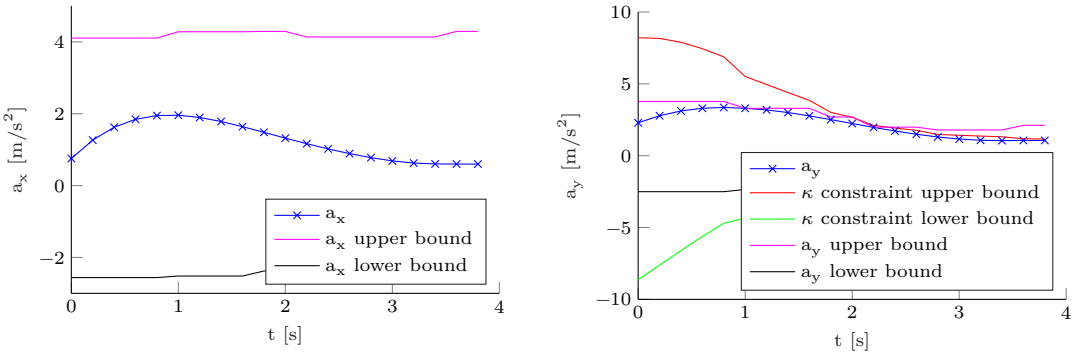
5.4.2 Preserving the Non-holonomy via Region-Based Constraints

This section will discuss that the region-based linearization approach produces dynamically feasible trajectories without introducing over-conservatism. As exemplary driving scenes, we pick a simple ideal circle and the racetrack from the competitive autonomous racing scenario, which will be further discussed in Section 5.7.5.

Obeying Curvature Limits

To show the effectiveness of the constraints guaranteeing non-holonomic motions, we optimize two different reference trajectories, both forming a circle to perform a 90 deg turn. While it is preferable to only generate reference trajectories with only valid curvatures, this example clearly shows that our model preserves the non-holonomy. The same property is also necessary for obstacle avoidance.

Figure 5.7a shows the results for a turning radius that the vehicle model is able to follow. Our optimization yields a trajectory that closely follows the reference. However, the second reference



(a) Acceleration in x direction

(b) Acceleration in y direction

Figure 5.8: Acceleration in x and y direction of one exemplary planning instance at the hairpin corner of the racetrack at x=150 m, y=0 m. In x-direction, the motion is not restricted, in y-direction the acceleration is in the first half restricted by the acceleration constraints and in the second half by the curvature constraints.

in Figure 5.7b models a turning radius that is too small (the curvature of the reference exceeding the limits). As desired, our model does not follow the reference and yields a trajectory that stays within the curvature bounds. As we fit the curvature approximation polygons in the mean of each region (see Section 5.2), we can slightly exceed the curvature bound at region boundaries. This can easily be mitigated using a safety margin.

Effect of Acceleration and Curvature Constraints

One advantage of our region-based concept is the validity scope of the model of 360 degrees. As correct non-holonomic motion is achieved using a curvature constraint, the (region-dependent) constraints on acceleration in the longitudinal and lateral direction can purely be parameterized with respect to the vehicle capabilities or comfort limitations. These absolute longitudinal and lateral limitations form boundaries in x and y directions that are rotated with the regions. This rotation ensures that the appropriate fraction of longitudinal/lateral limits are correctly translated into constraints in x and y direction. Correct non-holonomics are enforced by the constraints (5.15). Depending on the driving situation, one or the other set of constraints forms the tighter bound. In a challenging situation of passing a hairpin corner on a racetrack with high speeds, we can observe that the vehicle is in a state near the physical limits. One instance is exemplarily depicted in Figure 5.8. In x direction, the acceleration does not need to be constrained. In y direction, in the first 2 s the acceleration constraints form the tighter bound, and the vehicle is moving at its physical limit. In the timespan from 2 s to 4 s, the curvature constraints are limiting tighter and force the planner to generate a correct non-holonomic motion. Dropping the curvature constraints will, in such a situation, result in the generation of a motion that is still valid in terms of acceleration bounds but fails to guarantee correct non-holonomics.

Performance Degradation Without the Concept of Regions

Without the concept of regions, a correct holonomic motion of the vehicle is achieved by limiting the lateral velocity with respect to the longitudinal velocity, such as in [Qia+16; BL18]. In scenarios with high demands on correct vehicle kinematics, such as the racetrack scenario we discuss in Section 5.7.5, this constraint is too restrictive. With a parameterization where v_y is limited with respect to v_x ensuring correct non-holonomy and not switching to an appropriate region, the vehicle is significantly slower on the racetrack. It cannot track the ideal line with high velocity anymore but has to decelerate in the curves to maintain correct non-holonomic behavior and not

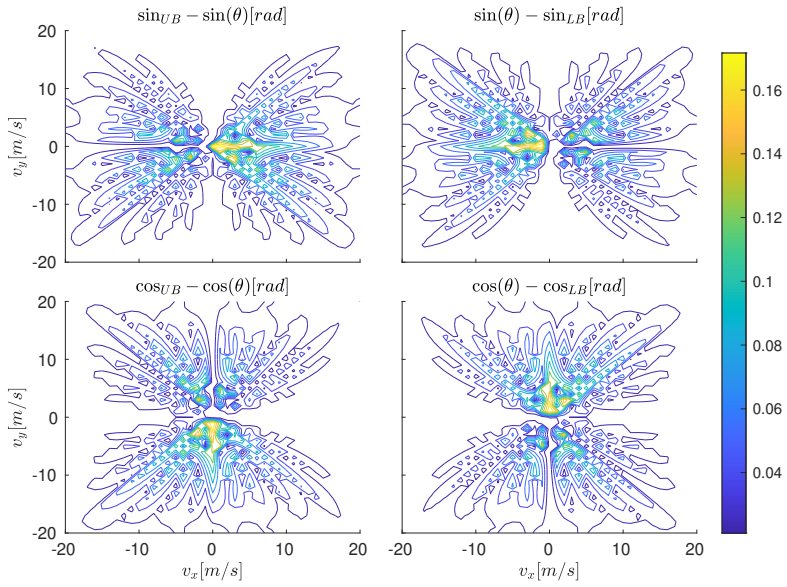


Figure 5.9: Errors of the piecewise linear fitting using 32 regions and $\overline{v} = 20\,\mathrm{m/s}$ of upper $\square_{UB}$ and lower bounds $\square_{LB}$ trigonometric functions. (graphic from [EKK20], ©2020 IEEE)

collide with the track boundaries. This degraded performance leads to a lap time of 105.7 s, which is significantly higher than MINIVAN.

5.4.3 Discussion of the Introduced Approximation Errors

The chosen linearizations and over-approximations introduce errors in the model, that we will discuss in the following.

Fitting Errors

The fitting method introduced in Section 5.2 introduces errors as the front axis position cannot be computed to be at an exact value but only to lie within a range of upper and lower bounds in x and y direction. An immediate consequence of this is that the algorithm reserves too much space within the drivable area and might fail to find a solution if space is too limited, even if the available area is sufficient.

Figure 5.9 shows the errors we obtain from the fitting in one exemplary setting. For the upper and lower bounds of the sine and cosine function the orientation error is always below 0.16 rad. Due to

$$\sin(v_x = 0, v_y \to \pm 0) = \pm\infty \tag{5.45a}$$
$$\cos(v_x \to \pm 0, v_y = 0) = \pm\infty, \tag{5.45b}$$

we get the highest error close to the origin. For higher velocities, the errors are significantly smaller due to a better approximation of the nonlinear function.

Table 5.4 shows the positional error of the front axle for a range of orientations in the first quadrant in comparison to the constant approximation (denoted by const.). We observe that the upper and lower bound always have different signs, which means that the actual axle position is always within the bounds. This shows the validity of the over-approximation of the front axle. As expected, the error becomes smaller with an increasing number of regions, as we are using smaller linear pieces to approximate the nonlinear functions. In general, the error of the velocity-dependent

approximation is smaller than for the constant approximation. Only for velocities $\lessapprox 0.1\,\mathrm{m/s}$, a constant approximation yields smaller errors, as motivated by (5.45). Also comparing the maximum velocity that was used to fit $\overline{v}$ we observe that the quality of the fit significantly improves, as the lower and upper bound can be chosen more tightly. This effect cannot only be observed for the highest velocity of the fit (which is expectable) but also for medium velocities from 1 m/s to 5 m/s. Note that due to the symmetry of the sine and cosine function, the errors are also rotated, see the 30 deg and 120 deg columns.

Convex Approximations of Obstacles

The introduced constraints for obstacle avoidance and the constraints enforcing the vehicle to stay within the road boundaries need all geometry polygons to be convex. Therefore non-convex shapes have to be split into several convex ones. While theoretically, an arbitrary number of (small) obstacle and environment polygons are possible, many small polygons can significantly increase the runtime. Therefore, we always use the convex hull of the obstacle shape for obstacles, which introduces additional conservatism.

Convex Approximations of the Environment

The convex approximation of the environment shape is explained in Section 6.3.3. This convexification can introduce a non-overapproximating error, as the drivable area can be increased, e.g., in a curve. To avoid too many small fragments of convex sub-polygons, the geometries are simplified by dropping points. This has the negative side effect that the overall drivable area increases virtually. With a suitable parameterization, a reasonable trade-off between the accuracy of the approximation and the number of sub-polygons can be found.

Agent Shape Approximation

Furthermore, the shape of agent models are approximated using a series of circles. This approximation introduces additional conservatism. For collision avoidance, the whole agent shape should be covered by circles. As a trade-off between conservatism, safety, and performance, we do not cover the middle part of the vehicle by circles. It is very unlikely that a collision occurs here, especially with a rather too over-conservative approximation of the front part of the vehicle. The approximation is also illustrated in Figure 5.3.

5.4.4 Limitations of the Model Formulation

The main drawback of the proposed MIP formulation is the handling of very slow velocities. At zero velocity, the model is invalid due to the needed transformation into velocities distributed into x and y direction using the arctan function and the computation of the orientation from v_x and v_y. Also, these transformations are not stable for low velocities, and the resulting trajectory shows non-holonomic properties. Experimental results show that a stable and valid performance of MINIVAN is achieved for velocities higher than 0.7 m/s. We also elaborate on this in Section 6.3 and show how we post-process the solution to obtain valid results. This effect leads to restrictions in multi-agent settings as no valid joint maneuvers can be planned where one agent has to come to a complete stop. Here the trajectory leading to this point is very likely to be invalid.

Also, due to the conservative approximation of the true shape of an agent, obstacle, and environment, we can ensure a collision-free motion. However, in very narrow scenarios, we could exclude the optimal solution or even all valid solutions. This effect can be mitigated by increasing the number of regions.

Table 5.4: Absolute positional errors $(f_x - [\underline{f_x}, \overline{f_x}])$ and $(f_y - [\underline{f_y}, \overline{f_y}])$ for the approximation of the front rear axle in meters. The maximum velocity used to fit is indicated by $\overline{v}$.

θ	v		16 regions $\overline{v} = 10\frac{m}{s}$	16 regions $\overline{v} = 20\frac{m}{s}$	32 regions $\overline{v} = 10\frac{m}{s}$	32 regions $\overline{v} = 20\frac{m}{s}$	64 regions $\overline{v} = 20\frac{m}{s}$	128 regions $\overline{v} = 20\frac{m}{s}$
0°	const.	x	0.21, 0.00	0.21, 0.00	0.05, 0.00	0.05, 0.00	0.01, 0.00	0.00, 0.00
		y	0.00, -1.07	0.00, -1.07	0.00, -0.55	0.00, -0.55	0.00, -0.27	0.00, -0.14
	$1\frac{m}{s}$	x	0.17, -0.01	0.20, -0.02	0.05, -0.00	0.06, -0.01	0.01, -0.01	0.01, -0.01
		y	0.00, -0.87	0.01, -0.93	0.00, -0.46	0.01, -0.50	0.01, -0.09	0.01, -0.01
	$5\frac{m}{s}$	x	0.10, -0.03	0.16, -0.03	0.03, -0.01	0.05, -0.02	0.01, -0.01	0.01, -0.01
		y	0.00, -0.48	0.01, -0.74	0.00, -0.25	0.01, -0.40	0.01, -0.07	0.01, -0.01
	$10\frac{m}{s}$	x	0.00, -0.07	0.11, -0.05	0.00, -0.02	0.04, -0.02	0.01, -0.01	0.01, -0.01
		y	0.00, -0.00	0.01, -0.49	0.00, -0.00	0.01, -0.27	0.01, -0.05	0.01, -0.01
	$20\frac{m}{s}$	x		0.01, -0.08		0.01, -0.03	0.01, -0.01	0.01, -0.01
		y		0.01, -0.01		0.01, -0.01	0.01, -0.01	0.01, -0.01
30°	const.	x	0.44, -0.16	0.44, -0.16	0.10, -0.16	0.10, -0.16	0.10, -0.04	0.02, -0.04
		y	0.33, -0.58	0.33, -0.58	0.33, -0.16	0.33, -0.16	0.08, -0.16	0.08, -0.04
	$1\frac{m}{s}$	x	0.41, -0.15	0.44, -0.17	0.09, -0.15	0.10, -0.18	0.11, -0.02	0.04, -0.03
		y	0.29, -0.52	0.31, -0.56	0.32, -0.13	0.36, -0.15	0.02, -0.16	0.04, -0.05
	$5\frac{m}{s}$	x	0.26, -0.10	0.37, -0.14	0.06, -0.10	0.09, -0.15	0.09, -0.02	0.03, -0.03
		y	0.26, -0.33	0.29, -0.46	0.21, -0.09	0.30, -0.13	0.03, -0.13	0.04, -0.05
	$10\frac{m}{s}$	x	0.07, -0.05	0.27, -0.12	0.03, -0.03	0.07, -0.11	0.07, -0.02	0.03, -0.03
		y	0.23, -0.08	0.28, -0.34	0.07, -0.03	0.22, -0.10	0.03, -0.10	0.03, -0.04
	$20\frac{m}{s}$	x		0.09, -0.06		0.04, -0.04	0.03, -0.03	0.02, -0.02
		y		0.24, -0.09		0.08, -0.04	0.04, -0.03	0.03, -0.02
60°	const.	x	0.33, -0.58	0.33, -0.58	0.33, -0.16	0.33, -0.16	0.08, -0.16	0.08, -0.04
		y	0.44, -0.16	0.44, -0.16	0.10, -0.16	0.10, -0.16	0.10, -0.04	0.02, -0.04
	$1\frac{m}{s}$	x	0.29, -0.52	0.31, -0.56	0.32, -0.13	0.36, -0.15	0.02, -0.16	0.04, -0.05
		y	0.41, -0.15	0.44, -0.17	0.09, -0.15	0.10, -0.18	0.11, -0.02	0.04, -0.03
	$5\frac{m}{s}$	x	0.26, -0.33	0.29, -0.46	0.21, -0.09	0.30, -0.13	0.03, -0.13	0.04, -0.05
		y	0.26, -0.10	0.37, -0.14	0.06, -0.10	0.09, -0.15	0.09, -0.02	0.03, -0.03
	$10\frac{m}{s}$	x	0.23, -0.08	0.28, -0.34	0.07, -0.03	0.22, -0.10	0.03, -0.10	0.03, -0.04
		y	0.07, -0.05	0.27, -0.12	0.03, -0.03	0.07, -0.11	0.07, -0.02	0.03, -0.03
	$20\frac{m}{s}$	x		0.24, -0.09		0.08, -0.04	0.04, -0.03	0.03, -0.02
		y		0.09, -0.06		0.04, -0.04	0.03, -0.03	0.02, -0.02
90°	const.	x	0.00, -1.07	0.00, -1.07	0.00, -0.55	0.00, -0.55	0.00, -0.27	0.00, -0.14
		y	0.21, 0.00	0.21, 0.00	0.05, 0.00	0.05, 0.00	0.01, 0.00	0.00, 0.00
	$1\frac{m}{s}$	x	0.00, -0.87	0.01, -0.93	0.00, -0.46	0.01, -0.51	0.01, -0.09	0.01, -0.01
		y	0.17, -0.00	0.20, -0.01	0.05, -0.00	0.06, -0.01	0.01, -0.01	0.01, -0.01
	$5\frac{m}{s}$	x	0.00, -0.48	0.01, -0.74	0.00, -0.26	0.01, -0.40	0.01, -0.08	0.01, -0.01
		y	0.10, -0.03	0.16, -0.03	0.03, -0.01	0.05, -0.02	0.01, -0.01	0.01, -0.01
	$10\frac{m}{s}$	x	0.00, -0.00	0.01, -0.50	0.00, -0.00	0.01, -0.27	0.01, -0.06	0.01, -0.01
		y	0.00, -0.07	0.11, -0.05	0.00, -0.02	0.04, -0.02	0.01, -0.01	0.01, -0.01
	$20\frac{m}{s}$	x		0.01, -0.01		0.01, -0.01	0.01, -0.01	0.01, -0.01
		y		0.01, -0.08		0.01, -0.03	0.01, -0.01	0.01, -0.01
120°	const.	x	0.58, -0.33	0.58, -0.33	0.16, -0.33	0.16, -0.33	0.16, -0.08	0.04, -0.08
		y	0.44, -0.16	0.44, -0.16	0.10, -0.16	0.10, -0.16	0.10, -0.04	0.02, -0.04
	$1\frac{m}{s}$	x	0.52, -0.29	0.56, -0.31	0.13, -0.32	0.15, -0.36	0.16, -0.02	0.05, -0.04
		y	0.41, -0.15	0.44, -0.17	0.09, -0.15	0.10, -0.18	0.11, -0.02	0.04, -0.03
	$5\frac{m}{s}$	x	0.33, -0.26	0.46, -0.29	0.09, -0.21	0.13, -0.30	0.13, -0.03	0.05, -0.04
		y	0.26, -0.10	0.37, -0.14	0.06, -0.10	0.09, -0.15	0.09, -0.02	0.03, -0.03
	$10\frac{m}{s}$	x	0.08, -0.23	0.34, -0.28	0.03, -0.07	0.10, -0.22	0.10, -0.03	0.04, -0.03
		y	0.07, -0.05	0.27, -0.12	0.03, -0.03	0.07, -0.11	0.07, -0.02	0.03, -0.03
	$20\frac{m}{s}$	x		0.09, -0.24		0.04, -0.08	0.03, -0.04	0.02, -0.03
		y		0.09, -0.06		0.04, -0.04	0.03, -0.03	0.02, -0.02

5.5 Real-Time Implementation

Two properties are essential to execute any planning component in a real-world setting. First, it has to be ensured that a solution is found under real-time conditions so that before the end of the current trajectory a new trajectory has been generated. Second, the solution can be sub-optimal, but the generated motion has to be kinematically feasible by the vehicle. The following section describes the necessary adaptions to the off-the-shelf solvers to achieve these two properties simultaneously.

5.5.1 Solver Settings

We used several parameters and custom implementations to tweak the solver to find a suitable solution fast. In Section 5.6 we show the effect of the tuned parameter set in an exemplary scenario. Note that tuning the following parameters helps to find a good solution fast, but there is no guarantee or proof that this parameter setting is optimal. In the following list, we describe the most effective settings.

- **Relative MIP gap tolerance:** The solver is stopped if the current best integer solution and the current best (non-integer) bound divided by the best integer solution falls below this threshold. Usually chosen as ϵ, we use a relatively high value of 0.1 as very low deviations in the objective do not have a significant influence on the resulting trajectory. By this, we also accept feasible integer solutions that are potentially far away from the global optimum.
- **Parallelization:** We use parallel search on multiple CPU cores to find the optimum in the branching tree. This introduces a slight non-determinism in the solution if the solver is terminated before the optimal solution is reached.
- **MIP emphasis:** We instruct the solver to sacrifice time to prove the optimality of a solution but find integer-feasible solutions fast. Due to timing and gap tolerance constraints, it is desirable to chose the best from several integer solutions. Also in our problem formulation usually several feasible solution often lie close to the global optimum, each resulting in a comparable trajectory.
- **Relative objective difference cutoff:** The optimization is forced to ignore new integer solutions that are not at least better by a certain amount than a previous solution. By this, the number of nodes to be expanded is limited, with the drawback that the true optimum can be missed.
- **Frequency to try to repair infeasible MIP start:** We instruct the solver to try repair heuristics often if one or more of the supplied warmstart solutions are not feasible. As we, cf. Section 5.5.2, use all integer feasible solutions from the last instance to warmstart the current instance of the optimization, some of these solutions are invalid but close to a feasible one.
- **Relaxation induced neighborhood search (RINS) heuristic frequency:** This heuristic tries to improve the current best solution. With the warmstarting strategy and other parameterization, we aim to find feasible solutions fast. Based on these, we instruct the solver here to try to improve these feasible solutions.
- **Defining special ordered sets (SOS):** A SOS is a set of variables where at most one variable is nonzero. Defining a SOS the solver uses specialized branching strategies that exploit the special topology of the problem. The region variables ρ form a SOS for each agent. The variables have to be ordered to assign the branching priority. We put a high weight on variables encoding the initial region and order the weight from early to late in the horizon as early decisions on the region influence the solution more.

- **Priority branching:** We instruct the solver to branch early on variables that heavily influence the solution topology. We assign priorities to two groups of variables. First, the variables defining the region ρ. If the region is marked available by ϱ^r, we increase the priority. We also increase the priority for the variables deciding if the rear axle center lies within one environment polygon e.

5.5.2 Warmstarting the Solution

In MIP, as in other optimization strategies, an initial solution can be provided to aid the solver in finding a solution faster. Warmstarting works best if the solution is already primal feasible (a valid but non-optimal integer solution), but also non-feasible or incomplete solutions can serve as a warmstart. With a solution available at the start of the branch and cut process, non-optimal parts of the branch-and-cut tree can be eliminated, and a lower solution bound exists. Also, with heuristics available from the solver that further improve an integer-valid solution, the convergence to the global optimal solution can be highly improved. However, the effectiveness of the warmstart is influenced mainly by the solution quality of the warmstart solution, which is unknown.

To speed up the solution computation and to enforce finding consistent solutions close to the solution of the previous planning step, we warmstart the MIQP solver using the solution from the previous receding horizon instance similar to [MT21]. From this previous solution, we use the decision variables from the start time onwards and perform a simple extrapolation to initialize the variables at the last timestep, which introduces only a neglectable overhead regarding computation time.

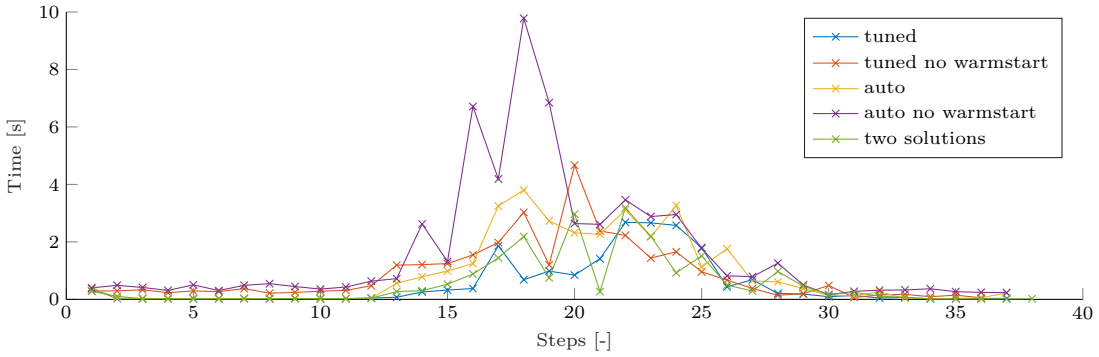
The solver can be warmstarted from several solutions. Therefore, we reuse all integer-feasible solutions from the last receding horizon instance as warmstarts. We input these solutions without modifications, such as shifting from the last timestep to the current timestep. Using this strategy, we have the previous optimal solution available for warmstarting and the non-optimal but integer-feasible solutions that might lead to the current optimal solution faster as the integer decision variables are initialized better.

The less the topology of the problem has changed, the more significant is the effect of the warmstart. Suppose environment sub-polygons, agents, or obstacles are added to the problem. In that case, the decision variables for those cannot be taken from the previous solution, which lowers the efficiency of the warmstart. Also, if the problem drastically changes, an old solution can guide the solver in the wrong direction. Especially the second approach of reusing all integer solutions as warmstart for the next planning cycle in practice had a huge effect on the runtime, which is why we chose to use this approach further in the following. In Section 5.6 and Section 5.5.4, we show the speedup from warmstarting.

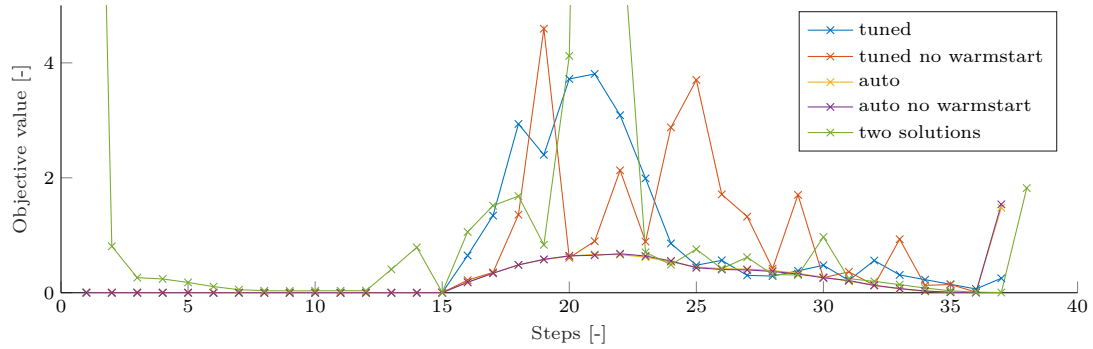
5.5.3 Implementation Remarks

The optimization problem is modeled and solved using the commercial solver IBM ILOG CPLEX Optimization Studio [Int21b]. For development and visualization, we implemented a gateway from MATLAB [The21b] using the command line interface of CPLEX. The main focus of the Matlab implementation is providing a higher level Application Programming Interface (API) including the ability to easily run the optimization repeatedly in the receding horizon setup and debug visualizations.

For production usage, we implemented a C++ gateway to our CPLEX model using the C++ API of CPLEX. As the most inner part, the `CplexWrapper` class holds and handles the optimizer object from the CPLEX API. It also mirrors all parameters from the CPLEX model into a C-struct to ease the read and write access, specifically done in the `ModelInputDataSource` class. The same applies to the decision variables. Furthermore, the `CplexWrapper` maintains the warmstart solution(s), statistics on the problem and the solution, and advanced solver settings such as branching priorities.



(a) Solution time



(b) Objective function value

Figure 5.10: Parameter analysis in a lane tracking scenario in a curve

To provide the user with a higher level API that does not expose the technical parameters of the CPLEX implementation but provides high-level service methods, we implemented the `MiqpPlanner` class. It holds a representation of the optimization problem and offers high-level service methods to add, update, or remove an agent or an obstacle. Also, it delegates to the necessary convexification methods for non-convex environments or obstacle polygons. Furthermore, the reference line is subsampled and prepared to be used as a reference trajectory using an intended acceleration profile and a target speed. In an initial step, we load or compute the parameters of the optimization model that are constant over time, such as the fitting polynomials.

This high-level API is wrapped and can be called by an agent in the open-source behavior simulation platform BARK [Ber+20] or the trajectory planner in the open-source driving stack Apollo, as described in Chapter 6. In BARK, MINIVAN is available as a dedicated agent implementation. Besides the conversion of data for the different APIs, comparable steps are needed as in the Apollo integration. As BARK is focused on multi-agent interactions, the integration of the behavior of other agents is essential. As MINIVAN needs a reference trajectory for the fellow agent, we generate these references using an existing behavior model in BARK to mimic a prediction component.

5.5.4 Analysis of Parameterization Effects

Parameterization and Branching Customization Effects We analyzed the effectiveness or the solver parameterization in different scenarios, such as a lane merging scenario and a lane tracking scenario with a sharp turn (see Figure 5.22). As all scenarios yield comparable results, we do not discuss them individually here. The results of the lane tracking scenario are depicted in Figure 5.10. We compare the following settings:

- The tuned parameter set, balanced between solution speed and solution quality, also including custom branching rules (tuned),
- the effect of the warmstart (warmstart),
- letting the solver decide for the parameters (auto), and
- stopping the optimization after the second integer-valid solution (two solutions).

We set the maximum available time for the solver to 10 s to observe the effect of the different parameterizations and analyze the total runtime of the solver alongside the objective value and the optimality gap as a measure for the solution quality. We always find an optimal solution within the given tolerances. When stopping after the first valid solution, we observed that this solution is often not suited in terms of solution quality and unsuited in the current driving situation. Therefore, in this experiment, we consider the second integer-valid solution.

We can observe the effect of the warmstart. In most situations, the solution with a warmstart is significantly faster than without a warmstart. However, situations exist where a warmstart yields a higher runtime. These are mostly situations where the problem topology changed, e.g., a new environment polygon has been added, and the previous solution is misleading. The tuned parameter set yields slightly worse solutions compared to the default solver settings with much lower optimality thresholds. In non-trivial situations, the objective value of the tuned parameter set is higher but still in the same order of magnitude. Stopping the solver after the second integer solution does not bring huge benefits in terms of runtime. Worse solutions for the warmstart are available, and often the solution is worse in terms of objective value and optimality gap.

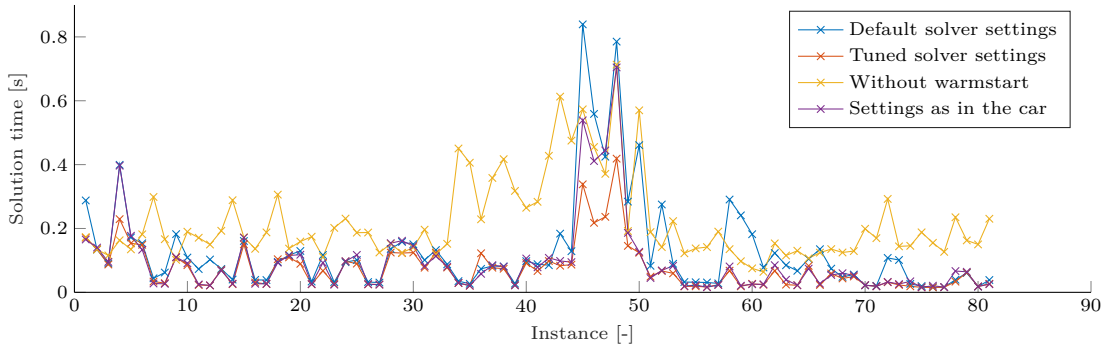
Parameter Analysis of Real-Road Test Drives In simulation scenarios, we have already shown that choosing a suitable parameter set can greatly influence the solver's performance. Also, providing a valid warmstart solution has a great effect on the solution speed of the MIP. In the following, we will show that this does not only apply to simulation benchmark scenarios but also in real-road test drives. In Section 6.6 we will describe the scenarios in detail, specifically

- **Static vehicle avoidance**: avoiding a static vehicle on the right side, see Figure 6.10.
- **Vehicle following**: approaching a slower preceding vehicle, eventually slowing down and accelerating again once the preceding vehicle accelerates, see Figure 6.11.
- **Pedestrian avoidance**: avoiding a static pedestrian standing on the road, see Figure 6.12.

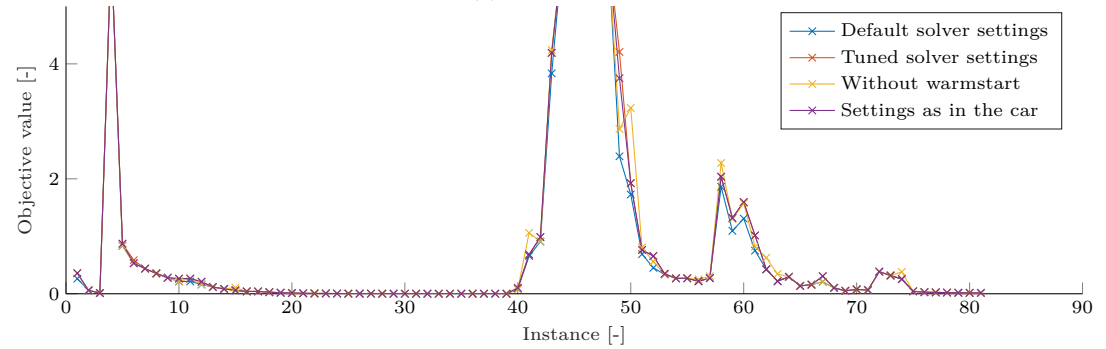
Choosing the suitable parameter set is difficult and crucial. Depending on the situation, different sets of parameters are optimal regarding runtime and solution quality. Often, the ideal parameter set can only be determined a posteriori. We used a tuned parameter set for our experiments that shows a good performance on average but can be improved for specific scenarios. Generally, the tuned parameter set shows better performance than defaulting all parameters, as we insert more knowledge into the system. Strategies for adapting the parameters to the current driving scenario have not been developed in this work.

To assess the performance of the tuned parameter set, we serialize all input data (including the warmstart information) for each planning instance while driving. We then rerun these optimization problems offline in four different settings:

- Default solver settings without any parameter modifications, especially no limitation of the available solution time (Default solver settings),
- tuned parameter setting: Modification of the in-vehicle parameter set to the specific properties of the scenario (Tuned solver settings),
- parameter settings as in the vehicle but without warmstart (Without warmstart), and



(a) Solution time



(b) Objective function value

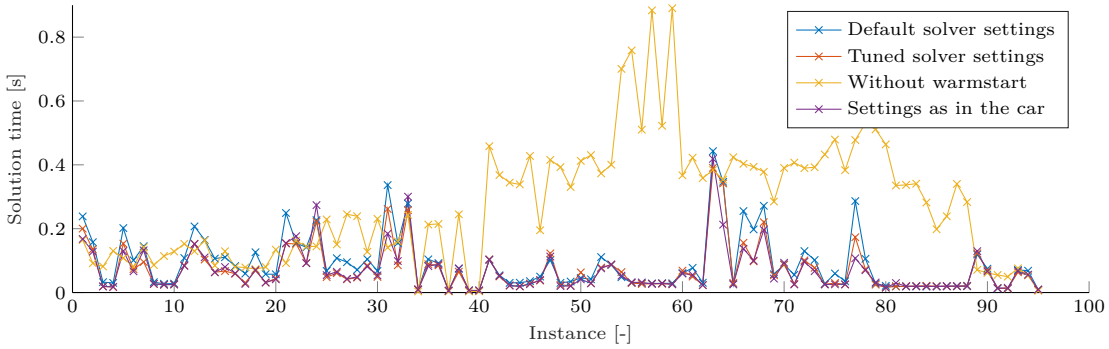
Figure 5.11: Parameter analysis in a vehicle following scenario comparing the default solver settings to a tuned general purpose parameter set and a posteriori tuned parameter set for this scenario. The effect of warmstarting is shown.

- same settings as in the vehicle: To ensure comparable results, we execute the same setting as in the vehicle on the test machine (Settings as in the car).

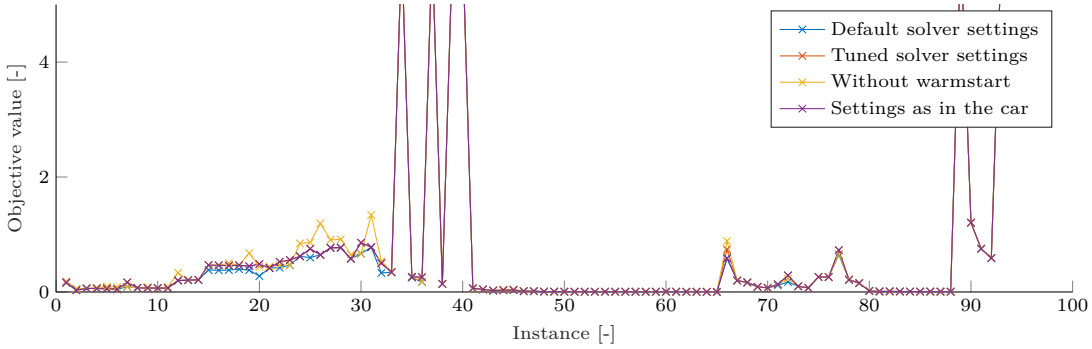
We analyze three test scenarios (see Section 6.6), vehicle following, pedestrian avoidance, static vehicle avoidance. The traces of the solution time and the objective function values are depicted separated for the three scenarios in Figure 5.11, Figure 5.12, and Figure 5.13.

We observe the significant effect of providing a warmstart solution for real-time execution in all scenarios. In these practical experiments, we barely see negative effects of the warmstart. In the vehicle avoidance and following scenario, instances exist where warmstarting has no or even a negative effect on the runtime.

As expected, the default solver settings yield a higher runtime than both tuned parameter sets. This effect is clearly visible in the vehicle following scenario. No clear conclusion can be drawn which parameter setting reduces the number of peaks in the runtime. Peaks occur if the solution of one receding horizon instance yields a more complicated optimization problem. This can have several reasons, such as unfavorable initial conditions of the ego vehicle, perception or prediction errors of obstacles, or a changed road geometry with many additional environment polygons. These peaks are extremely critical in real-time evaluation and should be avoided as much as possible. While the tuned parameter set avoided such peaks in simulation studies, this effect cannot be clearly shown in on-road driving scenarios. Generally, it can be observed that with a good warmstart solution, the evaluation time with default and the tuned parameter set are mostly very close. This is easily seen in the fully static vehicle avoidance scenario. In a situation where more adaptions to the initial solution are necessary, such as in the second half of the vehicle following scenario, the tuned parameter set is significantly faster. Table 5.5 summarizes the mean



(a) Solution time



(b) Objective function value

Figure 5.12: Parameter analysis in a vehicle avoidance scenario comparing the default solver settings to a tuned general purpose parameter set and a posteriori tuned parameter set for this scenario. The effect of warmstarting is shown.

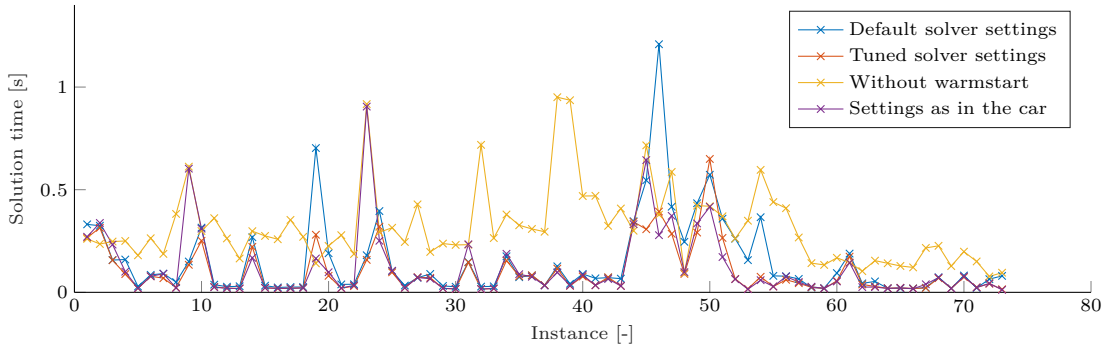
Table 5.5: Runtime improvement in percent between 0 and 1, compared to the default parameter setting.

Scenario	Tuned	Without warmstart	In car
Vehicle avoidance	0.17098	-0.38412	0.17567
Vehicle following	0.29339	-0.33518	0.21956
Pedestrian avoidance	0.25172	-0.44545	0.12529

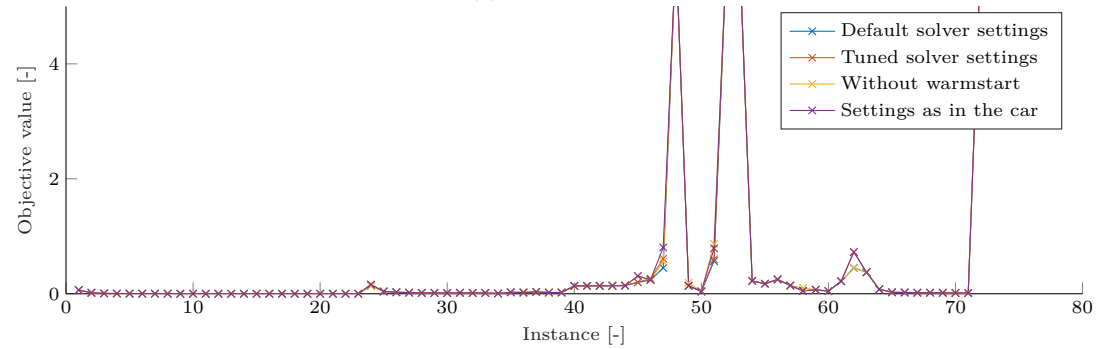
runtime improvements over all instances of one scenario compared to the default setting in percent. Note that without a warmstart, the runtime increases, and therefore the number is negative. It can clearly be observed that the parameters used for the test drives can further be improved.

The default parameter setting always performs best in terms of the objective value. This effect is expected, as we specified higher gap tolerances and integer solution steps in the tuned parameter settings. Recall that the objective function is a squared sum of the deviation from the reference trajectory plus the slack terms. Therefore the deviation in the objective function between the tuned and the default parameter set can be interpreted as the deviation of the tuned solution from the true optimum. In most instances, this deviation is small and close to zero. This indicates that the solution we compute is close to the global optimal solution, even if the solver was not parameterized to effectively find this global optimum. We lack the proof of this optimality with our chosen parameterization. However, for example, at the beginning of the vehicle following scenario, this deviation is big, which indicates that the trajectory computed in the car was significantly different from the true optimal trajectory.

In the posterior analysis, we then tweaked the in-vehicle parameter set specifically to the given scenario. We observed in all scenarios that with the optimized parameter setting, the peaks in the solution time can significantly be lowered compared to the chosen in-vehicle parameter setting



(a) Solution time



(b) Objective function value

Figure 5.13: Parameter analysis in a pedestrian avoidance scenario comparing the default solver settings to a tuned general purpose parameter set and a posteriori tuned parameter set for this scenario. The effect of warmstarting is shown.

without changes in the objective function. If this effect generalizes to a broader set of scenarios and how an automated adaption of parameters can be achieved is left for future research.

Conclusion In various simulated benchmark scenarios, we showed that inserting knowledge into the problem topology and exceptions on the solution could reduce the solution time while still achieving qualitatively good results. This was done by parameterization and custom branching rules. In scenarios that are easy to solve, parameter tuning only showed a minor effect. It helps to reduce high runtimes in complicated situations, which is essential for real-time application.

These findings from simulation studies also transfer to real scenarios. In three on-road benchmark scenarios, we showed that using custom-tuned solver parameters can further improve the solution speed, in most cases with equivalent solution quality. A selection of parameters specific to one scenario can speed up the solution more and especially reduce critical peaks in the runtime. By far, the most significant effect on the solution time is achieved by providing a warmstart solution.

5.6 Complexity Analysis

In this section, we will analyze the evaluation time and the size of the MIQP with growing complexity of the optimization problem, such as more obstacles. The experiments are all performed on the same machine with an Intel Core i7-9850H 2.60GHz CPU with 12 cores and 32GB of memory. We analyze the following for each scenario:

- The solution time of the MIQP with and without warmstarting the problem.

- The number of iterations needed for the solver to generate the result as a rough measure for the problem complexity with and without warmstarting the problem.
- The number of constraints, float variables, and binary variables as a measure for the problem size. This number can also be derived directly from the problem definition, we depict the statistics from the solver here. The shape of the curves in all cases reflect the exceptions from theoretical considerations. We generally observe, that the number of float variables is the same for a varying number of regions, which is expected.

The scenarios are randomly generated with a fixed random seed within their specifications. 100 variants of the same scenario with varying initial configurations are simulated, and the results are averaged to avoid randomness and runtime non-determinism on the machine. We set the maximum available time for the solver to a high value (30 s) to highlight that also, in complicated scenarios, a solution can be found. We filter out all failed optimizations as these correspond to unsolvable scenarios, e.g., an obstacle placed randomly just in front of the vehicle. Note that if we compare a different number of regions, the available orientation interval for each agent is the same.

Scaling with the Number of Points Here we analyze how the problem scales with a growing number of trajectory points that are optimized. As a scenario, the ego agent drives on a straight line inside one environment polygon and has to avoid one static obstacle blocking the reference line. For trajectories with 5 points or less, the optimization plans a braking trajectory and stops in front of the objects. With more points, a lateral avoidance is planned. Therefore, the times up to 5 points are roughly identical. Figure 5.14 shows the comparison for a growing number of points with a varying number of regions and warmstart activation.

We observe that with an existing warmstart solution, the solution time barely scales with the number of steps. Without a warmstart solution, the runtime significantly rises with the number of steps. For 10 points and below, the evaluation time is constantly less than one second for 16, 32, and 64 regions. For 10 to 20 points, we observe moderate runtimes below 2 s for 16 regions and below 3.5 s for 32 regions, but a strong increase in runtime until 15 s for 64 regions. For more than 20 points, the runtime rises exponentially and becomes untractable. As there is no randomness in the scenarios, the error bars only reflect the nondeterministic execution time. Without warmstarting, the number of iterations rises to 10^5 for all regions. Oscillating numbers of iterations are needed with an available warmstart. For less than 6 points, a significantly lower number of iterations of up to 100 is needed. Correlating with the problem size, we observe an increase slightly higher than linear. Note that the number of float variables is equal for any number of regions.

Static Obstacles Scaling As a scenario, we randomly place static obstacles around a straight reference line the planner shall track with a constant speed. The varying initial configurations of the obstacles are drawn from a bounded equal distribution in x-direction within a range of -5 m to 30 m and in y-direction within a range of ± 4 m around the ego vehicle center. The more obstacles are placed on the reference, the more evasive trajectories have to be planned. Due to the random placement of obstacles, some easy scenarios exist with obstacles only relatively far from the vehicle and complex ones with only obstacles close to the vehicle. To see an effect of placing an additional obstacle, the area for obstacle placement is relatively small and has to be near to the ego agent. Therefore, for more than 18 obstacles, most optimization problems become infeasible. This effect is expected and intentional to have comparable scenarios for fewer obstacles. The variance in the runtime reflects the scenario complexity. Figure 5.15 shows the analysis.

We observe no significant scaling effects with an increased number of regions. Below 10 obstacles, we observe a linear scaling with the number of obstacles without a warmstart. With an available warmstart, only a marginal scaling effect is visible. At around 10 obstacles, we observe a very high increase in runtime, also with worst-case runtimes up to 30 s. Starting from 10 obstacles, the solution time variance significantly increases. This observation is expected to some extent, as easy

and hard settings exist in the randomized scenarios. For 64 regions, this effect occurs earlier. The number of iterations reflects the runtime analysis; we observe a growth in the number of iterations needed until 10 obstacles. Afterward, the number of iterations stabilizes. With a valid warmstart, significantly fewer iterations are needed. Correlating with the problem size, we observe a linear increase. Note that the number of float variables is equal for any number of regions. If we place all obstacles very far away from the reference, we observe that the evaluation time of the planner does not scale with the number of obstacles.

Moving Objects Scaling As a scenario, we randomly place objects close to the ego vehicle and let them drive in the same direction as the ego vehicle with a randomly assigned speed within a $\pm 20\,\%$ range of the ego vehicle. These objects can either be modeled as dynamic obstacles that need to be avoided or as agents for which the model plans an interactive trajectory. The varying initial configurations of the obstacles are drawn from a bounded equal distribution in x-direction within a range of $\pm 30\,\mathrm{m}$ and in y-direction within a range of $\pm 5\,\mathrm{m}$ around the ego vehicle center. Figure 5.16 and Figure 5.17 show the analysis separately depicting the analysis with and without warmstart for figure clarity.

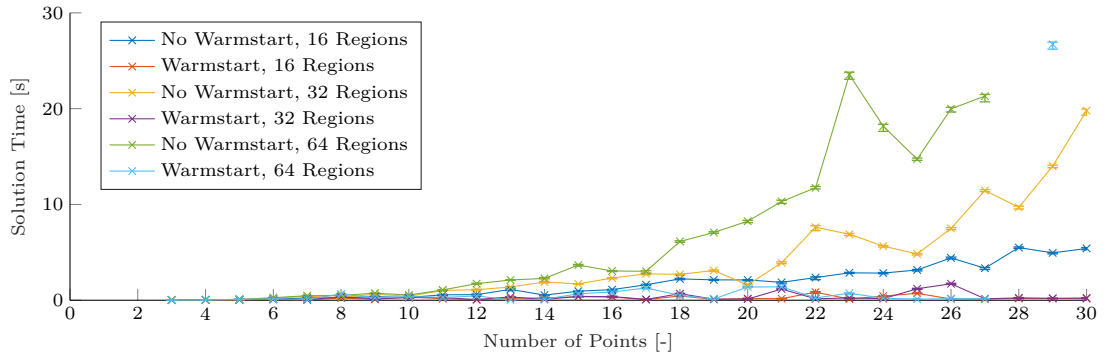
If other objects are modeled as moving obstacles, we observe a good performance with up to 9 obstacles. For 16 regions, the effect of warmstarting is not significant. Generally, warmstarting helps to lower the variance in the solution time. Again, the random placement of other objects yields easy and hard-to-solve scenarios, which explains the high variance. The increase of iterations and solution time with the number of regions is as expected.

Our analysis shows that the problem becomes intractable for more than two other objects if modeled as interacting agents. This effect is also visible in the exponential growth in the problem size. Beginning with seven other dynamic agents (and 16 regions), the solver cannot find a solution in the given time in most scenarios. We, therefore, show the evaluation until six other dynamic agents. With more agents, the runtime bound is reached in a majority of cases non-trivial cases, which distorts the evaluation. With more regions, this effect is observed for fewer objects. If we placed all objects modeled as obstacles very far from the reference, we observed that the planner's evaluation time did not scale with the number of obstacles. Warmstarting and the number of regions barely affected the runtime. In contrast, we observed a comparable scaling effect in the multi-agent case regardless of the object distances.

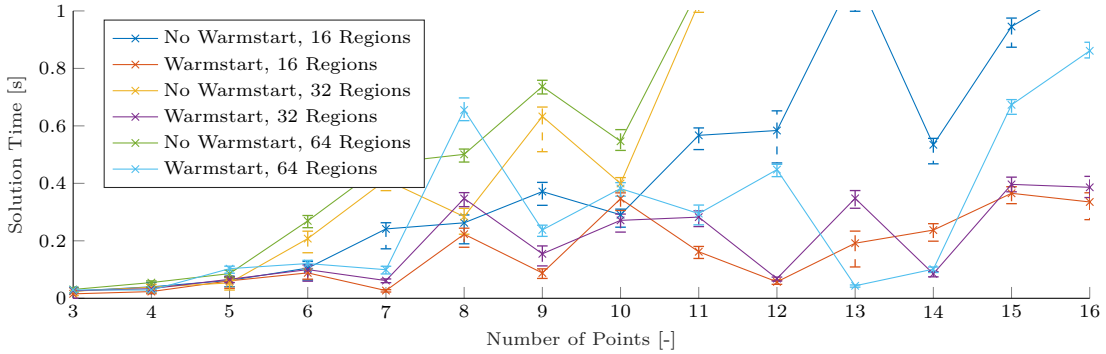
Environment Polygon Scaling As a scenario, we drive along a straight reference line intersecting with an increasing number of environment polygons. The polygons are hardcoded (to ensure the reference line passes all environment polygons and none are merged in the preprocessing); therefore, there is no randomness in the scenario. The variance in the solution time reflects the indeterministic execution time.

We observe only a moderate scaling effect for the number of environment polygons with a valid warmstart. Without a warmstart, a growth in runtime is visible. This observation is of high practical relevance, as while driving it frequently occurs that the environment polygons change, which results in non-suited or invalid warmstarts for the next iteration yielding longer runtimes. We do not see an effect with a growing number of regions. Figure 5.18 depicts the runtime, number of iterations, and problem size. The evaluation time is not affected by environment polygons that are placed far away from the agent.

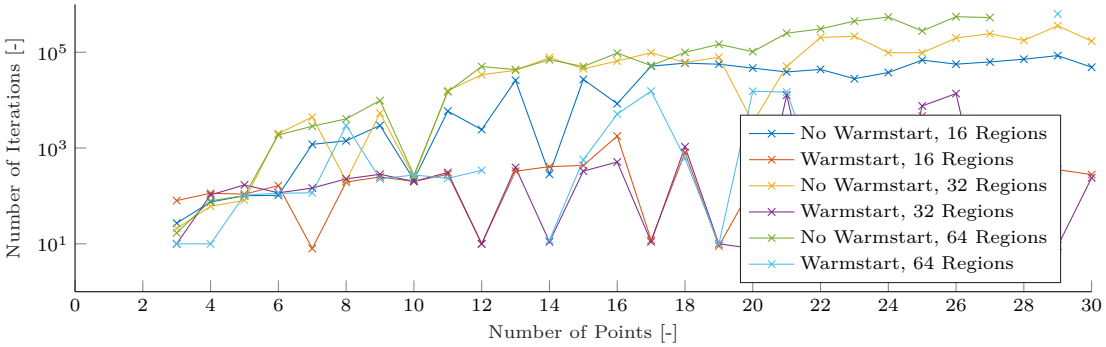
Note that we used a straight line as a reference here. In reality, the number of regions especially grows when driving in a curve, and the geometry of this curve is approximated by several polygons. In a curve, the number of regions crossed by the trajectory also increases, and the increased usage of both regions and environment polygons results in higher runtimes. Still, solely using more environment polygons does not yield exponential runtime scaling.



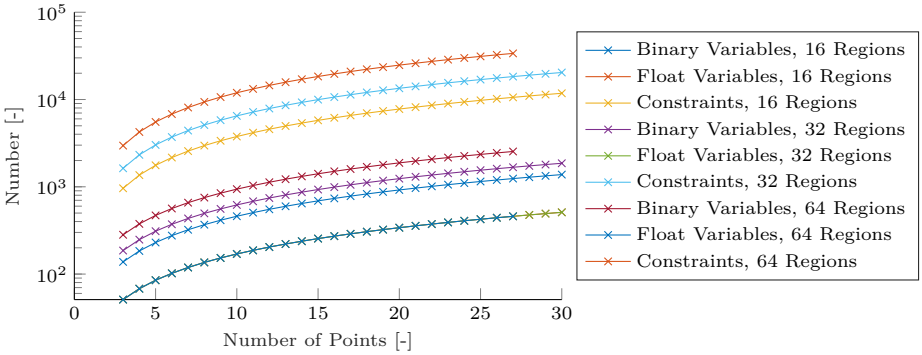
(a) Solution time. The error bars show the 5% and 95% quantiles.



(b) Solution time zoomed to 3-16 points. The error bars show the 5% and 95% quantiles.

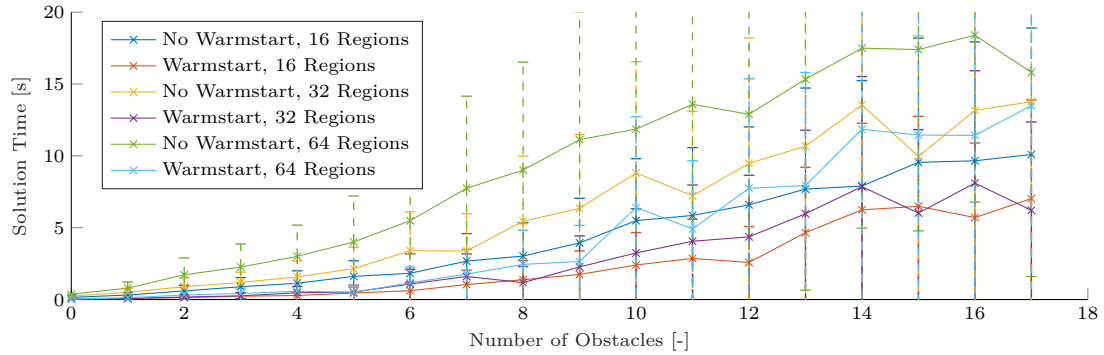


(c) Number of iterations.

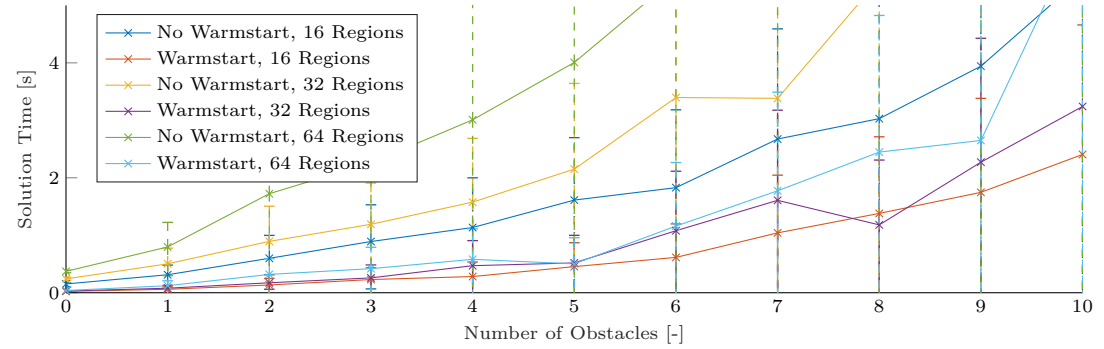


(d) Number of binary and float variables and number of constraints.

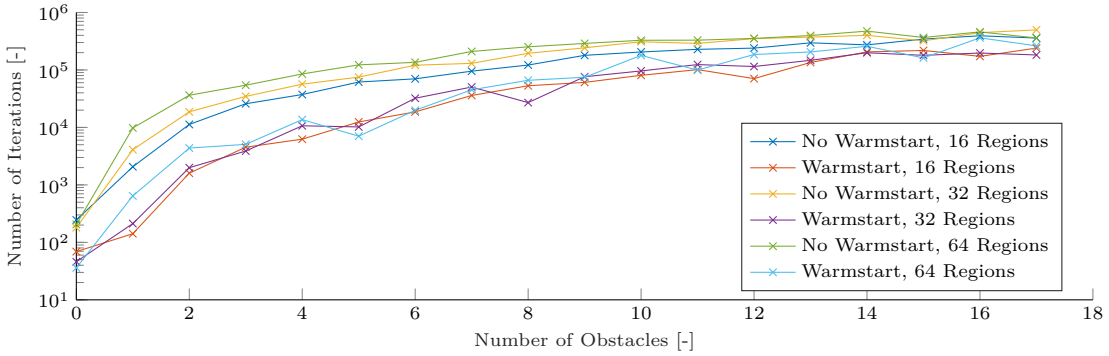
Figure 5.14: Scaling of the MIQP with an increasing number of points on the horizon.



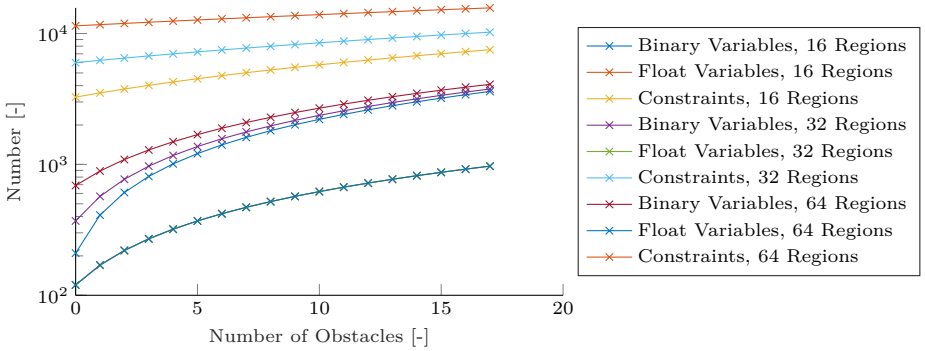
(a) Solution time. The error bars show the 5% and 95% quantiles.



(b) Solution time zommed to 1-10 obstacles. The error bars show the 5% and 95% quantiles.

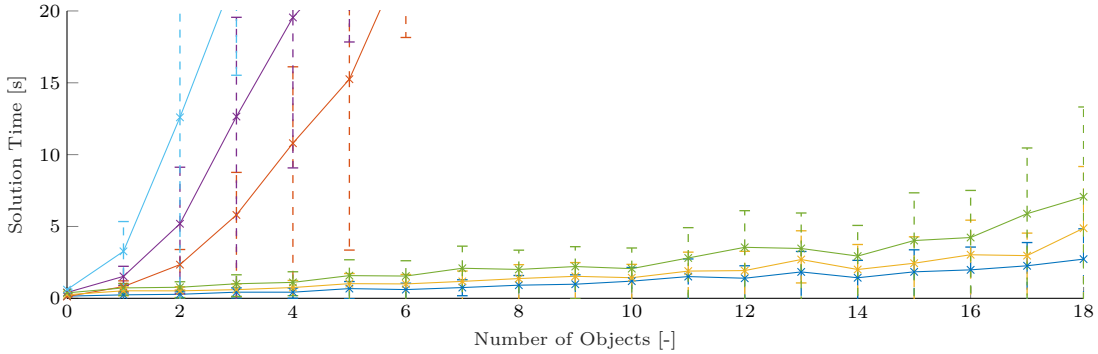


(c) Number of iterations.

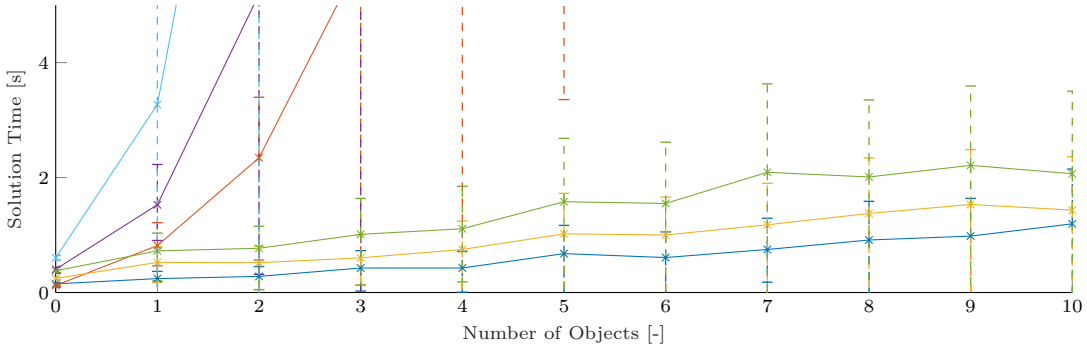


(d) Number of binary and float variables and number of constraints.

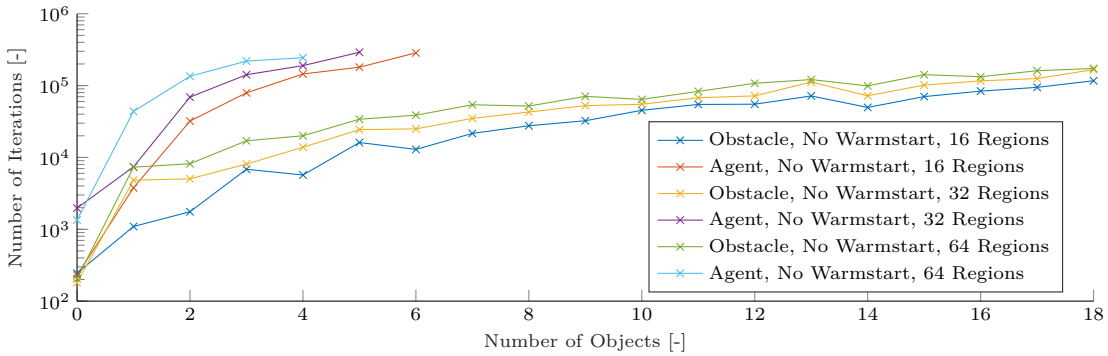
Figure 5.15: Scaling of the MIQP with an increasing number of static obstacles on the horizon.



(a) Solution time. The error bars show the 5% and 95% quantiles. For the legend see Figure 5.16c.

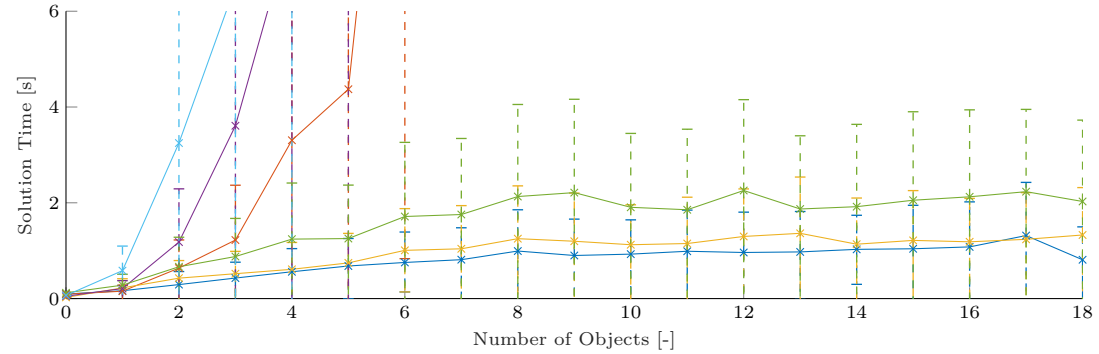


(b) Solution time zoomed to 1-10 objects. The error bars show the 5% and 95% quantiles. For the legend see Figure 5.16c.

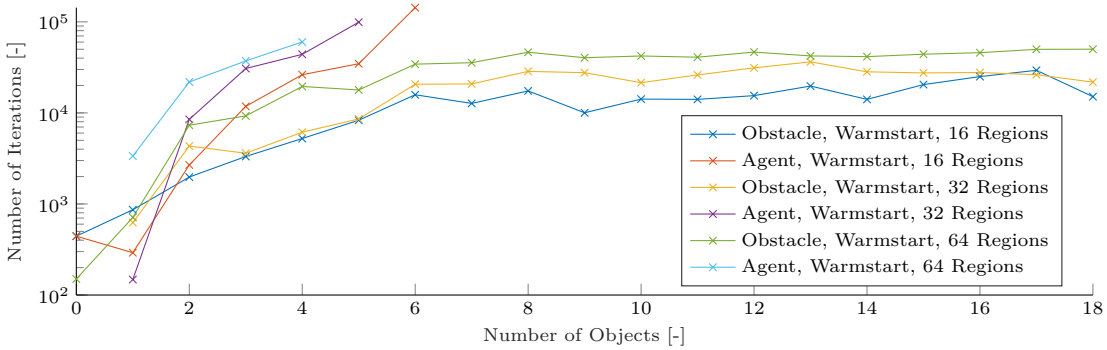


(c) Number of iterations.

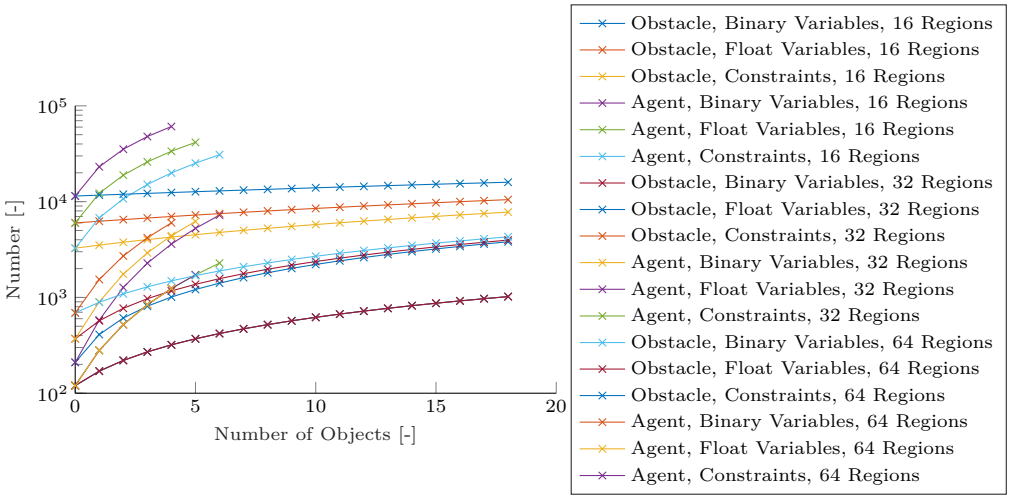
Figure 5.16: Scaling of the MIQP with an increasing number of dynamic objects (as dynamic obstacles or agents) on the horizon without providing a warmstart solution. (See Figure 5.17c for the number of constraints and obstacles.)



(a) Solution time. The error bars show the 5% and 95% quantiles. For the legend see Figure 5.17b.

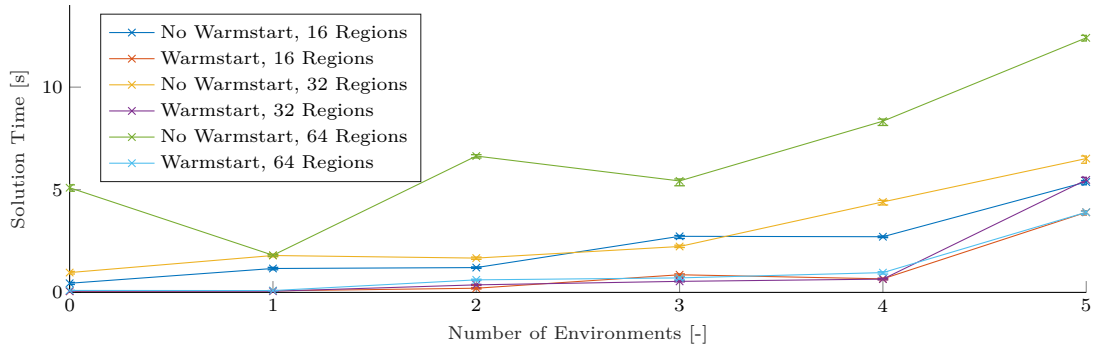


(b) Number of iterations.

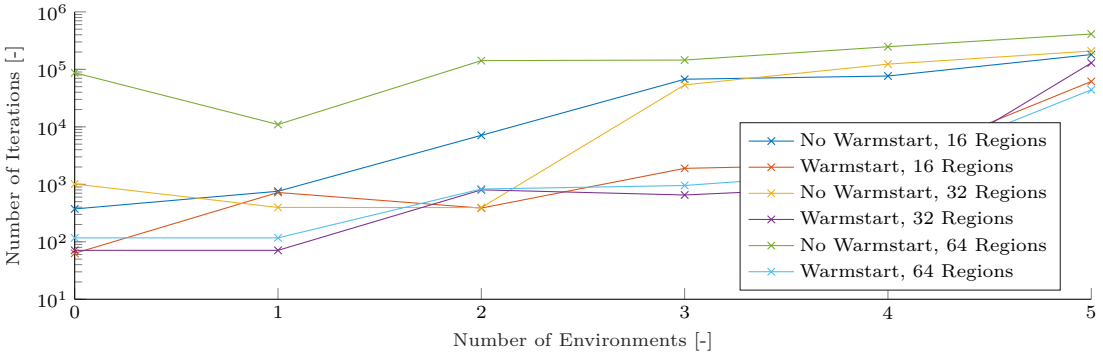


(c) Number of binary and float variables and number of constraints.

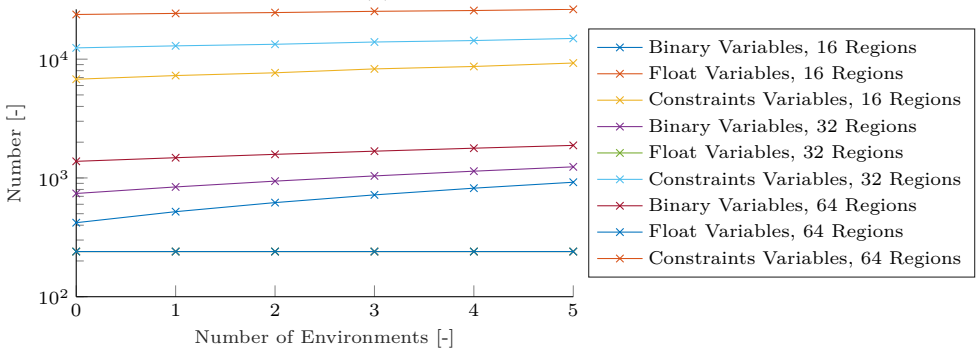
Figure 5.17: Scaling of the MIQP with an increasing number of dynamic objects (as dynamic obstacles or agents) on the horizon with a provided warmstart solution.



(a) Solution time. The error bars show the 5% and 95% quantiles.

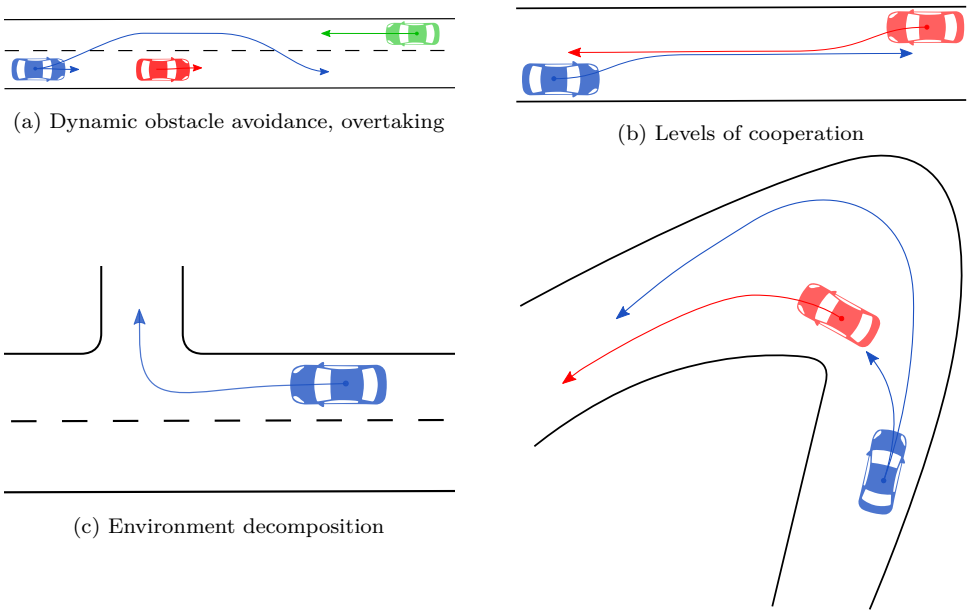


(b) Number of iterations.



(c) Number of binary and float variables and number of constraints.

Figure 5.18: Scaling of the MIQP with an increasing number of convex environment polygons on the horizon.



(a) Dynamic obstacle avoidance, overtaking

(b) Levels of cooperation

(c) Environment decomposition

(d) Autonomous racing

Figure 5.19: Schematic sketches of the evaluation scenarios.

Conclusion Summing up, we observe that for real-time applicability, 10 points on the planning horizon are feasible, even with high requirements on the accuracy of the approximation using 64 regions. 32 regions are a good compromise between solution time scaling and accuracy.

For a low runtime, warmstarting the solver is essential in complex situations. In some situations, warmstarting can also reduce the variance in the solution time. However, warmstarting does not fully mitigate the problem of a high solution time variance when the optimization problems are complicated. This effect is to some extent expected in the setup of this analysis, but the effect that the evaluation time is unstable remains a serious challenge in real-time application. Warmstarting can, in certain constellations, also have negative effects on the solution time, and the question remains open if these scenarios can be distinguished a priori. The solver can handle a realistically high number of static or dynamic obstacles, but more than three interacting agents are not tractable in a real-time setting.

5.7 Demonstration of Benefits and Effectiveness in Simulated Scenarios

We will now evaluate the performance of MINIVAN in five different scenarios. The scenarios are schematically sketched in Figure 5.19. We in Section 5.4.2 already proved that the result of the optimization is a drivable trajectory for a non-holonomic vehicle model. Table 5.6 gives an overview of the application scenarios. By coordinated, we denote a connected setting with Connected Autonomous Vehicles (CAVs) that follows a global optimal plan. By reactive or proactive, we denote the integration into a mixed-traffic scenario with the ego vehicle either showing passive behavior or actively enforcing its own goals.

5.7.1 Avoiding Dynamic Obstacles Within the Road Boundaries

We start with experiments having a single agent in an environment with multiple (static or dynamic) obstacles present. We here evaluate one receding horizon instance. This example considers a

Table 5.6: Overview of the five different evaluation scenarios used in this section to showcase the capabilities of MINIVAN. ◊ denotes not applicable, ♦ denotes applicable.

Section	5.7.1	5.7.2	5.7.3	5.7.4	5.7.5
Scenario	Dynamic obstacle avoidance	Levels of cooperation	Environment decomposition	Overtaking	Racing
Schematic figure	5.19a	5.19b	5.19c	5.19a	5.19d
Number of agents	1	2	1	3	2
Coordinated	◊	♦	◊	♦	◊
Proactive	◊	◊	◊	♦	♦
Reactive	♦	◊	◊	♦	♦
Receding horizon	◊	◊	♦	♦	♦
Speed	medium	low	low	high	high

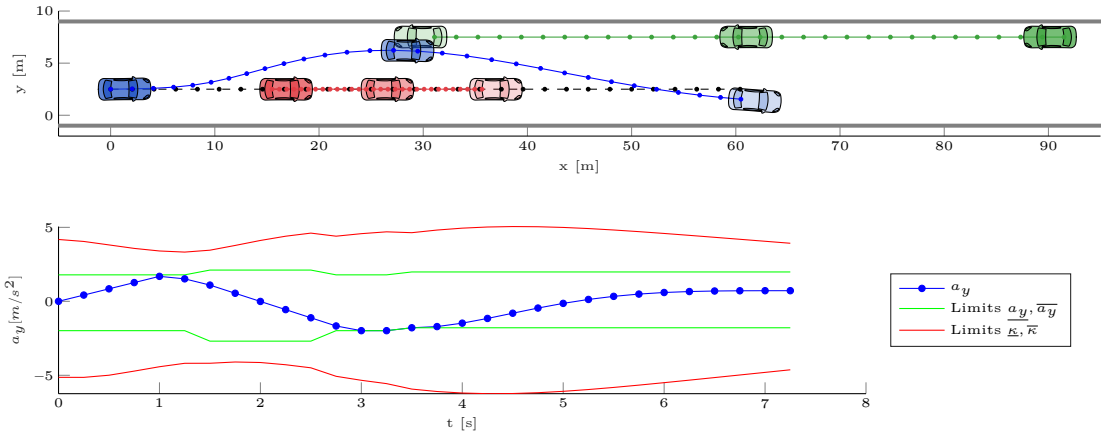


Figure 5.20: An overtaking scenario with oncoming traffic. The upper figure shows the trajectories for the ego vehicle (blue) along the reference (black, dashed) and the other traffic participants (red, green) modeled as dynamic obstacles. a_y of the planned trajectory stays within the region-dependent acceleration bounds as well as the bounds approximating the curvature limit. (graphic from [EKK20], ©2020 IEEE)

two-lane road, where the ego vehicle drives at the speed of 8.33 m/s and approaches a slower vehicle traveling at 2.77 m/s. There is oncoming traffic in the other lane traveling at 8.33 m/s. Both other vehicles are modeled as dynamic obstacles and not as interacting agents. The reference trajectory represents the centerline of the right lane traveling at the reference speed. We obtain deterministic predictions of those two traffic participants and include them as dynamic obstacles in the optimization. We only apply a cost term on reference position tracking. We use a model with 32 regions and a velocity-dependent front axle approximation. We simulate one timestep.

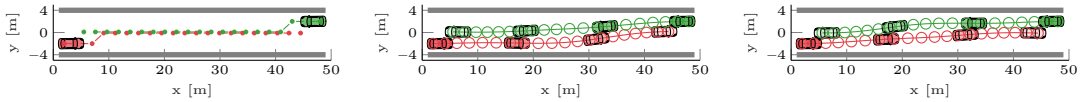
Figure 5.20 shows that the optimizer is able to find a trajectory that overtakes the red vehicle in front and changes back to the right lane to avoid the green, oncoming vehicle. We observe that the acceleration in the y-direction stays within the acceleration limits. It is much closer than the curvature-induced acceleration limit following (5.15), which is reasonable, as the lateral movement in straight scenarios traveling at moderate or high velocities is constrained by the inertia of the vehicle, not the non-holonomics. The scenario shows that with a long planning horizon of $N = 30$ steps, resulting in a 7.25 s long trajectory, a smooth valid avoidance trajectory is generated.

5.7.2 Levels of Cooperation in a Negotiation Scenario

We evaluate the effect of the cooperation factor λ in a negotiation scenario with two agents A^1 and A^2. The scenario is fully symmetric with two vehicles placed on a two-lane road in oncoming

Table 5.7: Quantitative evaluation of the negotiation scenario. We compare the overall distance to the reference, the time the reference is reached, and the contributions to the cost function of each agent.

λ	Dist. A^1	Dist. A^2	Idx A^1	Idx A^2	Cost A^1	Cost A^2
0	54.684	19.645	-	11	3.7452	5006.2
0.1	33.872	20.181	18	11	899.19	4568.5
0.2	30.788	21.362	17	12	1552.6	4183.6
0.3	28.937	22.543	17	14	2114.1	3796.1
0.4	28.383	22.894	17	14	2731.3	3305.4
0.5	22.98	28.297	14	17	2773	3380.1
0.6	22.884	28.383	14	17	3304.4	2731.3
0.7	22.532	28.929	14	17	3795.5	2113.2
0.8	21.364	30.772	12	17	4183.5	1551.5
0.9	20.168	33.865	11	18	4568.3	898.88
1	19.646	54.666	11	-	5006.2	3.699



(a) Reference and initial conditions (b) Cooperative solution for $\lambda = 0.2$ (c) Cooperative solution for $\lambda = 0.6$

Figure 5.21: Different solutions in a scenario with conflicting references. The optimized solution is leveraged with the cooperation factor λ. (graphic from [KEK20], ©2020 IEEE)

direction with road boundaries. Both reference lines do not track the lane centers but the road center. This yields conflicting goals (Figure 5.21a). The solution can be balanced to favor one over the other agent. The vehicle trajectories change as λ changes (Figure 5.21b and Figure 5.21c).

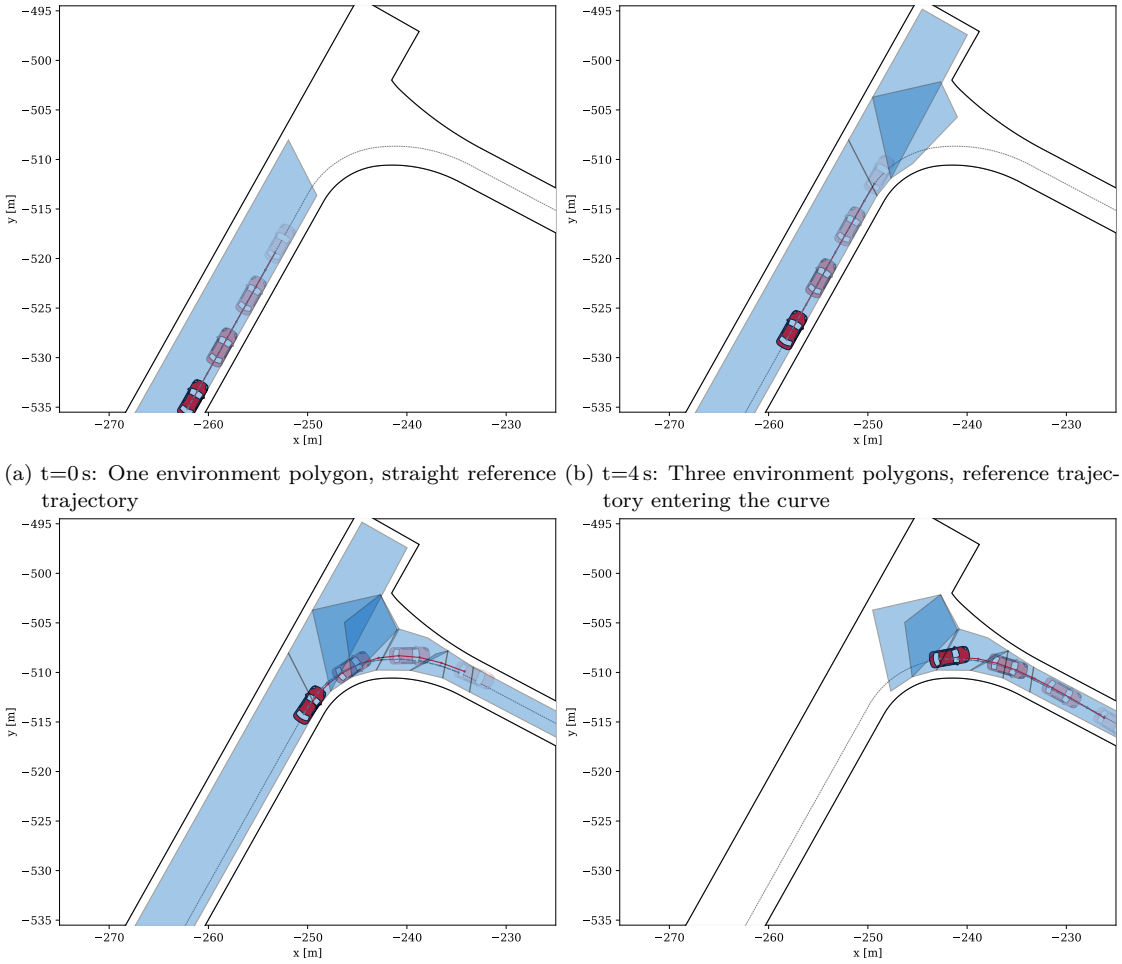
In Table 5.7, we qualitatively show the effect of varying λ. We analyze a single run of the algorithm. By Dist. $A^{\square}$ we denote the accumulated distance overall N timesteps in meters from the solution trajectory to the reference for the respective agent. The column Idx $A^{\square}$ indicates at which time index the respective agent has reached the reference trajectory. We also state the contribution of each agent to the global cost function, denoted by Cost $A^{\square}$. All three metrics show the same trend: by varying λ, the respective agent is favored.

We observe that for $\lambda \approx 0.5$, all metrics are balanced, but also with a strong favor of one agent. Still, valid solutions are computed. This indicates that the cooperation factor can effectively be used to realize cooperation on a continuous level.

5.7.3 Environment Decomposition and Driving Smoothness

At a T-intersection located in Munich, we analyze how the road environment is processed for MINIVAN. From a two-lane road, a curve is planned into a narrow one-lane road. The initial and target speed is set to 4 m/s. Hence the vehicle does not have to decelerate in the curve.

Figure 5.22 shows the evolution of the scenario. The reference line (dashed gray thin) the vehicle shall track is processed to a reference trajectory (dashed gray bold) at specified points in time. The non-convex road environment (solid black) is split into several convex polygons (light blue), starting from one when the vehicle approaches the intersection and ranging up to eight while the vehicle crosses the intersection. The receding horizon formulation dynamically adds or removes environment polygons as necessary. As the optimizer chooses to take a slightly wider curve, it decelerates slightly when entering the curve. It accelerates again after the apex to minimize the distance to the reference trajectory, a behavior that human drivers also show. We observe that the vehicle shape always stays within the road boundaries (solid black), as the approximating circles fully enclose the vehicle shape. The planned trajectory (solid red) is always inside at least one of the convex environment sub-polygons (light blue). The acceleration change between two steps is



(a) t=0 s: One environment polygon, straight reference trajectory

(b) t=4 s: Three environment polygons, reference trajectory entering the curve

(c) t=12 s: Eight environment polygons, curved reference trajectory

(d) t=17 s: Six environment polygons, reference trajectory existing the curve

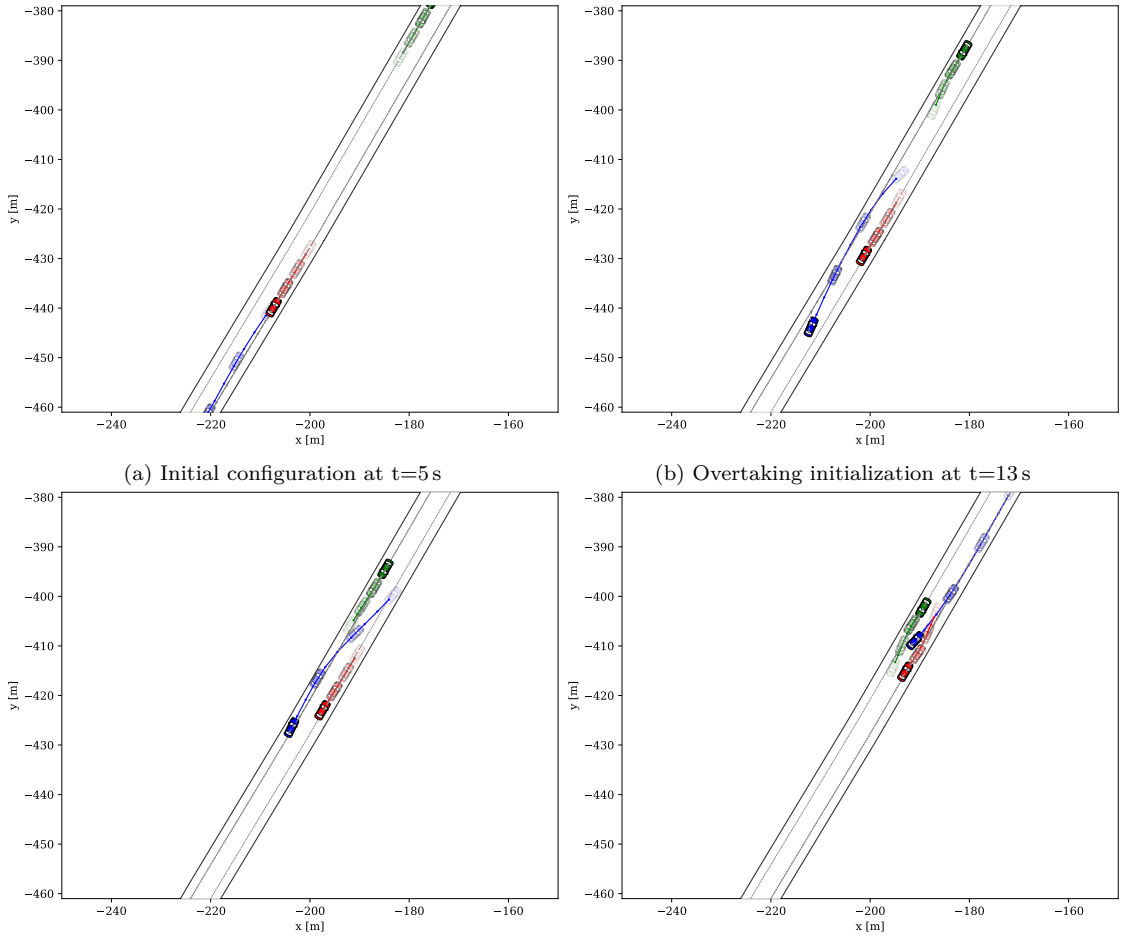
Figure 5.22: Processing of reference line and road environment along the planning horizon. The reference line is depicted in thin gray, the reference trajectory in thick gray, the planned trajectory in red, and the environment sub-polygons in blue.

in a range of $-0.076\,\mathrm{m/s^2}$ to $0.118\,\mathrm{m/s^2}$ and the steering angle change in a range of $-0.062\,\mathrm{rad}$ to $0.096\,\mathrm{rad}$ resulting in a smooth a comfortable motion without hard changes. This indicates that the agent follows a consistent plan over the time instances in the first points of the trajectory.

From this scenario, we conclude the following: MINIVAN can also operate in more complicated environments, such as sharp curves and narrow passages without violating the environment bounds. Also, the close and smooth tracking of a non-holonomic reference trajectory is possible. Referring to the other multi-agent scenarios demonstrated in this section, we with this experiment motivated that deviations from the reference are necessary to generate a valid multi-agent plan and are not the default behavior of the model.

5.7.4 Cooperative Overtaking with Robustness to Prediction Errors

We consider the ego vehicle overtaking a slower vehicle with oncoming traffic in this scenario. Due to the parameterized speed differences, the overtaking maneuver is distributed over multiple optimizations runs along the horizon. The two fellow cars are traveling on their respective lane with a constant speed of $3\,\mathrm{m/s}$. The time window for overtaking is chosen relatively small. The ego



(a) Initial configuration at t=5 s

(b) Overtaking initialization at t=13 s

(c) Overtaking scene with all trajectories densely interacting at t=18 s

(d) At t=24 s, the overtaking is finished and all agents are very close.

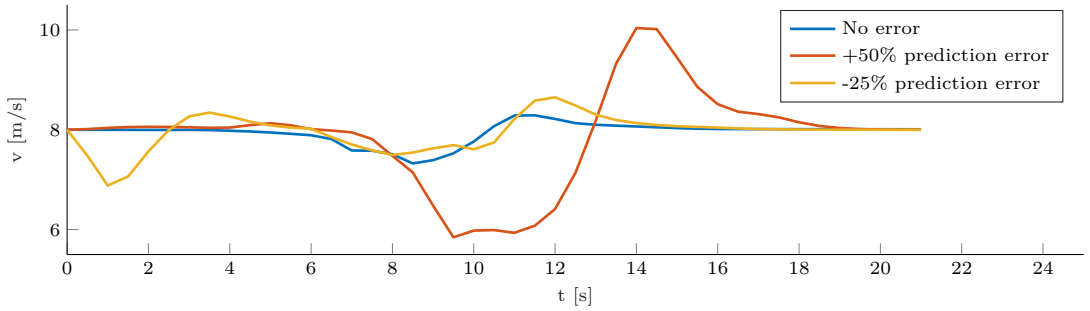
Figure 5.23: Evolution of an overtaking scenario with oncoming traffic with exact prediction models and a multi-agent formulation. The ego agent (blue) follows the trajectory computed by MINIVAN, the fellow agents (red, green) follow a fixed trajectory

vehicle has a desired speed of 8 m/s. We chose a horizon length of $N = 10$ points with a timestep of 0.5 s. We consider two settings in this experiment. First, the two other cars are modeled as dynamic obstacles, second, as interacting agents.

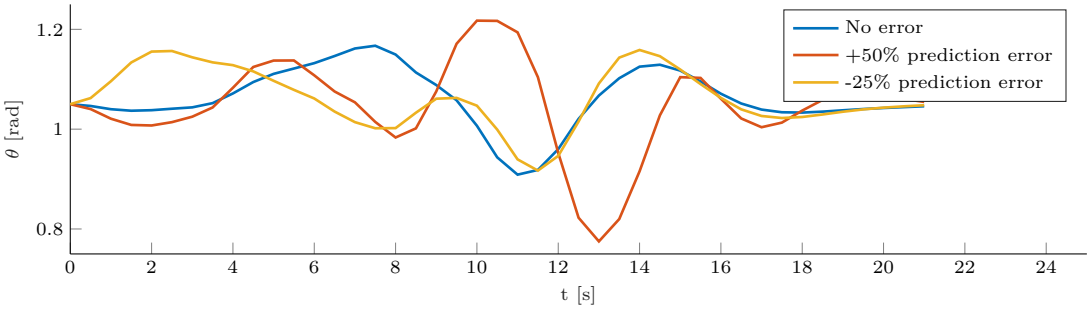
Robustness to Prediction Errors

In both settings, we vary the quality of the prediction. The current and future velocity of other vehicles can only be estimated with uncertainty. We model this speed estimation error by predicting both vehicles x% slower or faster than their actual speed. This prediction error is compensated by the slack terms that soften either the obstacle avoidance or the agent-to-agent collision constraints and can be interpreted as an additional, soft safety distance. With only hard constraints, even small prediction errors can yield crashes or infeasible optimization problems when the other vehicles behave very differently than planned.

If the velocity of the other vehicles is estimated too fast, the slack variables can compensate for this effect. The ego vehicle has to accelerate less than expected to catch up with the previous vehicle. Also, using up the slack, the ego vehicle can maneuver into the gap to overtake, and the gap closes slower than estimated. The other case that the velocity of the other vehicles is estimated



(a) Velocity v in the overtaking scenario.



(b) Orientation θ in the overtaking scenario.

Figure 5.24: Velocity and orientation plot over time of the multi-agent setting of the overtaking scenario with varying prediction errors. The resulting motion differs significantly with prediction errors present.

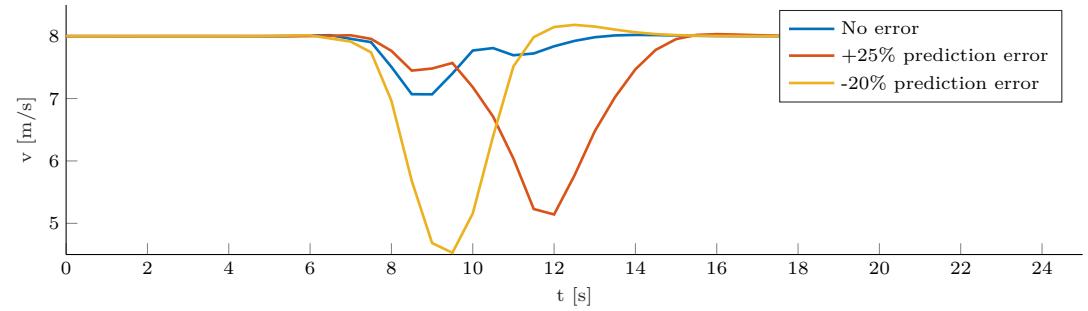
as being too slow is a security issue. The ego vehicle starts overtaking in the belief that sufficient space is available for overtaking, but the gap closes faster than expected.

We comparably parameterize both experiments, the single-agent setting with dynamic obstacles and the multi-agent setting. Still, the problem formulation differs, e.g., in modeling the agent shape (set of circles vs. rectangle) or the formulation of the slack terms. Therefore, we do not show a direct quantitative comparison. Also, we did not tune the cost parameters for maximum performance in this scenario but chose a generic setting.

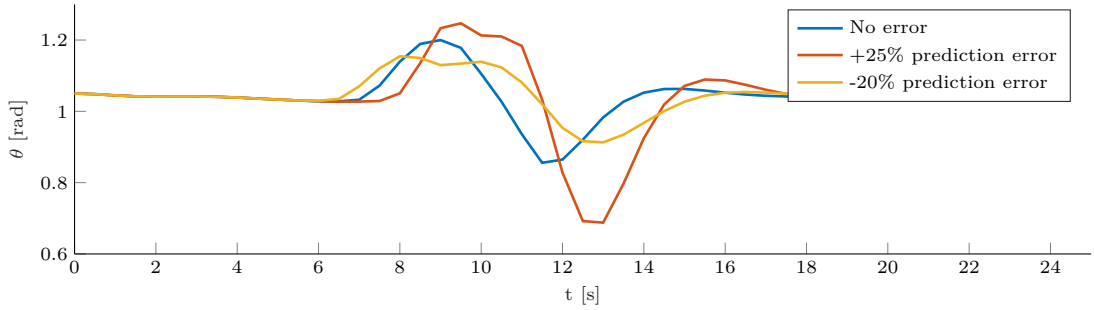
In the multi-agent setting, the cooperation factor λ_{ego} has to be chosen small, as the other vehicles show no reaction to the ego vehicle. A bigger cooperation factor results in scenarios with immanent collisions as the ego vehicle expects a, to some extent, cooperative reaction from the other vehicles, which they do not show.

Figure 5.23 shows an exemplary evolution of the scenario with the multi-agent formulation with an exact prediction without errors. The ego vehicle (blue) intends to overtake the red vehicle with an oncoming (green) vehicle. The blue ego trajectory shows the behavior and motion plan of the ego vehicle. The visualized future trajectories for the fellow vehicles are the estimations the ego vehicle bases its motion on. In the multi-agent setting, MINIVAN is able to find a collision-free solution in a wide range of prediction errors. Up to 25 %, negative error of the speed prediction can be compensated. With a speed prediction deviating more, the ego agent provokes an imminent collision when driving into the gap. Predictions faster than the actual vehicle speed can even be compensated up to speed errors of 50 %. If we let the error grow more, the ego vehicle accelerates too fast when driving on the left lane while overtaking as it expects the oncoming (green) vehicle to be faster than it is, resulting in an unavoidable collision with one of the other vehicles.

The error range is significantly smaller when both other vehicles are modeled as dynamic agents. Here the error in the prediction speed may vary between -20 % and 25 %. If the prediction is too slow, the ego vehicle enters the gap to overtake but cannot leave it fast enough with the



(a) Velocity v in the overtaking scenario.



(b) Orientation θ in the overtaking scenario.

Figure 5.25: Velocity and orientation plot over time in the dynamic obstacles setting of the overtaking scenario with varying prediction errors. With prediction errors present, a less smooth overtaking maneuver is executed.

other vehicles approaching faster than expected. If the prediction is too fast, it does not result in a collision, but a passive behavior of the ego vehicle as the gap is expected to be too small to overtake. The ego vehicle cannot successfully perform the overtaking maneuver and stays behind the (red) vehicle on the right lane.

Note that these maximum error bounds change by parameter variation, but the tendency stays the same. When planning a cooperative multi-agent motion, errors are compensated better, and even with agents showing non-cooperative behavior, better overall performance is achieved.

Smoothness of the Motion

Also, the continuity of the motion is influenced by the prediction error. In Figure 5.24 and Figure 5.25, we plot the ego vehicle velocity v and orientation θ over time in the multi-agent and the dynamic obstacles scenario respectively. The scenario without prediction error and the extreme cases are shown. In both cases, the velocity and orientation changes are higher with prediction errors present, as expected. With a too slow velocity estimation, the overtaking maneuver is started earlier, with a too fast estimation later than at the ideal point in time. Overtaking is finished at a comparable time. The magnitude of control inputs resulting in speed and orientation change is significantly higher in the dynamic obstacles scenario. This observation is expected, as the inaccurate velocity prediction is not compensated by planning a cooperative motion. Even though initially the distance between the ego vehicle and the other vehicles is high at $t = 0\,\mathrm{s}$ and the reference lines do not intersect, the other vehicles already influence the planning of the ego agent, as we can observe in Figure 5.24. From the beginning, steering and acceleration commands are triggered, a rather undesired side effect of multi-agent planning.

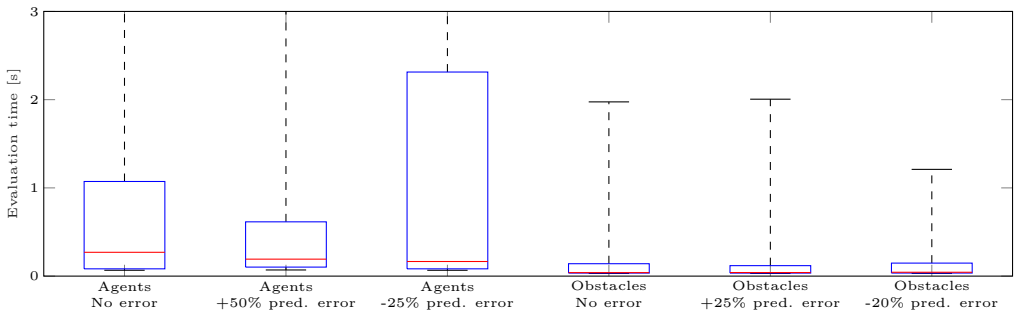
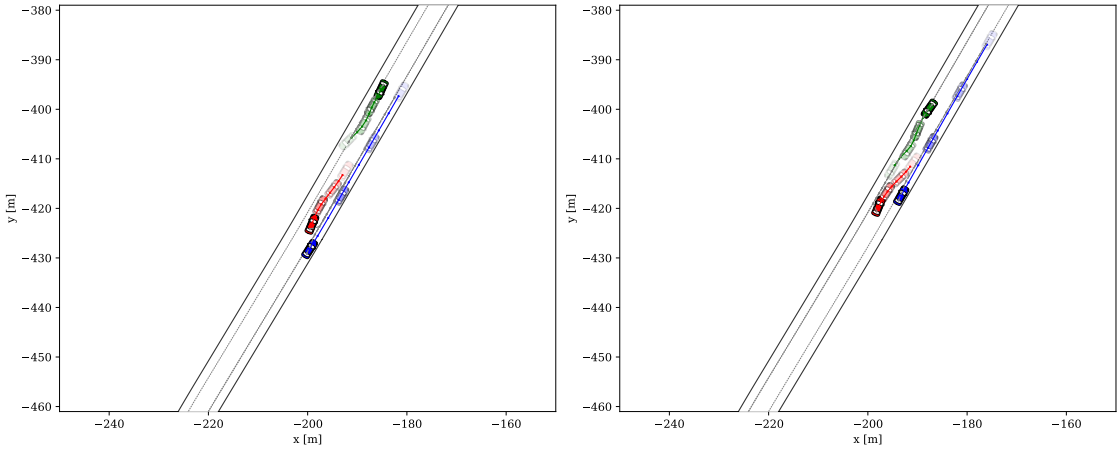


Figure 5.26: Evaluation time in different settings of the overtaking scenario. The mean runtime (red) stays below the real-time bound of 0.5 s, when modeling vehicles as dynamics obstacles also the 75th percentile. We observe outliers with a significantly higher solution time.



(a) The red vehicle steers to the lane center at t=9 s. (b) The blue vehicle overtakes on the right at t=10.5 s.

Figure 5.27: Evolution of the overtaking scenario with cooperative vehicles that perform one coordinated behavior plan. For an initial configuration of the scene see Figure 5.23a.

Runtime Considerations

We did not tune the parameters for real-time execution in this scenario. In Figure 5.26, we provide an analysis of the overall evaluation time per step, averaged over the whole scenario. We did not apply an upper time limit for the optimization for this evaluation. The time horizon is chosen as 5 s with a step size of 0.5 s. We observe that the mean runtime is below 0.5 s but has a high variance. In the dynamic obstacles scenarios, the 75th percentile is also far below 0.5 s, but with a significant number of outliers. These outliers are critical for real-time execution but in a tractable range of at most 2 s. The evaluation time does barely varies with the prediction error. When modeling the other vehicles as interacting agents, we observe extreme outliers > 5 s in the runtime where terminating the optimization earlier yields a significantly worse solution. With a mean runtime lower than 0.5 s, the 75th percentile clearly violates the real-time bound of 0.5 s. This makes real-time application challenging, and we consider three interacting agents as a maximum. The effect of the prediction error does not show a clear tendency. We generally observe that in dense scenarios with various driving options, the evaluation time rises, which is the case when underestimating the other vehicles' velocity.

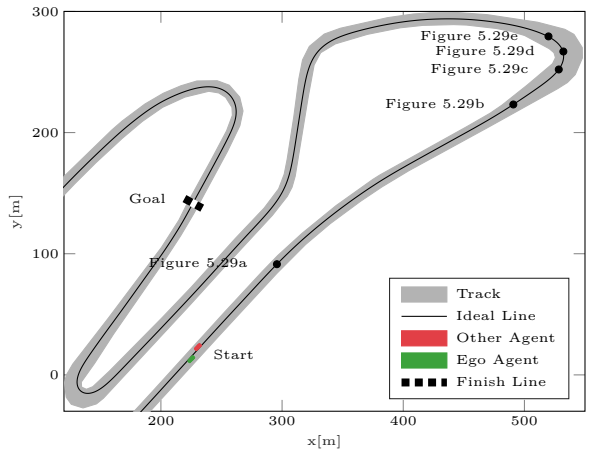


Figure 5.28: Setup of the racing scenario. The start points of scenes depicted in the other figures are located on the track. (graphic from [KEK20], ©2020 IEEE)

Coordinated Maneuver

As a concluding experiment, we model the scenario with three coordinated vehicles that follow the same coordinated behavior plan. We set the cooperation factors λ equal for all vehicles to generate a balanced solution. As we depict in Figure 5.27, a different solution is found where two vehicles perform a slight evasive maneuver to enable an overtaking with moderate acceleration and steering inputs. In this specific scene, the overtaking is finished 11.5 s after the scenario start, which is less than 12.5 s to 13.5 s in the previous scenarios without explicit cooperation. We observe that the motion of the oncoming (green) vehicle is slightly non-smooth. This is an effect of terminating the solver before reaching the global optimal solution according to the parameter setting for real-time applicability.

Conclusion

We showed and analyzed the evolution of a dense overtaking scenario in three different simulations: A multi-agent setting with (1) uncontrolled agents, (2) three coordinated agents, and (3) in a single-agent setting with the other cars modeled as dynamic obstacles whose trajectories are not varied by the optimizer. In all three cases, MINIVAN can find feasible trajectories with comparable performance. The scenario is finished faster with a coordinated solution while keeping comfort and safety limits. This observation indicates that MINIVAN not just by chance found good solutions. We conclude that both the multi-agent formulation and the formulation using dynamic obstacles can solve a dense overtaking scenario by investing more control input to avoid collisions. The multi-agent formulation is robust to bigger prediction errors. However, it has a significantly higher variance in the evaluation time that makes real-time applicability questionable for three or more agents.

5.7.5 Competitive Autonomous Racing

We in this section analyze both the implications of the introduced cooperation factor λ in the joint cost function and the agent-to-agent collision constraint, including the soft constraint term to account for model inaccuracies in two simulated scenarios. We use a racing track by Heilmeier et al. [Hei+20] with the provided ideal line and track boundaries (Figure 5.28). We define the finish line after 1177 m.

Scenario Setup

We evaluate our algorithm in an autonomous vehicle racing scenario, in which dense interactions with other agents usually occur. The goals of each agent are conflicting as each agent wants to win the race. Besides the observations of other agents, no communication is involved. We assume the optimal line on the racetrack to be known and equal for all agents. Therefore, each agent competes to stay on this ideal line, which we use as a reference path. In the curves, we limit the maximum velocity with respect to the maximum lateral acceleration possible for the vehicle model.

The ego agent A^1 starts with a disadvantage but has a slightly higher top speed of 14 m/s than the other agent A^2 with a top speed of 12 m/s, so it can eventually overtake and win the race. The ego agent A^1 uses MINIVAN to plan its behavior and trajectory. In contrast, the other agent A^2 only tracks the reference line with respect to its kinematic constraints without considering the ego agent. Reference tracking is favored over driving smoothness. We plan a horizon with 20 steps and a time interval $\Delta t = 0.2\,\text{s}$.

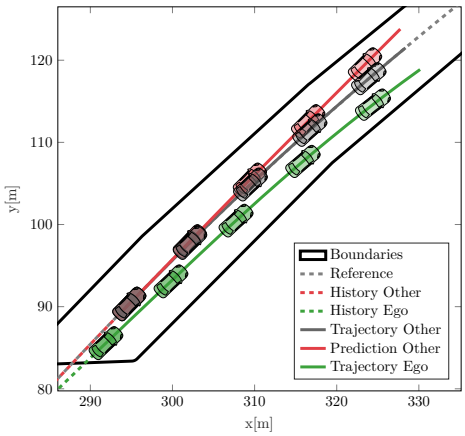
Effect of the Cooperation Factor

In the overtaking scene in Figure 5.29, we show exemplary for $\lambda^1 = 0.3$ how A^1 finally makes use of its higher top speed. When not leading to ambitiousness, we will, in this section, for the ease of reading, drop the agent superscript 1 for the cooperation factor meaning $\lambda = \lambda^1$. We observe, that even with a very inaccurate model of A^2 (Figure 5.29a), A^1 can safely catch up (Figure 5.29b), finally overtake (Figure 5.29c), and keep the leading position even though it has to take a wider curve due to its higher speed (Figure 5.29d, Figure 5.29e).

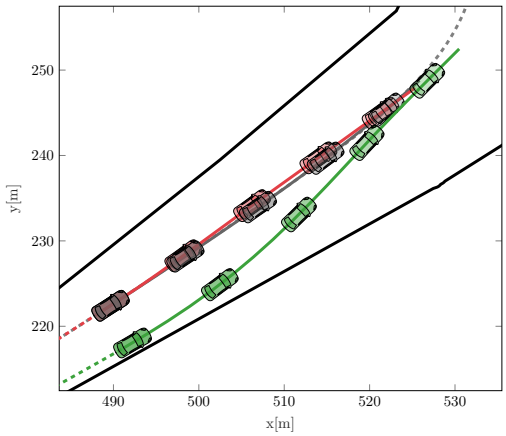
In Figure 5.30, we analyze how different scaling factors λ lead to different agent behavior by tracking the lap time of both agents. As a baseline, we simulate each agent alone on the racetrack. If A^1 ignores the predicted motion and intention of A^2 ($\lambda > 0.85$), it drives too aggressive and eventually provokes a crash of both vehicles.In the other extreme ($\lambda < 0.1$), A^1 ignores its own goal, which leads to very passive behavior. Even though it could accelerate to a higher top speed, it cannot successfully overtake. We observe that scaling factors in between depending on the λ, A^1 sooner or later successfully overtakes and wins the race. Hence, with a balanced scaling factor, A^1 can achieve its own goal of driving at a higher top speed, also considering the intent of A^2 to stay on the ideal line at a lower speed.

Effect of the Soft Collision-Avoidance Constraints

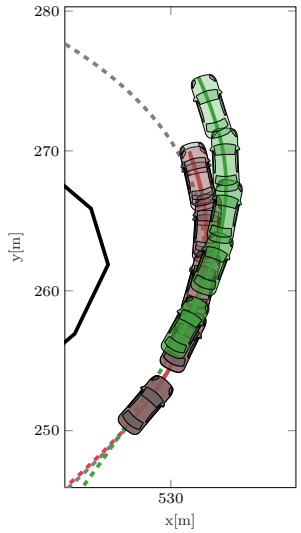
In Figure 5.31, we analyze the effect of the scaling factor λ on the interaction of the agents and the safety distance the ego agent A^1 is willing to keep. If A^1 behaves aggressively, it implicitly models that A^2 will give way, which it will not. This results in a high prediction error for high values of λ, which is compensated by the slack terms. We observe that more slack is used to mitigate imminent collisions at the beginning of the planning horizon (Figure 5.31a). At the end of the horizon, the magnitude of slack used is lower as the optimizer has cheaper alternatives to avoid collisions (changing position or jerk) even though the absolute errors from a non-ideal prediction are higher (Figure 5.31c). When both vehicles interact densely, the ego agent uses the slack to violate the soft safety distance to some extent. As expected, the amount is higher for high values of λ. We show three different values of λ representing three different behaviors of A^1; early overtaking ($\lambda = 0.8$, until t=17 s), late overtaking ($\lambda = 0.3$, until t=40 s, also see Figure 5.29), and no overtaking ($\lambda = 0.05$). As soon as the overtaking maneuver has been performed successfully, the slack costs go to zero, as the safety distance can trivially be fulfilled. In the edge case $\lambda = 0.05$, A^1 behaves very passive and always tries to keep the safety distance big. In the other extreme of $\lambda = 0.9$, A^1 behaves too aggressively, accepts a high prediction error that the slack terms cannot compensate, and finally provokes a crash. For $\lambda = 0.8$, A^1 performs an aggressive but still safe overtaking maneuver, whereas for $\lambda = 0.3$, it initiates the overtaking late as more drivable



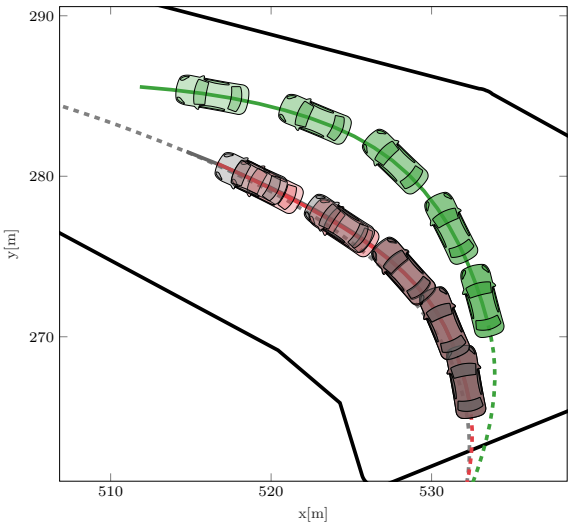
(a) $t = 9\,\mathrm{s}$ after start, the ego agent initiates overtaking.



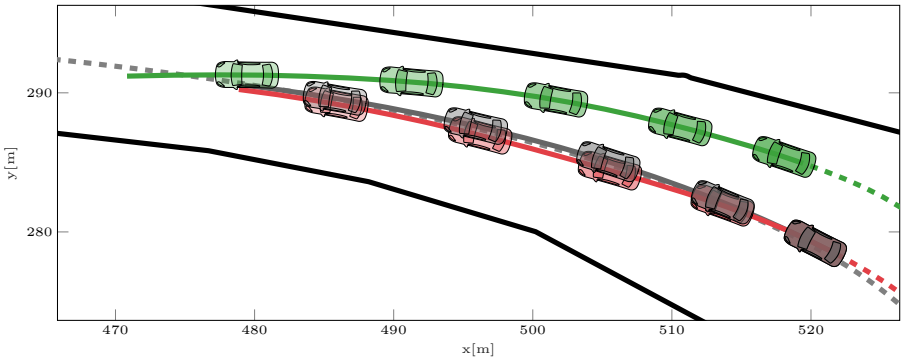
(b) Ego plans the first trajectory to be in front of the other agent at $t = 27\,\mathrm{s}$.



(c) Ego performs a successful overtaking at $t = 31\,\mathrm{s}$.



(d) Due to the higher speed, ego has to take w wider curve at $t = 34\,\mathrm{s}$.



(e) Ego plans the trajectory finally resulting in the leading position at $t = 37\,\mathrm{s}$.

Figure 5.29: The ego agent A^1 overtaking in the first curve with cooperation factor $\lambda = 0.3$. (modified graphic from [KEK20], ©2020 IEEE)

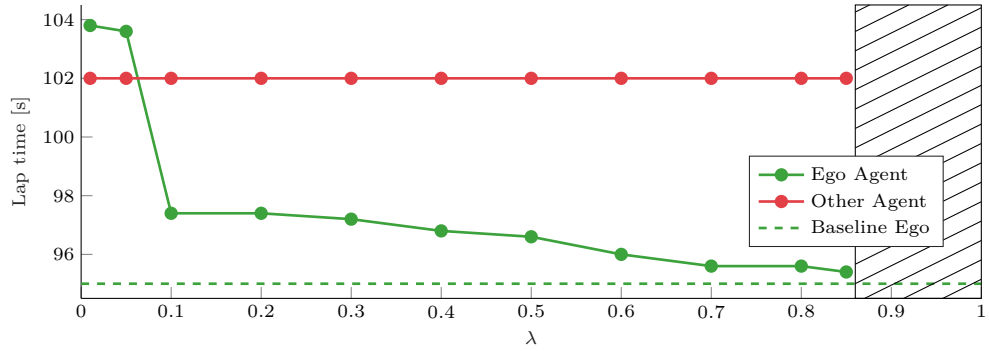
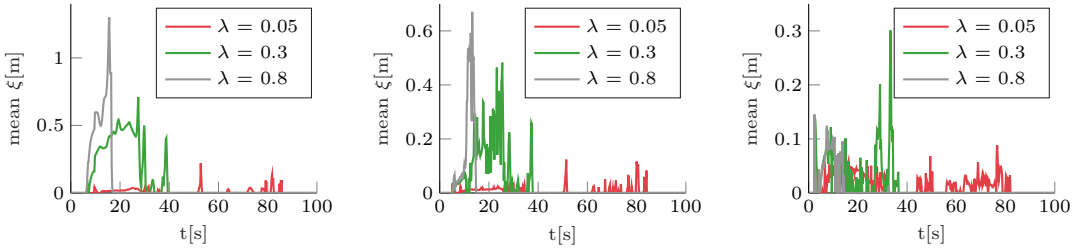


Figure 5.30: How can the ego agent A^1 win the race? A^1 incorporates the motion of A^2. It looses the race if parameterized too conservative ($\lambda < 0.1$) and overtakes the sooner the more aggressive λ is chosen. If the interests of A^2 are ignored, eventually a crash occurs while overtaking (hatched area, $\lambda > 0.85$). (graphic from [KEK20], ©2020 IEEE)



(a) Horizon index $k = 2\,\mathrm{s}$ (b) Horizon index $k = 10\,\mathrm{s}$ (c) Horizon index $k = 20\,\mathrm{s}$

Figure 5.31: The effect of soft constraints coping with prediction errors. At the respective horizon index, we depict the mean safety distance compensated by the slack variables ξ for different factors λ for all planning instances. (graphic from [KEK20], ©2020 IEEE)

area is available. Hence, with appropriate parameter selection, our algorithm can cope with not completely known cost functions of other agents, and the driving style of the ego agent, including its willingness to take risks can be adopted.

Conclusion

We conclude that even in a purely competitive setting, such as a car race, MINIVAN can, with appropriate parameterization, find a safe solution while achieving its own goal. It also takes the intent to stay on the ideal line of the fellow vehicle into account, which shows the cooperative design of the planner. The cooperation factor λ effectively leverages the agent behavior at a local scope (as we showed in Section 5.7.2) and also globally viewed over various receding horizon steps.

5.8 Conclusion

This chapter introduced the MINIVAN planning approach that solves the multi-agent behavior planning problem by formulating a MIQP using a continuous state and action space. Using an orientation-dependent linearization approach, we derive a suitable vehicle model obeying the non-holonomy of a road vehicle. It also over-approximates the vehicle shape for collision checking with arbitrary objects and agents. By introducing soft constraints, we ensure robustness against

perception and prediction errors. A cooperation factor in the joint cost function effectively leverages the levels of cooperative behavior. We show both in simulated scenarios.

The approach is scenario-agnostic; no theoretical limitations on the environment, obstacle, or agent configuration applies. With its continuous action formulation, it can even find a solution in dense and narrow situations, as we demonstrated in a racetrack scenario. It produces kinematically, and dynamically valid trajectories that inherently follow a coordinated strategy as the intents of all agents are taken into account in the leveraged global cost optimization. A practical drawback is the invalidity of the model at low speeds. This impedes the application in interactive scenarios, where it is for at least one agent necessary to (nearly) come to a complete stop.

The cost function effectively scales the level of cooperation between different agents. As human drivers, each autonomous agent can balance the own and others' interests to achieve different behavior from altruistic to egotistic. Nevertheless, the behavior is not human-like as the optimization does not take traffic rules and social conventions into account but finds a globally optimal solution with respect to driving smoothness and deviation from the intended reference track. Interaction with uncontrolled agents is still possible, either by considering these as dynamic obstacles whose motion cannot be changed, resulting in reactive ego behavior or by using a soft constraint agent model formulation that handles inaccuracies in fellow agents' motion and behavior prediction.

Naively executing the model with an off-the-shelf solver also in simple scenarios violates real-time bounds. We showed that using the previous solutions from the receding horizon planning problem as a warmstart solution, a significant decrease in the runtime is achieved. Also, adapting the solver setting and constraint evaluations to the model structure improves the runtime. In single-agent scenarios, a sufficiently high number of (static and dynamic) obstacles can be handled in real-time. Complicated receding horizon instances where the topology of the optimization problem changes can yield critical peaks in the runtime that have to be reduced by effective parameterization. Depending on the overall scenario complexity, multi-agent scenarios can be possible in real-time with few agents. However, the runtime is significantly higher than in single-agent scenarios with dynamic agents resulting in runtime peaks that violate real-time conditions.

6 The Software Stack for a Behavior Planning Research Vehicle

Executive Summary of this Chapter This chapter will elaborate on how and why we selected and designed the hard- and software of the experimental platform for fully autonomous test drives on public roads. We discuss why it is valuable to base the software stack on the existing open-source solution Apollo to concentrate on the behavior and motion planning research. Besides parameterization and new device drivers, only a small number of additional components were necessary to run Apollo on a German experimental vehicle, e.g., developing an adapter to a standard map format. We, in detail, explain how to integrate Mixed Integer Interactive Planning (MINIVAN) into the existing software architecture and interfaces of the stack. It required an efficient convexification algorithm for arbitrary road environments and a Model-Predictive Control (MPC)-based post-optimization to account for low speeds and efficient subsampling of trajectories. Due to the modular software structure, these features are also available for other (planning) modules. The implementation of the trajectory tracking control on a real-time rapid prototyping control unit provided additional stability, still maintaining the architectural patterns of Apollo. This chapter further elaborates on the strengths and weaknesses of our hard- and software setup and the suitability of the Apollo software stack. Apollo provided a good baseline performance of all functional modules, and especially the core functionalities, such as the communication middleware, works smooth and stable. It comes at the price of a relatively closed ecosystem and a restricted C++ development environment. We conclude this chapter by evaluating three on-road, closed-loop autonomously driven scenarios. In these, we quantitatively demonstrate the real-time capability and overall performance of the behavior planner operating inside the autonomous driving stack. The contributed code of this thesis is available as open-source software.

Content and Structure of this Chapter In this chapter, we will first in detail describe the hardware setup of the experimental platform in Section 6.1. We then describe the software setup that is based on the open-source platform Apollo in Section 6.2, putting a particular focus on the changes we introduced in the existing stack. In the third part of the chapter, Section 6.3, we describe how we integrated MINIVAN into the Apollo stack. Section 6.4 describes how the trajectory tracking approach we implemented. In Section 6.5, we elaborate on our experience with this setup in on-road driving experiments, described in Section 6.6. Section 6.7 concludes this chapter.

Contributions of this Thesis Parts of this chapter have already been published in [Kes+19] together with further colleges from the research institute fortiss, especially Martin Büchel. This applies to the hardware setup and an earlier version of the trajectory tracking controller. The integration of MINIVAN into Apollo, the adaption of the software stack for our research vehicle, and the driving experiments are based on a joint work together with Klemens Esterle [KEK22].

6.1 Hardware Setup of the Prototype Vehicle Fortuna

This section will describe the hardware setup of the research vehicle Fortuna. To meet the requirements of rapid prototyping, we chose to modify a production vehicle with non-production-

Figure 6.1: The Fortuna autonomous driving vehicle experimental platform: a modified VW Passat with lidar sensors and antennas on the roof rack, additional cameras inside the vehicle and additional radar sensors integrated in the bumpers. (graphic from [KK19], ©2019 IEEE)

vehicle hardware and provide access to production sensors and actuators. We use a refitted 2018 Volkswagen Passat Variant GTE (see Figure 6.1). This setup leverages the need for innovative, flexible, and powerful hardware and the need for a standardized and reliable base hardware setup. We did not aim for a hardware setup of a production vehicle in terms of redundancy or power consumption.

An architecture overview of Fortuna's additional hardware is depicted in Figure 6.2. Figure 6.3 provides an impression of the installed setup in the trunk. The modifications include additional sensors, interfaces to access production vehicle bus networks, and four computers connected via an industrial gigabit Ethernet switch splitting the traffic into several virtual networks.

- One industry standard real-time rapid-prototyping control unit (a dSpace Micro Autobox II) with an IBM Power PC 900MHz CPU and 16MB RAM for control algorithms with various low-latency hardware interfaces (including CAN) running a real-time operating system. We run the low-level trajectory control on this computer as described in Section 6.4.
- Two Car PCs with Intel i7 3.4GHz quad-core CPU and 32GB RAM for sensor data processing, motion planning, human-machine interfaces and further software components running Ubuntu Linux. These PCs run the driving stack as described in Section 6.2.
- One Nvidia Drive PX 2 AutoChauffeur with two Pascal GPUs running Nvidia Drive Works for accessing camera images.

The hardware setup includes a 12V backup battery with additional power management and a connection to the high-voltage system of the production vehicle. A prototype cellular 5G interface realizes Vehicle-to-Everything (V2X) connectivity. Proprietary gateways enable access to the CAN buses of the production vehicle, which allows reading sensor and vehicle information. Write access enables automated driving through steer-by-wire. Key switches are installed for safety reasons and allow to power and enable the reading access measurement system and to enable writing Car Area Network (CAN) access for longitudinal and lateral control. An emergency shutdown button

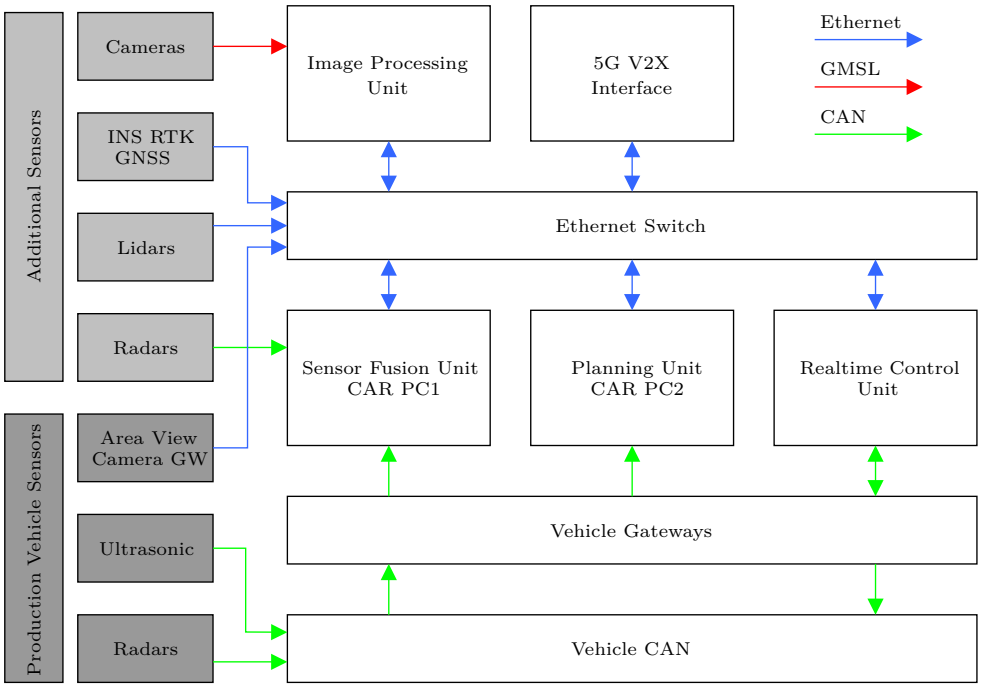


Figure 6.2: Schematic overview of the hardware setup and the interfaces between the components (modified graphic from [Kes+19], ©2019 IEEE)

allows the safety driver to return to a production vehicle mode. The production vehicle sensor data include object lists detected by the radar sensors of the Adaptive Cruise Control (ACC) system and kinematic vehicle state information. Also, raw data from the ultrasonic sensors and camera images from the Area View surround-view cameras are available. More in detail, the vehicle is equipped with the following additional sensors, cf. Figure 6.1, allowing a 360° Field of View (FOV) avoiding blind spots.

- Three Velodyne lidars: one VLP-32C with 32 layers in a central horizontal position on the rooftop and two VLP-16 with 16 layers at each side of the vehicle roof, inclined to scan the areas at each side of the vehicle. We regard lidar sensors as mandatory for automated driving. The setup was chosen to provide a sound point cloud density in combination with a sufficiently broad sensor range for various scenarios.

- Five Sekonix cameras: two front-facing cameras, one with 60° FOV, and one with 120° FOV, one camera to each side and one rear camera, all with 120° FOV.

- Four Smartmicro UMRR-146 radars: two facing forwards and two backward, integrated into the bumpers with access to raw sensor data.

- Inertial Navigation System (INS): iMAR iNAT FSSG-1, a fiber optic gyro (FOG) based INS supporting Real-Time Kinematic (RTK) with integrated Global Navigation Satellite System (GNSS) receiver offering a localization precision of up to 2cm. The device can serve as a positioning unit and also as high precision reference localization for algorithm validation. As we consider a very high and reproducible localization measurement as essential for benchmarking autonomous driving functions, we decided on this industry-standard but non-automotive production grade device.

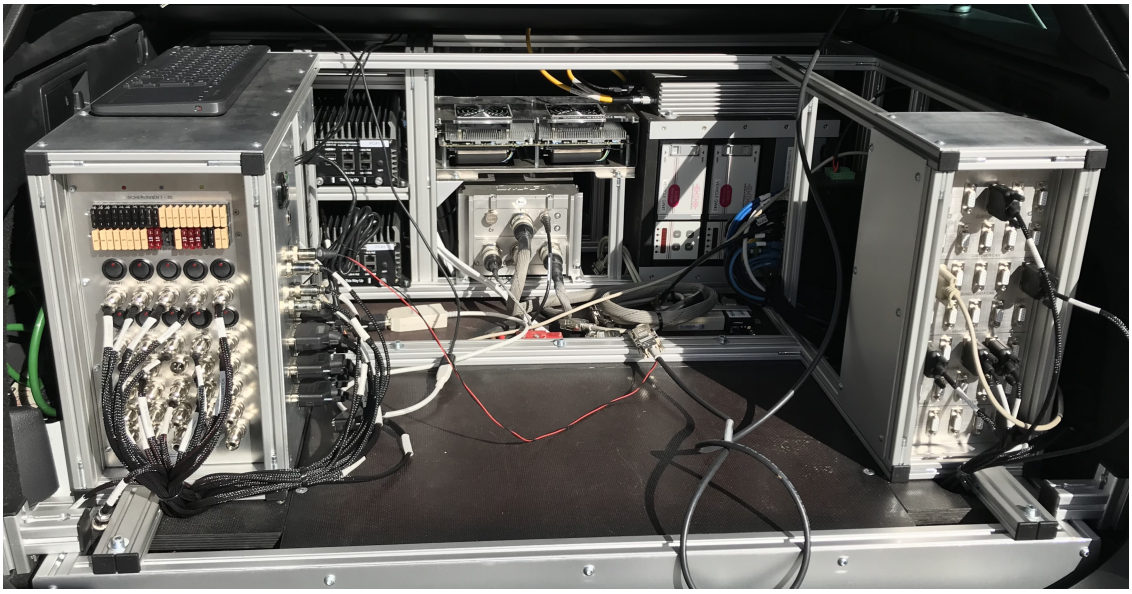


Figure 6.3: The additional hardware installed in the trunk showing (clockwise) the power supply, the two car PCs, the Drive PX 2, the iNat FSSG, the Micro Autobox II, the CAN gateways, the Ethernet switch, and the CAN patch panel.

In contrast to the Apollo reference vehicle [Bai17] or the Bertha vehicle [Tas+18], we chose to equip the vehicle with more than one computer to separate functionality. This setup comes at the price of network communication overhead and necessary design decisions on how to connect sensors and devices. For example, we connect the 360° FOV camera setup to the Drive PX 2 hardware via Gigabit Multimedia Serial Link (GMSL). This wiring hinders us from running image-based perception algorithms directly on either of the PCs. Further details on the hardware components and the requirements leading to the selection of the components are given by Buechel et al. [Bue+19].

6.2 Usage and Introduced Adaptions to the Apollo Driving Stack

We decided to use Apollo as a basic software stack, mainly because when this work started (in the year 2018), Apollo was more mature than Autoware. Apollo already supported on-road driving and is based on a standardized map format. We have since updated our system to Apollo v5.5. This section focuses on the extensions we implemented to run Apollo on our vehicle. Our extensions are also available as open-source software [EK21]. Apollo provides a variety of prebuilt functionality and a well-structured codebase. We aimed to introduce minimal changes to the Apollo stack to maintain the original functionality. We modified the sensor drivers and controller module and developed several adapters and software components to connect the vehicle hardware with the Apollo stack. After briefly describing Apollo's functionality and discussing the changes mentioned above, we conclude the section by discussing the advantages and shortcomings of Apollo's stack.

6.2.1 System Overview

Apollo follows a classical sense-plan-act architecture pattern. Each component is started as a separate process, and the messages are exchanged among the components using Apollo's custom publish-subscribe middleware Cyber RT, as we describe in Section 3.6.1. This middleware makes

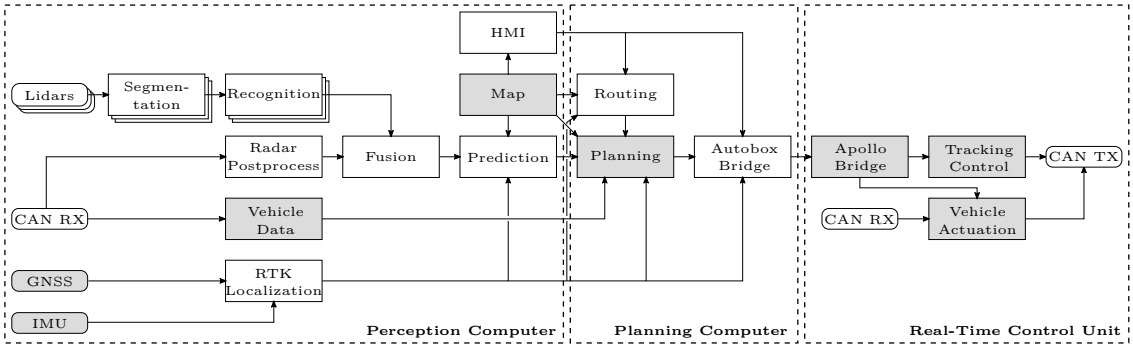


Figure 6.4: Architecture overview of the setup we chose to operate the vehicle. Driver components are visualized using smaller boxes with rounded corners. The components with main contributions of this work are shown in gray. The software is distributed on three hardware platforms. (graphic from [KEK22], ©2022 IEEE)

the integration of customized components straightforward. We followed Apollo's architecture pattern as sketched in Figure 6.4 but replaced some components with custom implementations. We also reflected the architecture by separating each architectural layer on an individual hardware computation component. For perception and planning, we used two standard in-vehicle computers. The most significant architectural change is the separation of the vehicle controller from Apollo, placing the control layer on a dedicated rapid prototyping real-time control unit, see also Section 6.1.

6.2.2 Perception Pipeline and Obstacle Motion Prediction

Our sensor setup is different from the recommended Apollo hardware setup, which led us to implement an adapted perception pipeline. As we focus on the planning and control layer, we only used a minimal set of sensors, mainly the central 32-layer lidar of type Velodyne VLP-32C, and the differential GNSS system for localization. The radar and vision pipelines are not used in this work. The sensor data fusion component thus only post-processes the objects detected from lidar.

Ego Localization Since we use different hardware for localization than the Apollo reference vehicle, we had to develop a customized localization adapter. As the INS/GNSS is supposed to provide highly accurate and consistent measurements, we decided to feed an already filtered GNSS position and velocity plus the Inertial Measurement Unit (IMU) data from the same device into the stack without changing Apollo's standard messages. Doing so, we did not introduce changes in the build-in RTK localization module that combines GNSS and IMU messages to a consistent localization message and provides the dynamic transformation from the world frame to the vehicle reference frame.

Radar Pipeline From the production vehicle's ACC system, the radar sensor data input is available. Production vehicle radars are optimized for ACC and emergency brake scenarios. Therefore, these radars are parameterized to suppress false positives in order to avoid unmotivated emergency braking maneuvers and are well suited to reliably detect vehicles around the ego vehicle. We first read these information from the CAN bus using the adapted Apollo node. We then extract the relevant objects (if any) from the processed CAN message and translate them into Apollo's object list format to be fed into the sensor data fusion component. Note that not every field of Apollo's message can be extracted from the data, e.g., the object dimensions. We here chose to input conservative bound values.

Lidar Pipeline Apollo provides a pipeline starting from the lidar drivers for the identification of objects in the pointcloud [Bai21c; Bai21e]. The vehicle's motion during the point cloud scan is compensated by comparing the first and last scan point and applying an appropriate translation and rotation into the ego vehicle frame. A segmentation component creates objects from the point cloud data. It projects all scan points into the two-dimensional Cartesian plane and quantizes them into a grid of cells. For each cell, statistical measures (features) such as the number of points per cell, intensity, or height are computed. This tensor of 2D grid $\times$ features is fed into a fully convolutional neural network that predicts center offset, objectness, positiveness, object height, and class probability per cell. From this, a directed graph of all cells is generated, and connected cells are clustered as objects. In a final post-processing step, improbable and small clusters are removed. These clusters are then wrapped into polygonal objects with a valid orientation. Whereas the networks' implementation is provided, the training data for the neural nets are not available open-source. Our setup relied on the trained network for a 64-layer Velodyne lidar to process the data of the 32-layer Velodyne lidar, which implies an adaption in the segmentation process that is hard to verify. A recognition component tracks the segmented obstacles and associates each obstacle with a track in a subsequent step. The main task is the creation and temporal association of tracks and the dropping of implausible tracks. The detection-to-track association is reduced to a bipartite graph matching problem and solved using the Hungarian algorithm. A Kalman filter with some robustness tweaks is then used to estimate the motion states for each track, such as position and velocity. The result is a list of objects with kinematic states, shapes, and types per lidar sensor. We only changed the parameterization of the lidar pipeline but no implementations.

Sensor Data Fusion The fusion component [Bai21c] takes the object lists from each individual sensor and creates one consistent list of objects in Apollo's target format. One main sensor is used to trigger the fusion, in our case, the central lidar scanner. Data from all other channels is cached. The association problem of individual sensor objects to the set of fused objects is again reduced to a Bipartite Graph Matching problem and solved using the Hungarian algorithm with the goal of minimizing the Euclidean distance of the object center points. Once the tracks from the sensors are matched to fused objects, the association is done using the sensor object ids. An Adaptive Kalman Filter fuses the motion states and uncertainty information for each sensor object. The Adaptive Kalman Filter is a strategy to adjust to dynamics that are not modeled in the process model.

Prediction We also did not introduce implementation changes to the prediction component [Bai21f]. It takes the objects from the sensor data fusion and generates an estimated future motion based on the object state and type and the ego-motion. The prediction operates in two scenarios: driving on a lane (cruise) and junctions. Different algorithms are executed according to the type of object and its proximity/probability to interact with the ego vehicle. First, an evaluator predicts path and velocity individually for each object. Second, a predictor generates trajectories for each object.

For nearby vehicles, the model is based on a convolutional long short-term memory network with extrapolation to longer trajectories. A deep neural network-based evaluator combined with a predictor moving the objects along the associated lanes is used for vehicles further away. Cyclists are predicted to stay driving in their respective lanes. The evaluator for pedestrians is based on the Social LSTM approach [Ala+16] combined with a free motion predictor. For unknown objects, an adapted version of the multilayer perceptron evaluator and the lane-keeping predictor is used. Objects outside any lane may move freely.

6.2.3 Further and Auxiliary Components

We had to use, change, or extend various components to make Apollo functional on our Passat GTE. This section will briefly give an overview of these auxiliary modules and specify which changes we introduced and which components could be used out-of-the-box.

Control We require the trajectory tracking controller to stabilize a given trajectory and output acceleration, and steering commands passed to the vehicle interfaces. Although Apollo would serve this need, we chose to replace the controller used by Apollo and run it on a separated real-time platform instead. This separation facilitates higher safety and reliability in our prototype setup by separating control tasks in real-time execution for other software applications on different hardware platforms. Therefore, we developed a bridge communicating between the Apollo stack and the trajectory tracking controller on the real-time system. We observed that this separation yields more stable system performance and optimally uses the different benefits of the modular computing hardware setup. In Section 6.4 we will describe in detail the design and implementation of the trajectory tracking controller.

Canbus The Canbus module reads and writes data from Apollo messages to CAN messages. It consists of two parts; a driver reading the raw CAN messages from a CAN hardware device (in our case a Socket CAN PCI card) and a module parsing the raw CAN messages and translating them into the appropriate Apollo message in Protobuf format (namely the chassis and chassis_detail message).

Apollo does not provide parsers to read messages from a Passat GTE but a hook to integrate new vehicles. We used this software interface to add CAN bus reading functionality and fill necessary Apollo messages such as motion and steering information. Also, we extended the standard message with data specific to our vehicle, such as the object information from the production vehicle ACC radar system. As we do not control our vehicle from a PC running Apollo, we did not implement writing CAN access for Apollo's CAN module.

Bridge Each Apollo module runs inside Apollo's Docker container. The bridge component makes it possible to communicate with modules running outside the container using Apollo's Protobuf-based messages. We use the bridge component to communicate between the PC running Apollo and the Autobox running the controller. We did not change or extend the bridge component as we natively decode Apollo's messages on the Autobox, as we will see in Section 6.4.2.

Routing To select a reference path on the High Definition Map (HD-Map), the map provides a topology graph of how the lanes are interconnected. The routing component selects the shortest path on the network using the A* algorithm [LaV06]. We did not introduce changes to this component. The integration into the HMI is solved conveniently and straightforwardly.

Map Generation and Conversion Apollo heavily relies on an HD-Map, which provides geometric information and a topology graph of how the lanes are interconnected. This map is stored in a customized format extending the OpenDrive format [ASA21]. While standard OpenDrive maps describe the geometric shape of a road and lane using mathematical functions, Apollo-style maps use linestrings instead, a discretized variant of the functions. We did not modify Apollo's map format but had to provide maps from our test area in Munich. We thus developed a converter that translates standard-conform OpenDrive maps into custom-Apollo OpenDrive maps. We did not adopt the map functionalities of Apollo itself but had to provide maps from our test areas in Munich.

The OpenDrive map format is a widely used industry-standardized format initially developed for simulation purposes. Representing all elements as mathematical functions comes with the benefit

of an interpretable file. However, for wheel contact point evaluation in simulation tools or other geometric map queries, most simulators (e.g., [Ber+20; Dos+17]) rely on sampled and discretized versions of these functions. These discretizations can be performed offline and stored once; online while evaluating only map lookups are needed then. Alternative map formats, such as Lanelet2 [Pog+18] directly rely on discretized linestrings.

Standard OpenDrive maps define a centerline of each lane as a sequence of functions (either line, spiral, or arc) along the road longitudinal s-coordinate. On top, lanes are then defined as functions defining the lateral t-offset from the centerline or the previous lane in a street-local st-coordinate system. Each road has a unique ID and, within each road, each lane has a unique ID. The linkage of roads is achieved by specifying successor and predecessor lanes on other roads. Also, junction elements can provide more sophisticated linkage information.

In Apollo's flavor of OpenDrive, each geometric function has to be provided additionally as a set of discretized points in an appropriate geographic coordinate system. Our converter performs this discretization into WGS84 (longitude/latitude) points and the respective associations. It resolves discretization errors at lane boundaries that can occur when transferring from the functional to a discretized representation as numerical errors in the sampling can lead to a non-continuous connection of two discretized functions. The converter further iterates through an input map and first converts the start point of a represented curve from the provided geographic coordinate system in the file header to WGS84. It then samples the mathematical representation of the curve using a given, fixed discretization distance. We treat some special cases, such as dropping too short segments or avoiding overlaps at neighboring segments. We sample the centerline and the left and right boundary of each lane. Also, lane linkages are implemented slightly different, which is resolved by the converter. Furthermore, Apollo-specific map content such as the area of an intersection is generated. After having generated the custom OpenDrive map in XML format, we use Apollo's toolchain to translate the map to a Protobuf-based internal map format. In this format, each lane is specified by its centerline and boundaries and a unique ID. Roads only contain topology information on how lanes are connected. Alongside, a downsampled version of the map is generated for visualization purposes and a graph-based representation of the map for routing purposes. The map converter is written in Python in contrast to the majority of code provided with this work. For a thorough analysis of the differences of standard OpenDrive and Apollo's custom format, we refer to Gran [Gra19].

Relying on OpenDrive maps as a base format is a reasonable choice as in the input format. On the other hand, the usage of a discretized and less complex data structures is preferable for online evaluation. Our provided map converter closes the existing gap in Apollo's map toolchain.

Kessler et al. [Kes+18] propose an alternative approach to generate a roadgraph for navigation and a polygonal representation of the available free space in unknown structured environments solely based on (lidar) sensor measurements. In such a scenario, we assemble a polygonal map representation without prior knowledge of the location. With a HD-Map available, this information is generally extracted from the map.

Visualization and HMI We use the provided web-based visualization and HMI tool Dreamview without changes. With the provided parametrization hooks, we introduced our vehicle configuration and adapted the visualization to our needs. The web-based architecture comes with the flexibility to view the content on various, also mobile devices.

Transformations Apollo offers the functionality to provide static and dynamic transformations between different coordinate systems. As a reference point, we chose the vehicle's rear axle center. We parameterize (sensor) components to operate in this reference frame whenever possible. For all others, we use the parameterized transformation chains. The code and usage are similar to the TF package in the Robot Operating System (ROS).

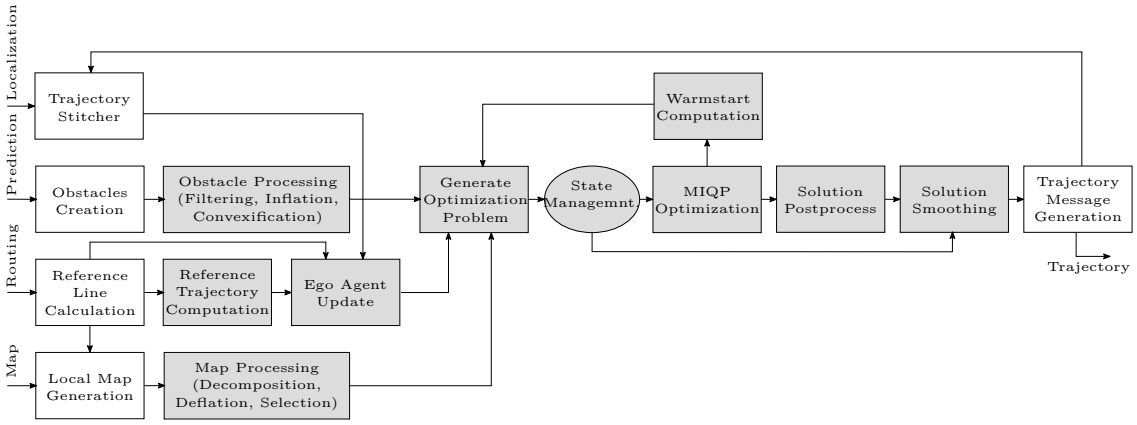


Figure 6.5: Overview of the software components of MINIVAN and the data flow. The components we contribute to in this work are shown in gray. The other components are reused from Apollo. Functional building blocks are represented as boxes, execution control is represented as an ellipse. (graphic from [KEK22], ©2022 IEEE)

6.3 Integration of MINIVAN in Apollo

This section describes how we integrated MINIVAN, introduced in Section 5, into Apollo. Figure 6.5 shows the architectural overview of the software components. Besides its core, solving the Mixed-Integer Quadratic Programming (MIQP) problem, various other functionalities are involved, e.g., in processing the input data, generating the problem in the appropriate coordinate system, and post-processing the solution. In this section, we describe all of the involved software functionalities.

We do not use MINIVAN in driving situations at very low speeds. As elaborated in Section 5.5, the main reason for this is the singularity of the arctan function at very low speeds. The arctan function is needed to translate the absolute vehicle speed to the decomposed speed vectors in x and y direction. Therefore, MINIVAN cannot start driving from a standstill or to bring the vehicle to a complete stop. These low-speed tasks are performed by our nonlinear trajectory smoothing module, as described in Section 6.3.4. We introduced a minimum speed threshold $v_{\text{miqp,min}}$ below which we do not use MINIVAN. Trajectory points from MINIVAN below this speed and all subsequent ones are considered invalid. Through empirical experiments, we identified $v_{\text{miqp,min}}$ for MIQP planning as 0.7 m/s and used 1.0 m/s as a conservative upper bound.

6.3.1 Planner Interfaces and Apollo Software Integration

We integrated our planner as a native component inside Apollo's Docker container. We aimed to reuse as many of the service functionalities already implemented. Therefore, we decided to derive from Apollo's `LatticePlanner` class and integrate our implementation as a new planning module besides the existing planners. The LatticePlanner follows the approach of Werling et al. [WZT10] and is capable of reference line tracking and obstacle avoidance but is not used in Apollo's production setup.

As the implementation of MINIVAN relies on C++ language and Standard Template Library features that are not available with the compiler used in Apollo's Docker container (which is GCC 4.8.5), we did not compile our planner inside Apollo's Docker environment. Instead, we decided to compile it into a library that is linked into the Apollo planning module. This approach also enables us to use third-party dependencies incompatible with Apollo's third-party dependencies, such as more recent versions of the Boost and Eigen libraries. With Apollo, we only compile a wrapper class that calls this library. Furthermore, this wrapper class gathers data, helps assemble the optimization problem, and post-processes the output.

The Apollo planning module has four main inputs. As we integrated MINIVAN into Apollo's planning component, we do not directly interface with the inputs but with already preprocessed data.

- A new planning cycle is triggered once a new prediction message arrives. This messages contains trajectories for moving obstacles and positions for static obstacles. The Apollo planner gathers these and creates a unified structure in a preprocessing step.
- As a second input, the planner takes the localization message containing the ego vehicle's current position. This position is used to estimate a reasonable initial point for the planning.
- The routing request provides the planner with the list of future lane segments on the HD-Map.
- The planner also has access to the HD-Map. Based on the currently active routing request, that forms a sequence of lanes in the map, a local view on the map is generated containing necessary information on the vehicle's driving horizon.

6.3.2 Usage and Modification of Apollo Functionalities

From the routing request, Apollo creates a reference line for planning. This line generally consists of the stacked centerlines of the lanes on the routing corridor. Also, relevant obstacles are extracted from the environment model and a local except of the map. We have not changed these functionalities. Apollo implements a trajectory stitching module to compute a reasonable initial point for the next planning cycle. Three cases are handled: static, moving without or moving along a valid trajectory. If the vehicle is static, the initial point is the current position. If the vehicle is moving, but no old trajectory is available (or the old trajectory is invalid), the trajectory stitcher extrapolates the current vehicle position into the future with the current speed vector for an estimated planning cycle duration.

While Apollo's planner has a rather deterministic low runtime, MINIVAN is terminated at an upper runtime bound. This bound is usually only reached in complicated planning situations, where mostly a feasible but sub-optimal solution is already available. In typical situations, this worst-case threshold is far from the mean runtime. The default behavior of Apollo when moving along a trajectory is to select the next initial planning point as the point on the existing trajectory that the vehicle should reach in one planning cycle. We modified the trajectory stitcher to shift this extrapolation time to our worst-case computation time-bound. This way, we always have a valid trajectory at the downside of additional inertia into the system. For the duration of the worst-case evaluation time, the planner cannot react to newly introduced obstacles, as the trajectory leading to the point at this time is fixed. This modification is, as such, no improvement of Apollo but a necessity to apply a planning concept with a non-deterministic runtime. Once a new valid trajectory is computed, it is appended to the old trajectory at the previously computed initial point. To interpolate the trajectory in the tracking controller, a sufficient backward horizon consisting of the old trajectories is kept.

We also changed the behavior if the planning module fails and cannot create a valid trajectory. In this case, Apollo would trigger an error state and compute an error trajectory, e.g., bring the vehicle to a stop. However, we often observed in experiments that if one planning cycle failed, e.g., due to an unfavorable prediction of one obstacle, the next planning cycle produces a valuable result, resulting in unnecessary error trajectories. Therefore, we decided not to trigger the error state with the first failing planning cycle but at the time instance when the vehicle has reached the end of the current planning horizon. This planner typically recovers to a normal state in a couple of planning cycles. We ensure the old trajectory is still safe by checking this old trajectory for collision with all obstacles. This modification yields more robust and stable overall system behavior without modifying components the stability issue comes from, which we wanted to avoid.

Algorithm 6.4 Obstacle Processing

Input: $\mathcal{O}_{\text{Apollo}}$ ▷ obstacles
Input: P_{roi} ▷ region of interest
Input: C_{d} ▷ convex deflated road cells
Input: r_{c} ▷ collision radius
Input: r_{s} ▷ safety margin
Output: $\mathcal{O}_{\text{MINIVAN}}$

```
for each o ∈ O_Apollo do
    if type(o) is static then
        if BBox(o, 0) ∩ C_d ∩ P_roi ≠ ∅ then
            O_MINIVAN ← Inflate(o, r_c + r_s)
    else                                              ▷ dynamic obstacle
        for each t ∈ 0...T do
            if BBox(o, t) ∩ C_d ∩ P_roi ≠ ∅ then
                O_MINIVAN ← Inflate(o, r_c + r_s)
                break
```

6.3.3 Pre- and Post-Processing the Optimization Problem

We first switch from the coordinate system in the driving direction of the vehicle to the global Cartesian coordinates

$$\begin{pmatrix} p_x & p_y & \theta & v & a & \kappa \end{pmatrix} \longrightarrow \begin{pmatrix} p_x & p_y & v_x & a_x & v_y & a_y \end{pmatrix} \tag{6.1}$$

with velocity v and acceleration a in the direction of the vehicle orientation θ and curvature κ. With velocities and acceleration in x and y direction v_x, a_x and v_y, a_y the transformation is then defined as:

$$v_x = v\cos(\theta), \tag{6.2a}$$
$$v_y = v\sin(\theta), \tag{6.2b}$$
$$a_x = a\cos(\theta) - v^2\kappa\sin(\theta), \tag{6.2c}$$
$$a_y = a\sin(\theta) + v^2\kappa\cos(\theta). \tag{6.2d}$$

The transformation is based on a kinematic single track model [Raj12]. To map the accelerations (6.2c) and (6.2d), we derive the respective velocities and use the relation of orientation change and curvature $\dot{\theta} = v\tan(\delta)/l = v\kappa$ (with steering angle δ and wheel base l).

Obstacle Processing The processing of obstacles is shown in Algorithm 6.4. We first inflate the shape of each obstacle o by the collision circle radius r_{c} plus an appropriate safety margin r_{s}. This margin can be selected differently for different object types. We then compute the minimum bounding rectangle $\text{BBox}(o, t)$ for each object as the shape to check against collision in the MINIVAN optimization. Theoretically, every convex shape would be possible, but rectangles have proven to be a reasonable compromise between accuracy, flexibility, and runtime. For dynamic obstacles, we compute a translated and rotated shape for each point on the predicted trajectory. Obstacles are only considered if they intersect with the convex deflated road cells C_{d} and the region of interest P_{roi}. For dynamic obstacles, the shape of at least one prediction step has to intersect with the environment. The Region of Interest (ROI) P_{roi} is computed as a very simplistic reachable set of the vehicle forming a square. It filters out obstacles that do not interfere with the vehicle anyway, such as objects behind the vehicle.

Reference Trajectory Computation and Ego Agent Update We get the reference line for planning alongside the destination point on this line. We use this line to create a linear reference velocity profile with a constant acceleration phase up to the target velocity and a constant velocity phase when the reference velocity is reached. The reference decelerates with a linear speed profile

Algorithm 6.5 Map Processing

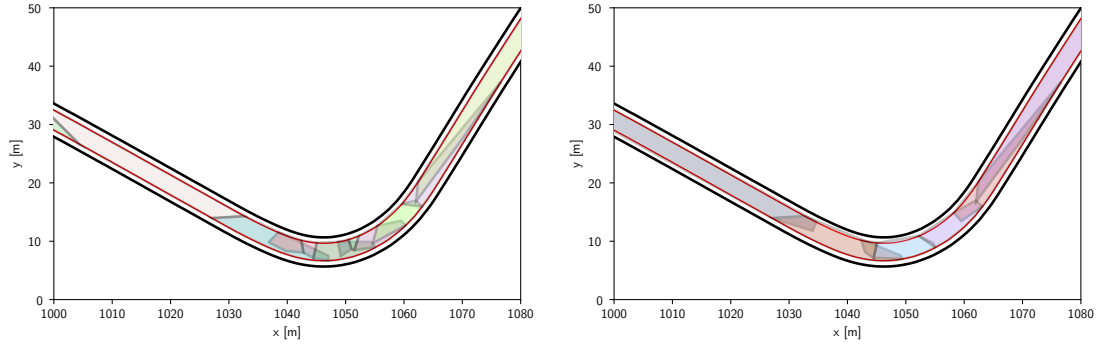
Input: P_{road} ▷ road polygon
Input: δ_{s} ▷ simplification distance
Input: r ▷ circle radius
Input: n ▷ number of merging iterations
Output: convex deflated road cells $\mathcal{C}_{\text{d}}$
1: $P_{\text{s}} = \text{Simplify}(P_{\text{road}}, \delta_{\text{s}})$
2: $P_{\text{s,d}} = \text{Deflate}(P_{\text{s}}, r)$
3: $\mathcal{C}_{\text{voronoi}} = \text{VoronoiDiagram}(P_{\text{s,d}})$
4: $\mathcal{C}_{\text{voronoi}\cap\text{road}} = \{\}$
5: **for each** $C \in \mathcal{C}_{\text{voronoi}}$ **do**
6: **if** $C \cap P_{\text{s,d}} \neq \emptyset$ **then**
7: $\mathcal{C}_{\text{voronoi}\cap\text{road}} \leftarrow C \cap P_{\text{s,d}}$
8: $\mathcal{T} = \text{Triangulate}(\mathcal{C}_{\text{voronoi}\cap\text{road}})$
9: $P_{\text{d}} = \text{Deflate}(P_{\text{road}}, r)$
10: $\mathcal{C}_{\text{d}} = \mathcal{T}$
11: **for each** $i \in 1...n$ **do**
12: $\mathcal{C}_{\text{d}} = \text{MergePolygons}(\mathcal{C}_{\text{d}}, \tau_{\text{m}}, P_{\text{d}})$

to zero velocity when approaching the goal point. We scale the target speed with the curvature and a maximum accepted lateral acceleration threshold in the curves.

As a start point for the reference, we select the initial point for planning. The second point of the reference is selected as the point on the reference line with minimal Euclidean distance in front of the vehicle. The initial velocity is chosen as the velocity of the initial point for planning. Note that this straightforward reference generation strategy can yield suboptimal results for starting positions far off the reference line, as possibly the optimization tracks an unreachable reference. We do not adapt the reference line when intersecting with obstacles or the environment but expect MINIVAN to resolve these conflicts. From this reference line and speed profile, we sample an interpolated reference trajectory as an input to the planning step.

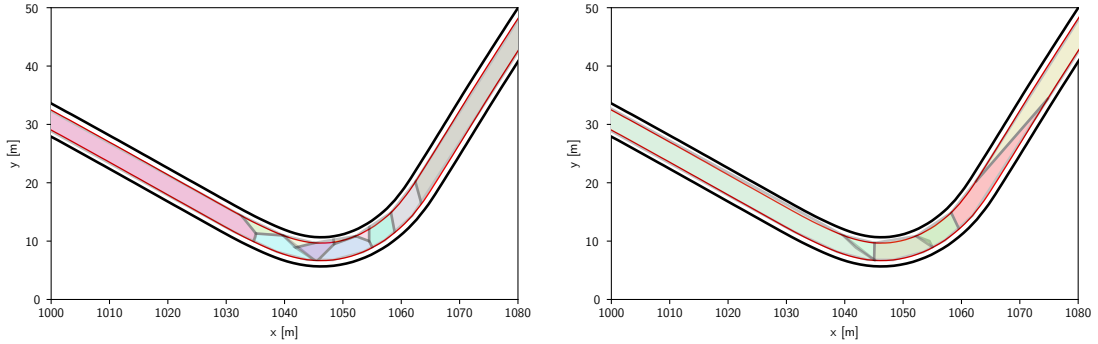
Map Processing MINIVAN relies on a convex approximation of the available free-space for navigation. Nearly all road shapes are non-convex and therefore have to be decomposed into several convex sub-polygons. A naive triangulation is possible but undesirable, as the number of constraints scales linearly with the number of environment polygons. Therefore, we aim for an efficient convexification algorithm that outputs as few polygons as possible while still covering the whole area. Algorithm 6.5 shows the pseudocode of this convexification, where P denotes an arbitrary-shaped polygon, C denotes a convex polygon, T denotes a triangle, and $\mathcal{P}, \mathcal{C}, \mathcal{T}$ the respective sets.

We construct the (non-convex) road polygon P_{road} from the left and right road boundaries and simplify it in line 1 using the Ramer-Douglas-Peucker algorithm. We deflate the environment polygon with the radius of the circles that approximate the vehicle and construct the Voronoi diagram on this polygon. using a sweep line algorithm [For86]. Each point of the input polygon is used as one point for constructing the Voronoi diagram. No linear segments are used. We create a sufficiently large bounding box around all points and add it to the construction of the Voronoi diagram to avoid infinite edges in the creation process. The Voronoi cells $\mathcal{C}_{\text{voronoi}}$ from the Voronoi diagram represent a segmentation of the available space into polygons. In our case, all cells are polygons with straight edges. In lines 5-7, we intersect the deflated original polygon with the set of Voronoi cells and drop all cells outside the original polygon. As Voronoi cells are not necessarily convex, we decompose the cells in line 8 into triangles. Finally, line 12 iteratively merges the triangles into bigger convex cells in a greedy manner as follows. Starting from one triangle, we merge it with a neighboring triangle. We then check if the convex hull of this union lies within the original (non-convex) road polygon or the union is itself convex. If yes, we try to merge further triangles. If not, we close this sub-polygon and start a new one. We do this until



(a) Simplification distance = 0.1m, convex merging tolerance = 0.1m (12 polygons)

(b) Simplification distance = 0.1m, convex merging tolerance = 0.5m (7 polygons)



(c) Simplification distance = 0.5m, convex merging tolerance = 0.1m (11 polygons)

(d) Simplification distance = 0.5m, convex merging tolerance = 0.5m (6 polygons)

Figure 6.6: Results of the convexification of an example environment, a lane corridor forming a curve. In black, the original lane boundary is depicted, as red line, the deflated lane corridor, that shall be decomposed into convex polygons (colored shapes). (graphics from [KEK22], ©2022 IEEE)

every triangle is processed. This procedure does not guarantee an optimal result. To cope with this, it has proven to be beneficial for the road shapes used in this work to iterate again once or twice over the resulting list of convex sub-polygons to reduce the number of polygons further and find larger convex shapes (line 11).

The convexity checks in the merge step are performed with a particular error margin that two shapes are merged, even if the union is marginally non-convex. Then the convex hull is used. This robust convexity check avoids numerical errors and helps to create significantly fewer sub-polygons. However, it introduces errors in the road approximation, which we add as a safety margin to the collision shape of the agents. As the last merging step, we simplify each polygon in the set of convex-sub polygons to avoid very small edges. We inflate each polygon by the maximum simplification error again to avoid holes between polygons. With the described merging strategy, the set of sub-polygons can intersect, which is not a problem for MINIVAN.

The number and size of the sub-polygons are highly dependent on the two introduced parameters: the simplification distance δ_s and the convexity threshold τ_m for merging two polygons. Hence, we can find a balanced approximation accuracy and sub-polygons size ratio with an appropriate setting. In Figure 6.6 we compare three different settings and show the effect of the parameters. We observe that the number of polygons is reduced with increased merging tolerance. With low values, the approximation of the (red) deflated environment shape is accurate, whereas a notable approximation error at the inner part of the curve occurs with higher values.

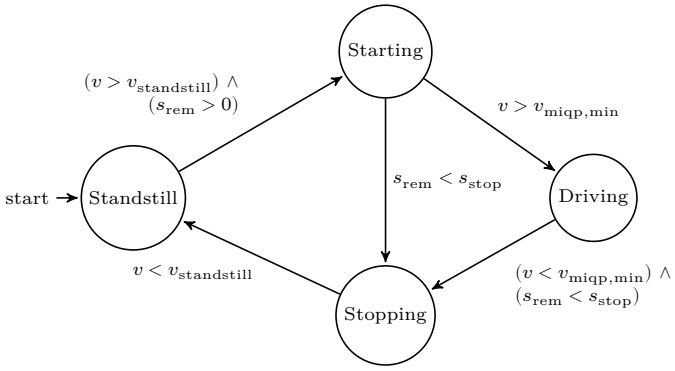


Figure 6.7: The finite state automaton controlling the planning sequence. Using the current velocity and the remaining distance on the trajectory, state transitions are triggered between the Driving (default), Starting, Stopping, and Standstill state. (graphic from [KEK22], ©2022 IEEE)

The strategy described here has proven to generate a sufficiently accurate environment decomposition with a fast runtime. Still, we want to emphasize that we rely on a greedy heuristic, and theoretically, shapes can be constructed where the algorithm yields poor results. We did not observe such problems in the tested road geometries. In a specific planning cycle, possibly not all convexified sub-polygons are needed, as most do not lie on the current vehicle horizon. As the solution complexity scales with the number of convex environment polygons, we only use the polygons that intersect with the reference trajectory of the current planning instance with some tolerance to make sure enough space is available if the vehicle, e.g., performs an evasive maneuver or drives faster than the reference.

Warmstart Computation In Section 5.5.2, we elaborate how we speed up the optimization using a warmstart solution.

Optimization Problem Generation This stage collects all previously generated data and assembles the optimization problem. We compute values specific to the MINIVAN optimization, such as the initial region and the possible regions in this planning cycle, followed by various consistency checks to avoid infeasible optimization problems, e.g., ensuring to start inside the lane polygon. Here we also load the warmstart solution from the previous planning cycle.

Planner State Management Our planner implementation is controlled by a finite state automaton with four states representing different driving situations:

- **Standstill**: We do not trigger a planning cycle but maintain the current position.
- **Starting**: We omit the MINIVAN planning step but compute a trajectory along the reference path and speed profile using the trajectory smoothing module.
- **Driving**: In this default state we set up, solve, and smooth the resulting trajectory.
- **Stopping**: We omit the MINIVAN planning step but compute a trajectory along the reference path and speed profile (ramped down to zero) using the trajectory smoothing module.

Figure 6.7 shows the resulting automaton and its respective transitions, where

$$s_{\text{stop}} := \min(s_{\text{goal}}, s_{\text{blocking}}). \tag{6.3}$$

The automaton transitions from Standstill to Starting if the velocity is greater than a standstill threshold and the distance to the routing destination s_{goal} is greater than zero, and the path is not blocked by an obstacle right in front of the vehicle (s_{blocking}). The automaton transitions from Starting to Driving if the current velocity is greater than $v_{\text{miqp,min}}$, and from Starting to Stopping if the vehicle approaches the destination point. A transition is triggered from Driving to Stopping if the vehicle approaches the destination point and the current speed falls below $v_{\text{miqp,min}}$. Once the velocity drops below $v_{\text{standstill}}$, the automaton transitions from Stopping to Standstill. In the Starting and Stopping state, we only consider obstacles that are immanently in front of the vehicle and do not plan trajectories avoiding all obstacles. This, of course, is a restriction of our system that we consider acceptable for a research platform.

Post-processing of the MIQP Solution After a successful MINIVAN optimization, we post-process the result. First, we compute and store the warmstart for the next planning cycle according to the warmstart strategy described in Section 5.5.2. We then transform the solution trajectory back from global Cartesian coordinates into the vehicle-relative frame having absolute values in vehicle orientation

$$\begin{pmatrix} p_x & p_y & v_x & a_x & v_y & a_y \end{pmatrix} \longrightarrow \begin{pmatrix} p_x & p_y & \theta & v & a & \kappa \end{pmatrix} \tag{6.4}$$

using trigonometric formulas and appropriate derivations:

$$\theta = \arctan(\frac{v_y}{v_x}), \tag{6.5a}$$

$$v = \frac{v_x}{\cos(\theta)}, \tag{6.5b}$$

$$a = \cos(\theta)a_x + \sin(\theta)a_y, \tag{6.5c}$$

$$\kappa = \frac{v_x a_y - a_x v_y}{(v_x^2 + v_y^2)^{3/2}}. \tag{6.5d}$$

To ensure no approximation errors have been introduced, we check the resulting trajectory for collision with the unmodified obstacle shapes and environment polygon. In the final step, we perform trajectory smoothing as described in Section 6.3.4.

6.3.4 Trajectory Smoother

MINIVAN does not allow putting direct costs on absolute acceleration change and curvature change. However, both are desirable to generate a comfortable driving experience. Also, due to the approximation of the orientation into a set of discrete regions, the vehicle orientation at the boundaries of regions can be inaccurate, and the orientation change can be non-smooth. Furthermore, in MINIVAN, we want to use a more extensive time discretization to achieve a longer planning horizon that we subsample in the smoother to hand over a smooth and more fine-grained trajectory representation to the controller. Note that due to the overapproximation of the vehicle shape, this is not a safety issue when avoiding obstacles. Nevertheless, it can lead to uncomfortable driving behavior.

Therefore, we introduce a trajectory smoothing step to find a trajectory T_{out} as a nonlinear MPC optimization in the vehicle frame to stay as close as possible to the original trajectory T_{in}. Obstacles are not directly used in the smoothing optimization but only indirectly through the input trajectory. We implicitly perform interpolation by increasing the number of support points from T_{in} to T_{out}.

We define $x^k := (p_x^k, p_y^k, \theta^k, v^k, a^k, \kappa^k)$ and $u^k := (j^k, \dot{\kappa}^k)$, where superscript k denotes the discrete timestep. We use Heun's method to integrate the single-track model equations. Generally, any explicit integration method would be possible, also the explicit Euler method, that needs

less function evaluations. By using a two-stage integration method, the introduced errors are significantly lower. The discretized system dynamics can then be formulated as

$$p_x^{k+1} = p_x^k + \frac{1}{2}\Delta t v^k \cos(\theta^k) + \frac{1}{2}\Delta t(v^k + \Delta t a^k)\cos(\theta^k + \Delta t v^k \kappa^k), \tag{6.6a}$$

$$p_y^{k+1} = p_y^k + \frac{1}{2}\Delta t v^k \sin(\theta^k) + \frac{1}{2}\Delta t(v^k + \Delta t a^k)\sin(\theta^k + \Delta t v^k \kappa^k), \tag{6.6b}$$

$$\theta^{k+1} = \theta^k + \frac{1}{2}\Delta t v^k \kappa^k + \frac{1}{2}\Delta t(v^k + \Delta t a^k)(\kappa^k + \Delta t \dot{\kappa}^k), \tag{6.6c}$$

$$v^{k+1} = v^k + \frac{1}{2}\Delta t a^k + \frac{1}{2}\Delta t(a^k + \Delta t j^k), \tag{6.6d}$$

$$a^{k+1} = a^k + \Delta t j^k, \tag{6.6e}$$

$$\kappa^{k+1} = \kappa^k + \Delta t \dot{\kappa}^k. \tag{6.6f}$$

We place bounds on the states

$$v_{\max} < v(k) < v_{\max}, \tag{6.7a}$$

$$\kappa_{\max} < \kappa(k) < \kappa_{\max} \tag{6.7b}$$

and inputs

$$j_{\max} < j(k) < j_{\max}, \tag{6.8a}$$

$$\dot{\kappa}_{\max} < \dot{\kappa}(k) < \dot{\kappa}_{\max}. \tag{6.8b}$$

We compute a quadratic cost function

$$J(u_1, ..., u_N, x_1, ..., x_N) = \sum_{k=1}^{N} \left((x^k - x_{\text{in}}^k)^\top Q_1^k (x^k - x_{\text{in}}^k) + (x^k)^\top Q_2^k x^k + (u^k)^\top Q_3^k u^k \right) \tag{6.9}$$

where x_{in} denotes the state of the input trajectory to the smoother module, which may not be available at some time instances due to the wider sampling of the input trajectory T_{in}. Therefore, we select the weight matrices Q_1^k, Q_2^k, Q_3^k in a way to not penalize deviations of subsampled states. Q_1 ensures minimal deviations from the input trajectory and Q_2, Q_3 ensure a smooth motion.

6.4 Trajectory Tracking Control

This section describes how we implemented and tested our trajectory tracking controller. We show how we extended a trajectory tracking algorithm from literature and state the component's architecture. As main contributions, we consider the description of the communication setup between the non-real-time trajectory planner, the real-time trajectory controller, and the vehicle control units.

We modeled the controller and the CAN and Ethernet interfaces in Mathworks Simulink. The handling and interpreting of Apollo's Protocol Buffers messages are embedded as C-Code S-Functions. We execute the model on a development PC for a seamless module test without real-time hardware. Mocking the same interfaces as the vehicle CAN bus enables us to run open or closed-loop tests with the controller running in Simulink on a computer while receiving trajectory and localization from Apollo or recorded data. To debug errors occuring in test drives, it has been proven valuable to record all input signals and internal controller states using the toolchains offered of the different platforms [Min+16]. A unified analysis and plotting framework has been implemented to analyze the recorded data.

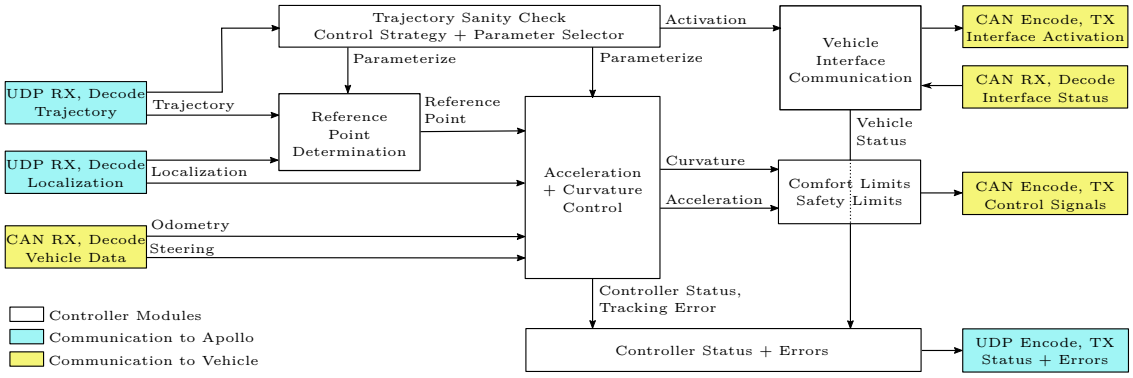


Figure 6.8: Architecture overview of the trajectory tracking controller with the longitudinal/lateral vehicle control data flow from the left to the right. The module parameterization and error handling flow from top to bottom. (modified graphic from [Kes+19], ©2019 IEEE)

6.4.1 Control Algorithm

The control algorithm is based on the work of Werling et al. [WZT10] and shown as a block diagram in the middle part of Figure 6.8. This work focuses on driving scenarios, including interactions with other dynamic traffic participants; therefore, we apply full trajectory tracking by interpolating the time on the trajectory received from the planning component. We use the localization time signal as a common time base for the trajectory planner and the controller. Full trajectory tracking is especially valuable when driving on longer road segments with sufficient time and space to cope with sensor and actuator errors. The trajectory planner must ensure that a sufficient backward horizon of the trajectory is available to guarantee a valid result of this interpolation.

When starting to drive from a standstill, we implemented two special cases to avoid huge tracking errors due to the vehicle's inertia. First, all trajectories are dropped, if not all low-level vehicle interfaces are ready to drive. Once the interfaces are set up, the controller waits for the subsequent trajectory to start driving. Otherwise, the controller could try to track an unreachable reference point which results in unnecessary high control values. The introduced delay of one planner cycle is neglectable. Second, for very low velocities, we do not perform a time-based trajectory tracking but track the one trajectory point with zero time. Proceeding this way also avoids high control signals if the vehicle cannot track the accelerating trajectory properly, e.g., when starting to drive. As soon as the vehicle starts moving, we switch to the trajectory tracking mode. With appropriate parameterization reflecting the accelerating behavior of the vehicle, the introduced error when switching is low.

The tracking errors and their derivatives are extracted from the tracking point in a Frenet reference frame. Using input substitution and backstepping asymptotic stability of the control law can be proven [WZT10]. For the sake of driving smoothness and planner error tolerance, we limit the absolute values and rates of all control signals. A separated software module, implemented as a finite state automaton, activates and parameterizes the different controller components based on the received trajectory and a host PC HMI command (omitted in Figure 6.8). Doing so, we achieve a complete separation of control algorithm code from functional execution logic.

We perform basic consistency checking of each trajectory and localization signal in terms of data and time validity, and time deviation. Furthermore, we detect actuator failures and vehicle interface errors. In case of an error, the control is handed over to the safety driver. Higher-level components are expected to treat more sophisticated errors like a degraded performance in tracking quality.

In [Kes+17] we show that in the presence of sensor and actuator errors that are not treated in the planning layer, the control concept implemented here has deficits in robustness. This work

assumes errors to be treated in the planning layer and does not focus on maximized robustness. An advanced MPC scheme, such as [LKK15], can stabilize a trajectory even with errors present.

6.4.2 Communication Interfaces

Besides performing the trajectory tracking, the controller module translates from publish/subscribe messages in Apollo's middleware to cyclic messages on the vehicle's CAN bus.

Trajectory Planner Communication Interfaces

The control algorithm runs on a rapid prototyping hard real-time computation platform, whereas PC-like hardware executes the trajectory generation components in a soft real-time context. As the trajectory is available to the controller over a particular horizon, no real-time communication between planner and controller must be implemented. Consequently, delays or packet loss in communication become acceptable. Also, short planner computation delays still result in smooth motions. In case of a planner software failure, the controller can still evaluate the last valid trajectory and can trigger an emergency stop or hand over the control to the safety driver. The controller assumes that while driving, always a valid trajectory is available and triggers an error state otherwise. The trajectory planning module outputs a collision-free trajectory over the time horizon (with respect to the predicted motion of dynamic traffic participants) and transfers a sampled representation on a defined horizon to the trajectory controller. The controller performs no collision checks allowing it to run at a high frequency.

Figure 6.8 depicts the interfaces of the controller and the communication channels. The communication to the trajectory planner and the localization module is realized using Ethernet with a User Datgram Protocol (UDP). With the interpolation method described in Section 6.4.1, no assumptions on the time or spatial distance of trajectory points are required. As an encoding format of the messages, we use the Protocol Buffer definitions for trajectory and localization from Apollo. To shrink the size of the transmitted trajectories, we take the original trajectory message in Apollo and remove unnecessary information for the controller, such as debug values, redundant fields, or unused variables. Still, due to size limitations in the UDP stack implementation on the real-time device, trajectory messages are split into multiple messages and have to be reassembled again. We also ensure trajectory and localization are transmitted with a suitable frequency. The control algorithm is evaluated on a real-time device with a fixed cycle time. Note that localization and trajectory messages in Apollo are still standard messages, and we transmit them to the Autobox using the standard Bridge component. On the Autobox side, we re-implemented the pack and unpack functionalities of the Bridge component in ANSI C. This way, no specialized components are necessary, and the Autobox acts like a standard component in the overall system.

Vehicle Communication Interfaces and Execution Platform

In contrast to the Apollo reference vehicle, we separate the computers for planning, perception, and other driving functions from the closed-loop vehicle control. That way, in case of timing problems on the computers or Ethernet network outtakes, we can still keep up the control loop on a short horizon. The trajectory controller is executed with a cycle time of 0.01 s. Only this real-time hardware holds access to the vehicle controls.

The controller does not directly influence the vehicle actuators but computes high-level control values like acceleration and steering wheel angle. A subsequent vehicle gateway control unit uses these high-level control values and actuates the production vehicle's control unit interfaces to control the vehicle motion. The main interfaces are

- the acceleration interface from the production vehicle's automatic cruise control,
- the steering wheel angle interface from the production vehicle's park assistance system.

For low-speed and maneuvering scenarios with reverse driving segments, the gear selection, throttle, and brake are actuated directly. No modification of any production control units is necessary. The implementation of the low-level interfaces and the actuation of the production control units are realized on private CAN buses. These are proprietary and out of the scope of this work.

6.5 Lessons Learned

After having described the experimental vehicle based on Apollo and the integration of MINIVAN we will now elaborate on our experiences with this setup.

The Apollo Platform

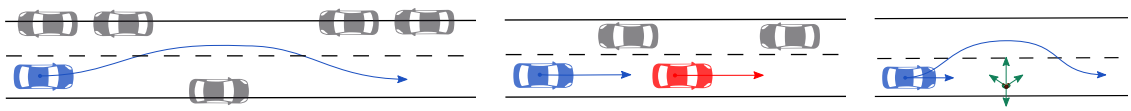
Apollo's setup and build environment are easy to use for experienced developers after an initial hurdle, and the usage of one pre-configured Docker container eases this. Still, the pure size and number of dependencies, combined with the non-comprehensive documentation and hidden inter-dependencies, increases the risk of breaking an existing component when introducing changes in the build environment. Also, the usage of various outdated versions of packages and software tools (such as the Bazel build tool or the Boost C++ libraries) makes porting existing software into Apollo's Docker environment cumbersome. All modules are compatible with each other.

The performance of the whole Apollo system heavily relies on the accuracy and level of detail of the HD-Map. For example, the planning component uses lane centerlines as a reference line the vehicle should drive on. The prediction component relies on the assumption that road users intend to stay close to the centerline of their respective lanes. Therefore, an inaccurate, unrealistic, or asymmetric true lane center yields unrealistic predictions or undesired ego planning maneuvers. Also, various ROI filters operate on lane geometries and benefit from maps with high accuracy. We decided not to activate the ROI filtering in the perception component, not to be dependent on the accuracy and detail of the map. The benefit is that we do not lose any relevant obstacles, while the drawback is having more objects available for the prediction and planning to process. The generation and maintenance of the maps in Apollo's customized format was an underestimated effort, as no standardized and open toolchain is available. Furthermore, the routing and maneuvering are purely map-based, which again poses high demands on the quality of the map topology. An inaccurate map is thus a very crucial issue.

The simulation engine shipped with Apollo is suitable for preliminary integration tests of a new planner but not comprehensive. For example, no obstacle simulation is available. Also, the vehicle motion is extracted from the planned trajectory position and not determined by a tracking controller component. Baidu offers a more comprehensive, cloud-based simulator that was not examined in this work, as the planner evaluations were executed in the BARK simulator [Ber+20]. BARK is designed as a benchmarking tool for planning algorithms, and could be used for a simulative comparison of MINIVAN and Apollo's planner. We leave this to future work.

The Cyber RT middleware worked smoothly without introducing noticeable delay, jitter, latency, throughput bottlenecks, or hiccups. Distributing the components over multiple computers did not show any problem and only needed accurate time synchronization and appropriate configuration. Integrating additional components based on existing implementations is straightforward. The toolset provided with Cyber RT is sufficient but less comprehensive than the toolset of ROS. All basic functionalities for monitoring and debugging the system are available, such as recording and visualization. For analysis, mainly scripting Application Programming Interfaces (APIs) exist, but no set of prebuilt tools.

Apollo's lidar pipeline proved to work well also with a 32-layer Velodyne lidar. Even though we do not have internal knowledge of the involved networks, i.e., how the networks were trained, and cannot judge the quality of the training data, the overall performance of the perception systems translates to German roads. As we will discuss later in Figure 6.14 the accuracy of the detection



(a) Static vehicle avoidance (b) Vehicle following (c) Pedestrian avoidance

Figure 6.9: Schematic sketches of the three evaluation scenarios.

varies. Still, the performance is good enough for the conducted experiments, and no effort was invested in mitigating suboptimal object detection and size estimation. However, we only tested this for low-speed scenarios. Evaluating the accuracy in scenarios with higher speeds remains to be studied.

The data fusion component proved to be configurable for the usage in our setup with different sensor types and positions due to its configurable and modular code structure. Object recognition and segmentation provide state-of-the-art results that are sufficient for on-road driving experiments. The drawbacks are the high dependency of the relevant object-filtering algorithms to the HD-Map and the high demands on the graphics card. In the setup with only lidar sensors, object type classification often showed unreliable results for non-vehicle object types.

Apollo's CAN bus component is extendable but based on low-level parsing of the CAN messages, which makes integration of custom CAN buses time consuming and requires testing effort. No parsing and automated analysis of CAN database files or similar are provided.

Fortuna's System Setup

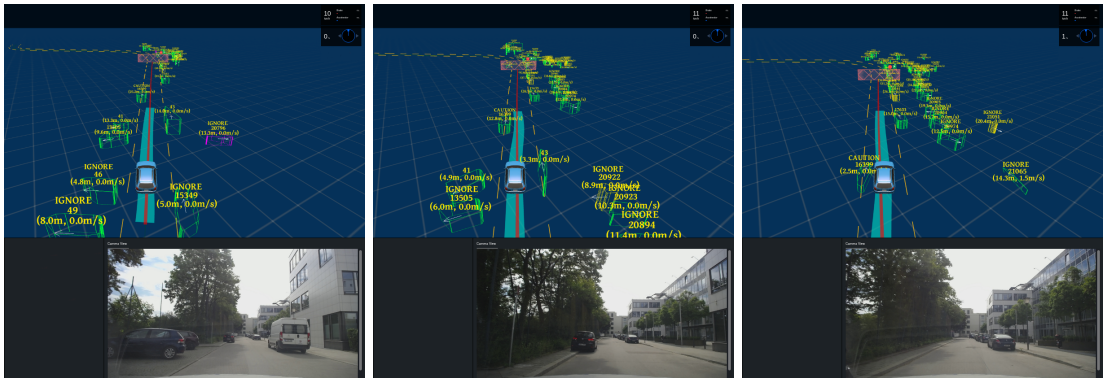
To simplify the system setup, we chose to use the localization information and not the IMU information for control as our differential GNSS position is stabilized with three fiber-optical gyroscopes, an accelerometer, and a high-resolution wheel tick sensor. The assumption that tracking control is possible on localization data in such a setup turned out to be false. The data quality in terms of stability and absence of jumps is not high enough for robust trajectory tracking. Future work should also include lidar information in the localization and base the trajectory tracking control solely on IMU data.

To maintain the experimental platform's stable setup, we reduced the system complexity at various points, such as mainly relying on lidar as sensor technology. All simplifications are not transferable to production-grade automated vehicles but proved to fit the needs of a research vehicle except for the high reliance on the GNSS. While the employed minimal sensor set to reduce engineering maintenance served our needs, relying on the differential GNSS did not.

The planning and prediction components in Apollo do not operate together interactively. Trajectories from prediction have to be post-processed in the planner. Otherwise, frequently errors occur, such as vehicles driving from behind into the ego vehicle or pedestrians walking on the side of the street being predicted to jump in front of the vehicle. This yields unnecessary planning errors. However, the prediction component yields a good baseline performance and reliable trajectories for vehicles moving on roads. Generally, the prediction system focuses on the lane centers and tends to project obstacles to follow these lane centerlines.

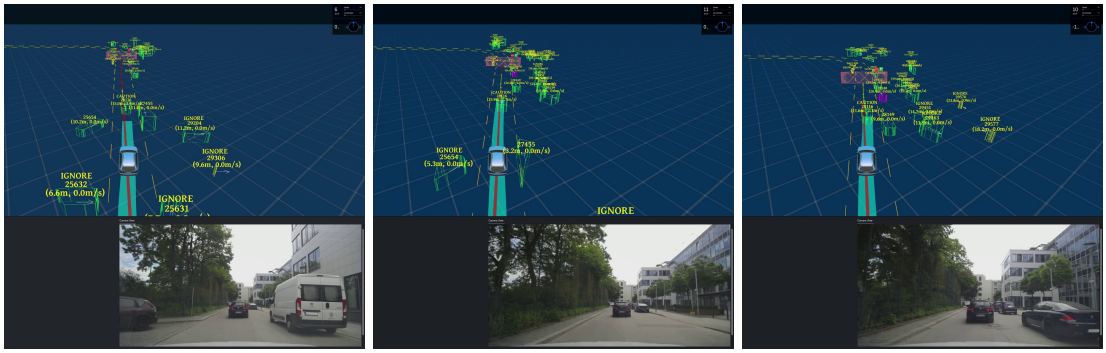
6.6 Evaluation in Real-Road Scenarios

To evaluate the capabilities of MINIVAN, we performed several experiments on a real road, from which we analyze three here, schematically sketched in Figure 6.9. We chose to analyze the planner's performance and overall system setup in these benchmarks scenarios and not in arbitrary road situations to have reproducible results and examine specific strengths and weaknesses in detail. Figure 6.10 – Figure 6.12 show the test drives.



(a) 3 s after start (b) 7 s after start (c) 11 s after start

Figure 6.10: Visualization of the static vehicle avoidance scenario showing the vehicle front camera alongside with the processed obstacles and planned trajectory.

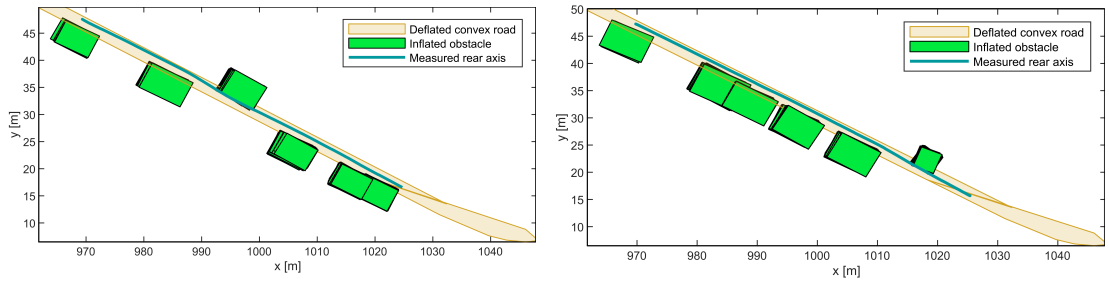


(a) 5 s after start (b) 10 s after start (c) 16 s after start

Figure 6.11: Visualization of the vehicle following scenario showing the vehicle front camera alongside with the processed obstacles and planned trajectory.

(a) 11 s after start (b) 16 s after start (c) 18 s after start

Figure 6.12: Visualization of the pedestrian avoidance scenario showing the vehicle front camera alongside with the processed obstacles and planned trajectory. (graphic from [KEK22], ©2022 IEEE)



(a) Static vehicle avoidance scenario

(b) Pedestrian avoidance scenario

Figure 6.13: Obstacle avoidance scenarios with deflated road polygon, inflated obstacles, and the driven path. The inflated obstacles are shown throughout the scenario. The resulting trajectory is collision-free, although it appears to be colliding due to inaccurate obstacle measurements. On the right hand side, we see parked cars and on the left hand side the obstacle an evasive maneuver is planned for. (This visualization is not suited for the vehicle following scenario and it is therefore omitted here.) (graphic from [KEK22], ©2022 IEEE)

Specifically, the three scenarios analyzed in detail are:

- **Static vehicle avoidance**: avoiding a static vehicle on the right side, sketched in Figure 6.9a. The evolution of the scenario is shown at three different timesteps in Figure 6.10.
- **Vehicle following**: approaching a slower preceding vehicle, eventually slowing down and accelerating again once the preceding vehicle accelerates, sketched in Figure 6.9b. The evolution of the scenario is shown at three different timesteps in Figure 6.11.
- **Pedestrian avoidance**: avoiding a static pedestrian standing on the road, sketched in Figure 6.9c. The evolution of the scenario is shown at three different timesteps in Figure 6.12.

Figure 6.13 displays both avoidance scenarios as MINIVAN processes the environment data. The obstacle shapes are plotted for each timestep and are already inflated for planning. The environment geometry is plotted deflated. The resulting trajectory is collision-free, but due to inaccurate obstacle measurements, which we will discuss in Section 6.6.1, it appears colliding.

6.6.1 Perception

To study Apollo's off-the-shelf object detection quality, we consider only the two scenarios with static obstacles. We analyze the deviation $(\cdot) - \overline{(\cdot)}$, where $\overline{(\cdot)}$ denotes the mean value. We perform the analysis for the object's center position (x, y), orientation, and the bounding box width and length. Figure 6.14 shows the resulting box plot. Extreme outliers (outside of the whiskers) are rare but may happen with deviations of more than 1.3 m. Comparing the detection of vehicle and pedestrian, the errors regarding position and bounding box of the pedestrian are significantly smaller. On the other hand, the orientation of the pedestrian is much noisier than the orientation estimation of the vehicle. We expect that to improve when fusing with other sensors, e.g., cameras. However, poor orientation estimation of pedestrians may harm the prediction quality. As a takeaway, we can conclude that our planner needs to cope with noise within the standard outliers. We have thus added safety margins to the obstacles processed within the planning problem.

6.6.2 Planning

To answer the question of MINIVAN's real-time capability, we have analyzed the frequency of the perception, prediction, and planning modules. The goal for our planner is to come up with a new trajectory within at least 1 s. Lidar sensor data triggers the perception, which triggers prediction,

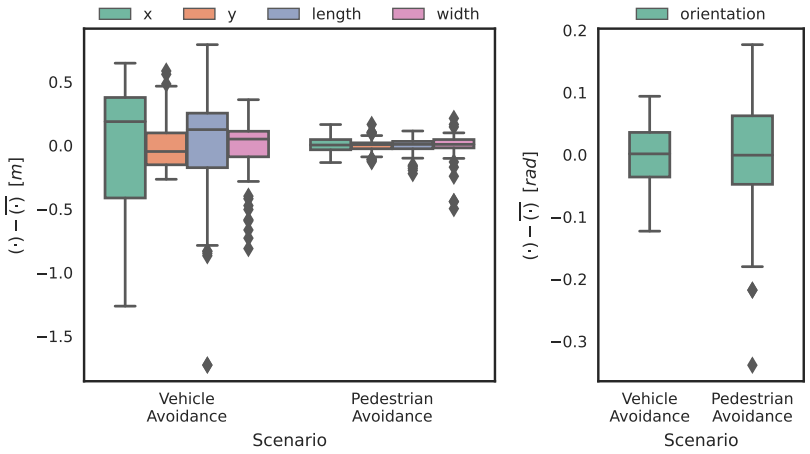


Figure 6.14: Distribution of the deviation from the mean value of pose and bounding box dimension of the respective static obstacle. We observe, that especially the position of there is some uncertainty in the perception of a static vehicle. (graphic from [KEK22], ©2022 IEEE)

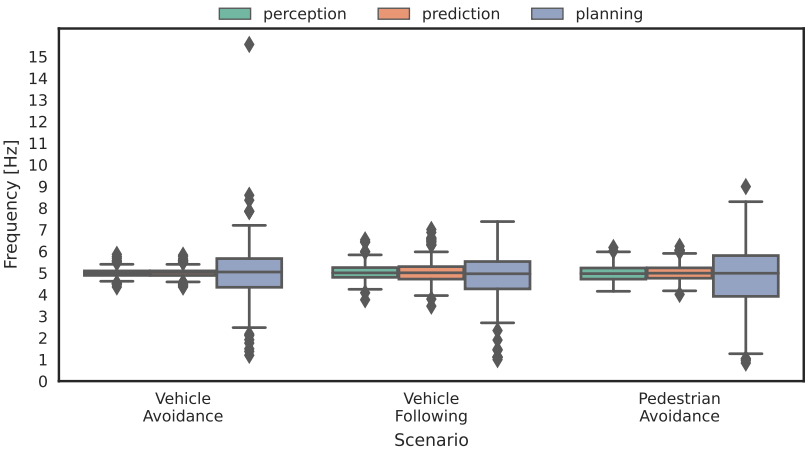


Figure 6.15: Distribution of the frequency of the perception, prediction, and planning modules. On average, the overall system meets 5 Hz, but the planning frequency also drops to 1 Hz at single, complicated planning instances. (graphic from [KEK22], ©2022 IEEE)

which triggers planning. Figure 6.15 shows the results in the vehicle avoidance scenario, all others are comparable. We configured the lidar to publish sensor data at 5 Hz. Both the perception and prediction components are capable of running at 5 Hz with some standard outliers down to 4 Hz and up to 6 Hz (to catch up). Increasing the lidar's publishing frequency would, at our planning frequency, only raise the system load. Our planner still runs at a mean frequency of 5 Hz. However, the extreme outliers go down to 1 Hz, which mostly happens when being close to an obstacle, where the solution space becomes narrow.

Second, we have analyzed the time consumption of the stages of the planning component: pre-processing, solving the MIQP problem, and post-processing. Figure 6.16 shows the results. Pre-processing the input (e.g., decomposition of the map) is fast, as we are caching relevant data (e.g., decomposed parts of the map). Likewise, post-processing (including smoothing) is very efficient. The duration of pre- and post-processing is reliably below 0.06 s and does not have

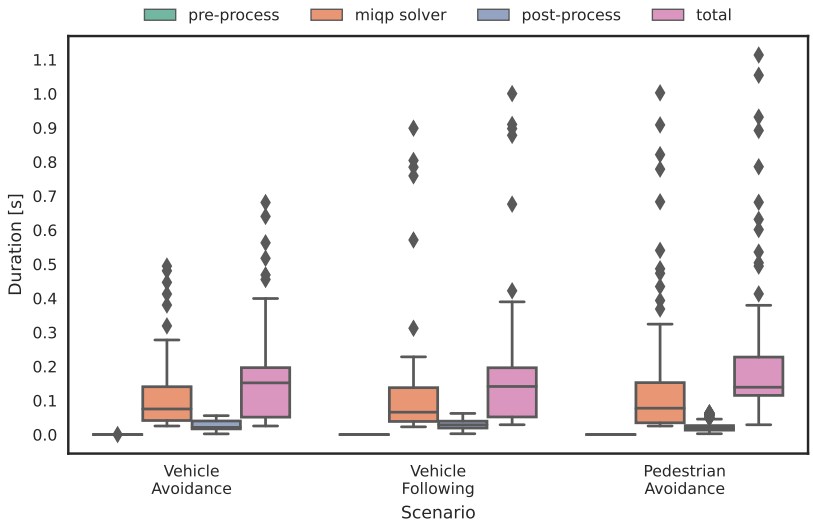
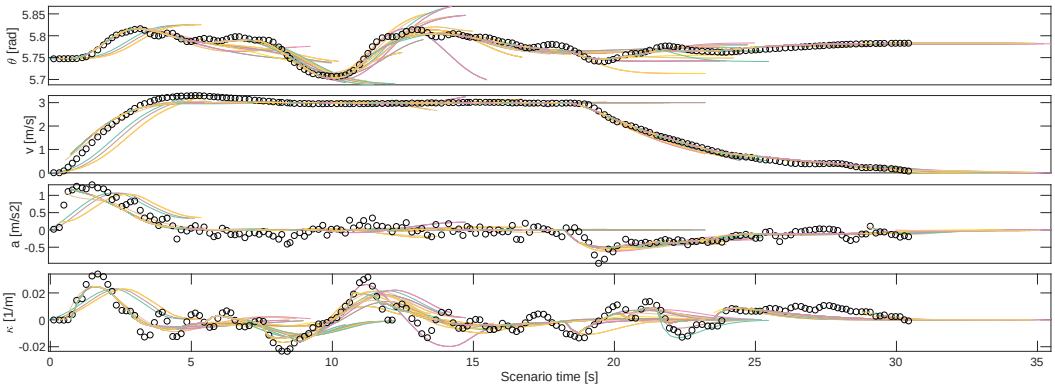


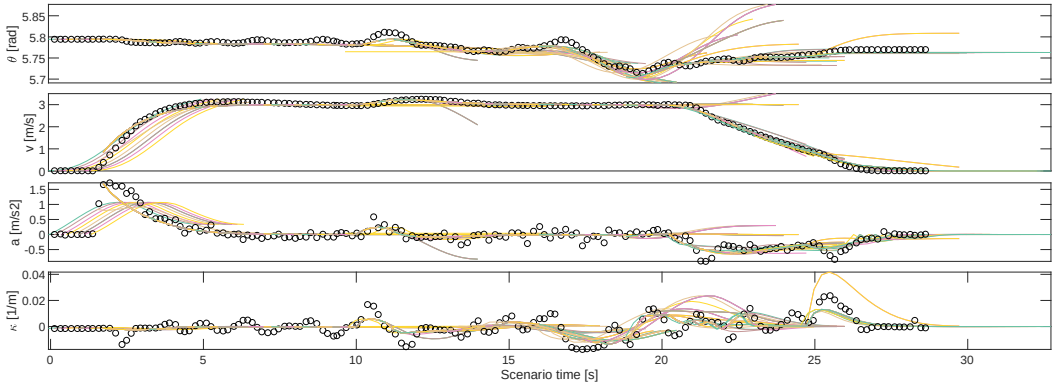
Figure 6.16: Distribution of planning times of the respective parts of the planning module. Clearly, the solution time of the MIQP solver is responsible for the majority of time, including the peaks in the runtime. On average, the MIQP solution time is low with 0.1 s. (graphic from [KEK22], ©2022 IEEE)

significant outliers. Most of the planning time is used by the MIQP solver. It has significant outliers of up to 1 s (when the solver is terminated), which consequently makes the total planning time range up to more than 1 s. The analysis shows that our intended general-purpose planning for arbitrary roads that avoids static and dynamic obstacles works smoothly at a replanning rate of 4 Hz to 6 Hz with outliers down to 1 Hz.

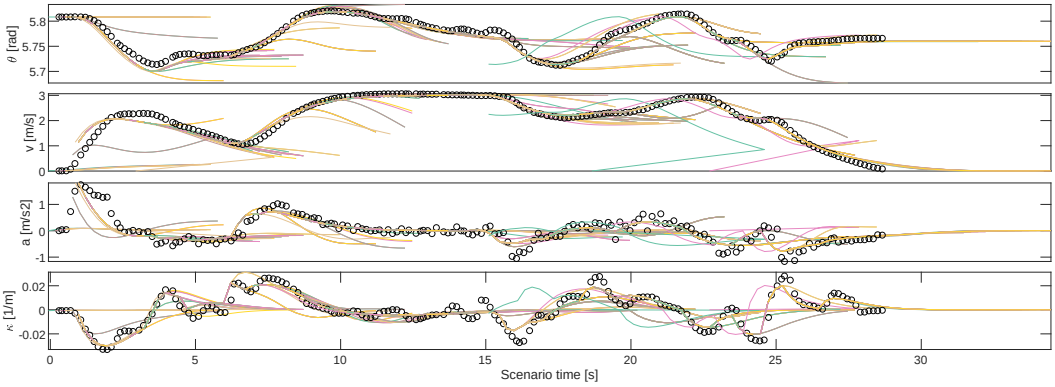
In Figure 6.17 we depict the planned trajectories over time in each scenario alongside the actual measured value to analyze the smoothness and consistency of the generated trajectories and the resulting tracking quality.. Figure 6.17a shows the trajectories generated from the planner for the vehicle avoidance scenario. Each line represents one trajectory with start and end times. In the ideal case without perception and tracking errors, these trajectories should form one continuous line. During the evasive maneuver from 8 s to 13 s, we observe the effect of the perception and prediction error as well as the limited trajectory length in the orientation plot. When further away from the obstacle, the planned trajectories start the evasive maneuver earlier than necessary as the obstacle's position is estimated closer to the vehicle than it actually is. When steering back to the reference line, the planner first generates trajectories taking a wider curve than necessary as the length estimation gets more accurate when driving parallel to the obstacle. These effects can also be observed in Figure 6.14. When accelerating from standstill, the vehicle after activation of the low-level interfaces, accelerates slightly slower than the trajectory indicates. As we find the initial point for the next planning step on the old trajectory (if the tracking errors are small), the trajectories are not affected by localization or measurement errors. We observe that the velocity and orientation profiles are smooth and tracked smoothly. As the command to stop the vehicle is triggered based on the distance to the goal, one trajectory is visible that keeps the constant velocity longer. In the acceleration profile, we observe a typical noise along the reference. The curvature is computed from the steering wheel angle measurement without smoothing, rate limitations, or stabilization terms. Therefore, minimal control interactions on the steering wheel result in visible oscillations here. Still, we clearly see the evasive maneuver and an initial steering command towards the reference line. We observe very similar effects in the trajectories of the pedestrian avoidance scenario in Figure 6.17b.



(a) Vehicle avoidance scenario



(b) Pedestrian avoidance scenario



(c) Vehicle following scenario

Figure 6.17: Trajectories generated by the planner plotted in different color for different times. ◦ denotes the measurement of the respective data. For visualization clarity, only every 5th trajectory is plotted. (graphic from [KEK22], ©2022 IEEE)

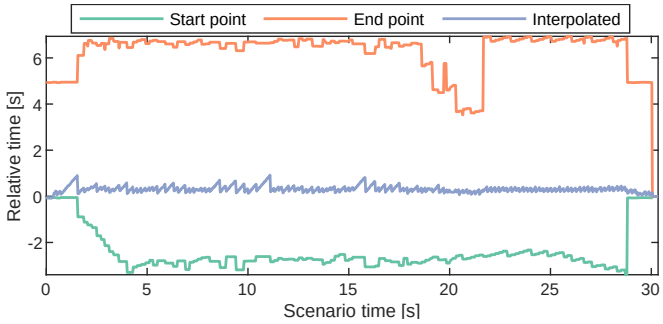


Figure 6.18: Relative start and end times of the trajectory available to the controller measured on the real-time device and the controller's interpolated (matched) time for the vehicle avoidance scenario. We observe, that always a horizon of at least 4 s is maintained. (graphic from [KEK22], ©2022 IEEE)

The analysis of the vehicle following scenario in Figure 6.17c reveals that braking trajectories are generated once the preceding vehicle intersects with the reference trajectory, and accelerating trajectories are generated if enough space in front of the vehicle is available again. As a negative side effect, we observe that the vehicle's orientation also changes with the decelerating trajectory. This effect arises as the road is not entirely straight, and we are tracking unreachable (due to collision with the other car) reference trajectory points with changed orientation. To minimize the error over the whole trajectory, the optimizer here plans an undesired slight steering maneuver. Quantitative data reveals this effect that is barely notable when driving in the car.

6.6.3 Control

In the following, we describe the tracking performance of the controller in combination with trajectory planning. We show the results exemplary for the static vehicle avoidance scenario. The controller performance in the other scenarios is comparable. In contrast to the other modules, the controller is executed on a dedicated hard real-time platform with a fixed cycle rate of 100 Hz. A trajectory with a sufficient time horizon must always be available to the controller. These trajectories are sent to the controller after each successful planning step, but our system has no guarantee that the controller receives a new trajectory in a given frequency. Therefore, we analyze if a sufficient horizon is always available to the controller.

Each trajectory has a particular duration, from some start time to an end time in the future. As the controller might need to interpolate between two trajectory points, a sufficient forward and backward horizon has to be available. Figure 6.18 shows the relative start time and end time of the trajectory available to the controller and the interpolated time of the controller for the vehicle avoidance scenario. By *interpolated*, we denote the (relative) timestep that the controller currently tracks. We generally plan a trajectory for 6 s and the planner starts planning from a start point each timestep, that is 1.25 s in the future. With a maximum solution time of 1 s for the solver and some further delay in the system, a trajectory endpoint between 6 s and 7 s is therefore expected. At around 20 s, the trajectory is cut off at an intermediate point, as all subsequent points have too low velocity and could therefore violate the non-holonomy assumption of MINIVAN. Still, a sufficiently large horizon is available. Interpolation values greater than zero indicate that a point is interpolated in the future trajectory and not only the first point in time is tracked. We observe that the interpolated timestep is never greater than 1 s. Therefore the planning system operates without delays in real-time.

Figure 6.19 shows the measured orientation, velocity, and curvature in comparison to reference signals for the vehicle avoidance scenario and Figure 6.20 shows the corresponding tracking errors in the Frenet frame of normal component e_n, tangential component e_t, orientation e_θ and velocity

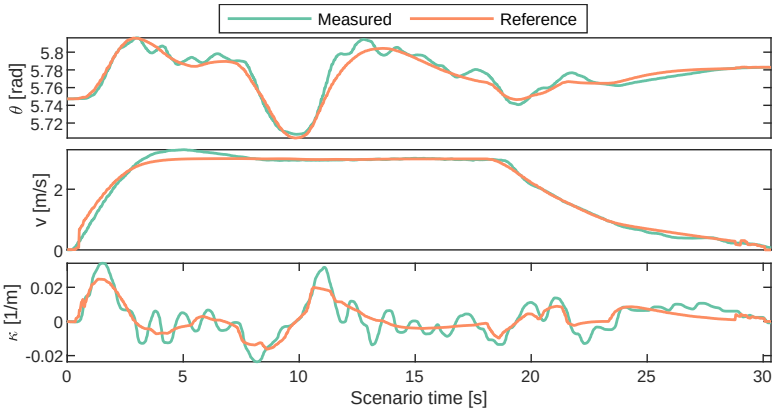


Figure 6.19: Measured orientation, velocity, and curvature in comparison to reference signals for the vehicle avoidance scenario. By reference we here denote the controller's reference, which is the planned trajectory from MINIVAN. (graphic from [KEK22], ©2022 IEEE)

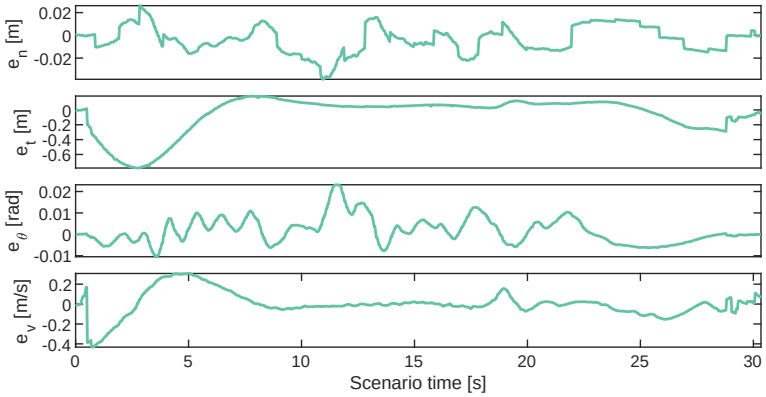


Figure 6.20: Tracking errors of normal component e_n, tangential component e_t, orientation e_θ and velocity e_v for the vehicle avoidance scenario. Huge tangential and velocity errors when the vehicle accelerates from standstill are balanced by the controller. (graphic from [KEK22], ©2022 IEEE)

e_v. We observe that the reference values from the trajectory planner are smooth but have to be stabilized by the controller in the lateral direction to achieve a slight deviation from the trajectory in the normal (lateral) direction of at most 0.02 m. Also, the orientation of the vehicle is stabilized up to an error of 0.02 rad. This stabilization results in a solid tracking of the desired position, which we omit in the figures. In tangential (longitudinal) direction, we observe that the vehicle first accelerates too slowly, which results in velocity and tangential errors that are compensated in the first seconds. Due to the technical limitations of the prototype setup, tracking very low velocities are hardly possible. Therefore we initially accelerate from a standstill with a feedforward control law without tracking the reference. The effect can be seen in the very first second of the figures. Also, the evasive maneuver at around 10 s is observed.

From this analysis, we conclude that the whole chain of modules to compute the motion, starting from the generation of the reference line, the MIQP-based trajectory planning, the MPC trajectory smoothing, and real-time trajectory tracking yields a good and stable driving performance, not only in simulation but also on the road. The trajectories generated by MINIVAN can be executed by a standard control approach regarding timing and actual values. The parameterization was chosen for tracking performance rather than smoothness. Also, the control performance shows

deficits which are not relevant here, such as the aggregation of large errors when accelerating from standstill.

6.7 Conclusion

This chapter introduced the hardware and software setup of our autonomous driving research vehicle. We described how we enhanced the open-source driving stack Apollo to operate on our retrofitted VW Passat GTE and described the software design and implementation of the replaced planning and control components as the core components under test in this work. For planning, we hooked an additional software module in the internal interfaces of Apollo's planning component that effectively pre- and post-processes planning requests to MINIVAN. Many aspects, for example, the convex decomposition of the road environment, are also valuable for other planning implementations. All implementations have proven to operate smoothly in a real-time environment. Also, the trajectory smoother, implemented as a nonlinear MPC planner effectively mitigates the drawbacks of the MINIVAN algorithm, such as the inability to generate valid trajectories at very low speeds. The implementation is general and not specific to MINIVAN and can therefore be used in combination with any behavior planner. Implementing the low-level trajectory tracking controller on a dedicated hard real-time device has in on-road driving experiments proven to mitigate the stability drawbacks of our prototype software setup running on non-real-time computers. Architecture-wise, the controller hooks to the standard middleware interfaces, making a transition to a production-grade, centralized computation platform easy.

Our driving experiments showed that the planning and control system can effectively be tested and demonstrated even with the reduced sensor setup, as the perception pipeline generates stable results at a sufficient frequency. Regarding flexibility, stability, and implementation quality, Apollo proved to be a good choice for our needs to base the software stack on; for other stacks of comparable complexity, a comparable initial hurdle is expected. Various components provide an excellent state-of-the-are baseline performance, and we did not encounter inter-component compatibility issues. The perception pipeline in this work was purely based on lidar data provided a solid performance, with varying accuracy. As this was not the focus of this work, we did not examine or mitigate these drawbacks. The prediction component, which interacts densely with the planning component, heavily relies on the quality of the map and perception data to generate valid results. The main drawbacks are the focus on a non-standardized map format (which we mitigated by a custom map converter) and a very narrow C++ development environment that hinders the out-of-the-box usage of up-to-date libraries and requires creativity from the developer. Our design decision solely based the ego localization on the differential GNSS system proved to be the primary source of instability in the system setup. We also showed the real-time capability with the chosen parameterization of the MINIVAN approach. However, We revealed that critical peaks in the runtime of the MIQP solver occur, which are handled by the remaining components of the planning and control system. Always a trajectory of at least 4 s in the future is available for the tracking controller.

7 Conclusion

This chapter concludes this thesis by summarizing the benefits of the Autonomous Car Coordination (ACCORD) and Mixed Integer Interactive Planning (MINIVAN) planning approaches and the usage of the Apollo driving stack. Section 7.1 summarizes this work and in Section 7.2, we compare both developed planning algorithms with respect to technical properties and performance. Section 7.3 assesses the research questions. Section 7.4 gives an outlook to open research questions. A final conclusion is drawn in Section 7.5.

7.1 Summary

This work introduced two novel scenario-agnostic, deterministic multi-agent behavior and motion planning and coordination algorithms to compute a cooperative vehicle motion strategy incorporating human-driven vehicles and Connected Autonomous Vehicles (CAVs). Both methods solve a dynamic game by reformulating it as a Model-Predictive Control (MPC) problem that is optimized using Mixed-Integer Progamming (MIP).

ACCORD solves the multi-agent game using a decision tree with discretized actions and continuous states. The tree is generated by stepping nonlinear agent models. An optimal set of traces through the tree is found using Mixed-Integer Linear Programming (MILP). The main focus of the approach is the coordination of connected vehicles. It also creates cooperative behavior interacting with non-communicating agents such as human-driven vehicles. The estimated intention of every agent is adopted online to fit the actual observed behaviors to handle non-cooperating agents. Behavior and motion are planned alongside, and the solution is deterministic. The complexity and runtime analysis showed in which settings the approach can be applied and where restrictions apply. The implementation can slow down the exponential growth in runtime with a growing number of agents, discrete decision options, or horizon steps but cannot avoid it. This work quantitatively showed that the runtime complexity does not scale with up to three loosely coupled agents. With an appropriate discretization, either in the number of decision options per step or the number of steps along the horizon, real-time bounds can be met regardless of the environment. The algorithm is scenario- and location-agnostic. The effectiveness and performance were demonstrated in various simulated scenarios, such as cooperative intersection management, highway merging, and bottleneck passing. This work showed a wide range of settings on different road geometries, with high and low vehicle speeds, including multiple interacting agents. To also support fully autonomous valet parking, the modified variant Autonomous Car Coordination for Valet Parking (ACCORD-P) handles direction changes, non-unique goals, and multiple references along predefined paths. The experiments showed the impact of an altruistic, cooperative, and egoistic behavior setting on the outcome of scenarios with conflicting agents' goals. A cooperation factor was integrated into the optimization that effectively scales the aggressiveness of the ego planning while conflicts with a non-cooperating vehicle are safely resolved. ACCORD handles prediction errors by altering the trace in the behavior option tree. It invests acceleration and steering savings and generates a worse but safe solution. Qualitatively, the cooperation strategies human drivers follow in traffic evolve in the proposed cooperative planning framework.

Contrary to the discrete action space of ACCORD's game tree, MINIVAN introduced an integrated Mixed-Integer Quadratic Programming (MIQP)-based formulation for the behavior planning problem on continuous action and state space. Multiple agents' behavior and motion planning in generic environments and scenarios is integrated into one linearized optimization

problem. The model shows a correct non-holonomic motion of the optimized trajectory for arbitrary road curvatures and orientations of the vehicle despite using a linear triple integrator vehicle model. Linear overapproximations of the collision shape safely avoid collisions. The algorithm is hence scenario- and location-agnostic. The multi-agent formulation leverages the interests of all agents and finds collision-free trajectories. The introduction of slack variables to the collision constraints makes the method robust against inaccuracies of the modeled future motion of other agents. The cooperation factor effectively scales the aggressiveness of the ego planning, and conflicts with a non-cooperating vehicle are safely resolved. A thorough complexity analysis and parameter study revealed that MINIVAN can be applied in real-time when interacting densely with up to around ten static or dynamic objects. Scenario-specific restrictions apply in the multi-agent formulation, and more than three interacting agents are computationally intractable in real-time. In a simulation study, MINIVAN proved to operate reliably, reproducible, deterministic, and fast, even with a high number of obstacles present. Various simulated scenarios demonstrated the effectiveness and performance of the algorithms. A wide range of experimental settings was shown on different road geometries, with high and low vehicle speeds, including static or dynamic obstacles and multiple interacting objects. Connected and unconnected settings were studied, including various levels of cooperation ranging from reaction to egoistic behavior of others to proactively enforcing own goals. The algorithm's robustness was demonstrated by introducing inaccuracies between the actual and modeled behavior of the other agent and analyzing the effect on the ego planner. MINIVAN handles these using soft constraints to generate a safe solution that is less desirable in terms of comfort and intended safety margins from the perspective of the ego vehicle.

MINIVAN was integrated into the open-source fully autonomous driving stack Apollo. This work described how the Apollo stack was adapted for the research vehicle Fortuna and the integration of MINIVAN. The modularity and flexibility of Apollo for adding new (planning) concepts were assessed. Apollo proved to be a stable, compatible, and performant baseline implementation of an autonomous driving stack that offers state-of-the-art implementations for most functionalities without posing special requirements on vehicle interfaces, vehicle hardware, software development, and organizational processes. Also, this work provided a complete toolchain from standard OpenDrive High Definition Maps (HD-Maps) to a convex, polygonal decomposition of arbitrary road environments within the Apollo stack. The contributed planning components implement a two-stage optimization approach: MINIVAN for behavior and coarse motion is followed by a smoother, realized as a nonlinear MPC-based optimizer. This trajectory smoothing method compensates for the limitations of MINIVAN that it by design produces invalid results when driving at very low velocities (less than one meter per second). The theoretic limitation that the motion is invalid for velocities lower than one meter per second is mitigated by the smoother. A suitable trajectory tracking controller fitting the vehicle and the planner was implemented on a rapid real-time prototyping system to account for stable vehicle control even with computation or communication flaws in an experimental software stack. We further shared our lessons learned while developing and working with Apollo. Three on-road benchmark scenarios showed the real-time performance of the planner in settings with static and dynamic obstacles. The MINIVAN planning approach is real-time capable, and the trajectories are kinematically valid. The system setup, including hardware, the Apollo platform, the perception pipeline, and the trajectory tracking control, proved to be performant, robust, and accurate enough for conduction driving experiments to evaluate behavior and trajectory planning research.

7.2 Comparison of MINIVAN and ACCORD

We developed two cooperative planning approaches that share properties but have individual strengths and weaknesses in this work. Table 7.1 summarizes some essential properties. As the main difference, MINIVAN is based on continuous action space, and the multi-agent planning

Table 7.1: Similarities and differences of the discrete and continuous game formulation.

	ACCORD (Chapter 4)	MINIVAN (Chapter 5)
State space	continuous	continuous
Action space	discrete	continuous
Solution method	MILP	MIQP
Non-cooperating agent integration	reflection	soft constraints
Generative model	motion primitives, single track model	triple integrator
Collision check	circle approximation	circle approximation with front axle approximation
Solution Guiding	None	last solution warmstart, priority branching
Optimality	w.r.t. motion primitives	w.r.t. vehicle model approximations
Interactivity	multi-agent planning	multi-agent planning
Cooperation	leveraged joint objective	leveraged joint objective

problem is formulated as a single mathematical optimization program. ACCORD optimizes discrete behavior options on a tree of actions separating the generation of the motion from the optimization. This inherent discretization yields limitations, as the true global optimal solution in a scenario, can be excluded from the solution space, or, if the solution space is narrow, all valid solutions can be excluded. All discretizing approaches share these effects. The discretization can be adapted to the scenario, a suitable tuning parameter to narrow the available options for planning. The possibility to generate behavior for fellow agents from an arbitrary generative model is a huge benefit that was not leveraged in this thesis but is an immediate next step. MINIVAN with its continuous action formulation, can find solutions also in very dense scenarios but introduces unnecessary complexity in scenarios that are straightforward to solve.

Both approaches are generally applicable for all traffic scenarios, and no design restrictions apply. Also, both are inherently designed to generate cooperative behavior among the agents present. Furthermore, both plan motions with respect to environmental boundaries, static and dynamic obstacles, and fellow interacting agents. Both approaches have been designed to incorporate measures to increase the robustness of the planning. MINIVAN formulates constraints between agents as soft. Therefore agents may violate non-safety critical driving restrictions if necessary, as human drivers do, depending on the aggressiveness of the behavior. ACCORD relies on a tree of possible behaviors for all agents where the optimal trace is selected, but all others, including emergency maneuvers, are present. The selection of traces is adapted via the reflection step, which synchronizes technical parameters with the observed behavior of all surrounding agents. For non-controlled agents, a tree of possible behavior options is also available. The behavioral uncertainty is addressed by the selection of different traces.

The main weakness of both approaches is the exponential runtime scaling with the number of densely interacting agents. While we in this work sketched the path towards real-time application, for general applicability, a reasonable trade-off has to be found between interactive and reactive planning with multiple obstacles. MINIVAN has the property to model the complete multi-agent behavior in one mathematical model. For this, several approximations have to be introduced, foremost the over-approximation of the collision shape of each agent, which introduces additional conservatism. Also, only linear models for the motion of an agent can be applied.

Both approaches can operate in the presence of human-driven vehicles and CAVs. The advantages and disadvantages of the approaches lead to MINIVAN favoring scenarios interacting with conventional vehicles, while ACCORD is primarily suited as a coordination approach for CAVs.

7.3 Validation of the Research Questions

In Section 1.2 we defined four research questions, that we will assess here.

1. **How can an existing open-source autonomous driving software stack be transferred to an experimental vehicle for behavior and motion planning research?** Apollo proved to be an excellent choice to base the experimental platform on due to its solid baseline performance, customizable setup, and acceptable integration effort. The performance of the overall stack is stable and powerful enough for driving experiments. It comes at the price of a relatively closed environment where developers have to live with certain restrictions, such as a fixed C++ development environment and the usage of a customized map format.
2. **Can we develop a scenario-agnostic, unified cooperative planning approach integrating behavior and trajectory planning?** No clear answer can be given here. The fact that this thesis developed two approaches already suspects that one approach does not solve every scenario with high performance. Whereas there are no theoretic restrictions in the scenario (the multi-agent game formulation can model and solve each scenario), the performance in a scenario is parameter-dependent for both approaches. When restricting the operational design domain, the analysis in this thesis reveals parameter settings that achieve reliable, fast, and high solution quality performance will be achieved in production-grade settings. Motion and behavior planning is effectively coupled, and both approaches only yield plans in the form of a kinematically valid and collision-free trajectory.
3. **Can autonomous vehicles plan cooperative behavior, like human drivers, while still achieving their own goals?** Both approaches inherently create cooperative plans that leverage the interests of all included agents while being robust to errors in other agents' predictions. Cooperative strategies of human drivers, such as lane changes on highways for letting other vehicles merge, evolve. Nevertheless, the behavior cannot be classified as human-like. The cooperative plans, especially for the coordination of CAVs, go beyond how human drivers cooperate, such as letting faster vehicles pass on the right side. New strategies evolve that use space more effectively and result in fewer state changes for every single agent.
4. **Can MIP be applied for real-time planning and, if so, which restrictions apply?** Our work demonstrated that MIP-based behavior and motion planning is capable of operating in real-time considering a sufficient number of traffic participants around the ego vehicle, both in simulation and embedded in a driving stack on a research vehicle. Using various solver tweaks, such as providing a warmstart solution, the runtime of the MIP is sufficiently fast for real-time planning. Still, peaks in the solution time occur that are critical for the overall system performance and have to be addressed in future research. The number of densely interacting agents is limited. Even with effective implementation, the problem formulation remains NP-complete, and at some point, the exponential growth in runtime impedes a real-time application. Therefore, a reasonable trade-off must be found for objects to be treated as interacting agents or dynamic obstacles.

7.4 Future Work

This section states further research directions that evolved from this thesis. The work at hand has paved the path towards an easier usage of an experimental vehicle platform by coupling it to a fully autonomous driving open-source stack. Now not only can a planning module be examined in a real-road setup, but other components can be replaced, which is an enormous benefit that should be leveraged, but this work is focused on planning. Other planners, such as ACCORD, should be integrated, compared, and tested. To also demonstrate the real-road performance of connected settings, the integration of a communication gateway model is very valuable.

In the following, we will discuss specific open research points in detail. We will first discuss possible improvements that apply to both presented planning approaches in Section 7.4.1, then discuss possibilities individual for each planner in Section 7.4.2 and Section 7.4.3. Also, in Section 7.4.4, we elaborate on possible next steps when continuing the development of an open-source-based experimental software stack.

7.4.1 Further Research Directions for ACCORD and MINIVAN

Both ACCORD and MINIVAN can be improved in the following directions:

Integrate Further Agent Types In interactive multi-agent scenarios, we discussed only the interaction of road vehicles, which is a simplification. The integration of agents whose kinematics follow a bicycle model, such as heavy-duty vehicles or bicycles, is just a matter of parameterization. For all other types of agents (e.g., pedestrians), a new motion model has to be integrated. In the case of MINIVAN, this model has to be linear. The methodology to model the agent shape as a series of circles for the collision check can remain regardless of the agent model. When considering other objects as (dynamic) obstacles, we could already show that MINIVAN can interact with pedestrians, and we are therefore convinced the integration of further agent types is possible.

Comprehensive Parameter Study When selecting the parameterization of the algorithms in this work, we relied on our in-depth knowledge of the methods to select a reasonable set of parameters. Also, variations of the cooperation factor λ have been simulated to quantify the effect of a different parameterization. However, no comprehensive parameterization strategy has been proposed to select λ, and the magnitude of all other parameters in relation to λ. Future work should add a sensitivity analysis in different scenarios to generate an in-depth understanding of the relation of the parameters and identify a parameter set with solid methodological foundation.

Scenario-adaptive Parameter Selection We could show that the parameterization of the solution algorithm can have a high influence on the solution time. Therefore, if adapted parameter sets perform best in different driving situations, an automated blending or switching of parameters will be of high benefit. Future work should also include a strategy to choose different parameter sets for different scenarios in an automated fashion based on a sensitivity analysis of the parameter set.

Improve Reference In this work, the reference line the optimizer aims to track is implemented in a very basic fashion. The reference line is just the centerline of the respective lane without any processing. Especially in obstacle avoidance scenarios, when this reference is not collision-free anymore, the convergence to a valid solution gets slower. Therefore, a more advanced reference computation could help find a smoother reference line. One possibility is the integration of a sampling-based planner, such as a rapidly-exploring random tree (RRT) [LaV06], that could find a path closed to the optimal solution fast.

Safety and Security Our algorithms do not handle security aspects at all. Also, the setup of the software stack on the prototype vehicle ignored security aspects. We assume that we do not interact with destructive agents and do not consider communicating but adversarial agents.

Regarding the safety aspects, various concepts exist nowadays, such as reachable sets [AD14] or the Responsibility-Sensitive Safety (RSS) approach [SSS17]. Both concepts can be integrated into the planning framework [Pek+20] to make sure only a motion complying with the safety concept is produced. For this, the simulation environment BARK can be a starting point for the implementation. Restricting the available solution space to a safe subset could simplify the planning problem and enhance the performance.

Performance for Multiple Interacting Agents We could, in this work, show that complex multi-agent scenarios are generally not tractable. However, we have not yet investigated performance tuning strategies besides the effective parameterization of the solver to leverage the full benefits of MINIVAN and ACCORD, the performance of the solver can be increased further. E.g., Balcan et al. [Bal+17] use machine learning to generate branching strategies for branch-and-bound trees effectively. Also, the structure, topology, and domain knowledge can be used to improve the solver performance further.

Evaluate Performance Benefits with Different Solver The optimization model is formulated in an abstract, mathematical language. Nowadays, several MIP-solvers suites are available that offer different solution strategies and parameterization options. This work demonstrated that CPLEX can solve the problem effectively but did not evaluate if other solvers offer a higher performance or a more effective user parameterization. Examples of other solvers are Gurobi [Gur21] of Google GLOP [Goo21] to name one commercial software package and one with an open-source license. As an alternative, a problem-specific customized solver implementation is possible. For example, each step depends on the previous one in the receding horizon formulation, a property that a general-purpose solver hardly utilizes.

Generate Human-like Behavior The development of the planners in this work was inspired by the implicit or explicit cooperation of human drivers. Both ACCORD and MINIVAN do not generate human-like behavior as drivers base their decisions on more aspects than cooperation. One aspect is encoding traffic rules in the planning to generate only legal behaviors [EGK20]. With driving data available, the model parameters (such as desired acceleration and steering rate) can be tuned scenario-adaptive to realistic values.

From a behavioral perspective, ACCORD and MINIVAN do not exclude kinematically valid options, so any (human-driven) trajectory can be realized. Comparing trajectories generated by the planner with trajectories driven by humans can reveal deviations. Based on these, new (hard or soft) constraints can be implemented in the model to account for human behavior.

Training Useage Imitation learning [PGP17] is a promising approach to generate behavior based on provided training data. Both ACCORD and MINIVAN can be used to generate optimal cooperative trajectories for a given scenario, which then can be used to train an imitation learning-based planner. Doing so ideally, this planner can be fast in runtime while producing similar cooperative plans as ACCORD or MINIVAN.

7.4.2 Further Research Directions for ACCORD

The following points are specific to ACCORD. It was not integrated into the Apollo stack, as the main focus was the development of a coordination approach for CAVs and our research group only maintains one prototype vehicle.

Improve Behavior Reflection The behavior reflection step that automatically tunes the weights of the MILP with respect to observations relies on a basic implementation in this thesis to evaluate the concept. The better the weights are tuned, the more the true intention of another traffic participant is reflected. Several directions are possible: A theoretically optimal assignment of costs to graph edges within a given cost range is proposed by Wilfong [Wil90], which can be applied to the problem at hand. A linear-time algorithm can find a minimum value for each scaling parameter individually when considering the costs along the actually chosen path as an upper limit for the total costs. Also, learning-based approaches are suitable for extracting a weight-changing strategy from data. Inverse reinforcement learning is an approach to learning the objective or reward from an observed agent behavior. The goal is to find a reward function that fits the data

best, relying on a Markov Decision Process [PGP17]. The work of Naumann et al. [Nau+20] analyzes how human behavior can be encoded in cost functions and applies inverse reinforcement learning to generate cost function weights based on observed trajectories. The behavior reflection approach is not exclusive to the ACCORD but can also be beneficial for other optimization-based planners.

Generative Model We span the tree of possible behaviors by sampling motion primitives using a single-track vehicle model. This strategy exploits the available space well in unstructured and semi-structured environments, but in structured environments, the performance depends on the chosen sampling discretization. With a narrow discretization, a variety of behavior options is available at the price of a high runtime. Choosing a wide discretization of the runtime is fast, but the true optimal solution can be excluded. A compromise between performance and evaluation can be to populate the behavior option graph with high-level actions, e.g., to follow a lane for one second or to perform a lane change in a time interval of 5 s. These trajectories can be efficiently generated in a street-local coordinate system (Frenet frame). The subsequent steps of the ACCORD algorithm can be applied without modifications.

Handling of Dynamic Obstacles ACCORD was designed to operate in static environments, and every moving object is considered as an agent, for which we coordinate behavior and a resulting motion. To also account for dynamic obstacles with fixed and unchangeable predicted trajectories that the ego agent shall avoid without performing a joint action planning, the tree of behavior options can simply be reused without algorithmic changes. Instead of sampling motion options for an agent, we insert a new agent in the model with a new behavior option tree. This tree only contains exactly one trace starting for the current position. Vertices in the graph are created matching the trajectory prediction for this moving object. With this extension, also purely reactive maneuvers are possible for ACCORD and a basic integration into a full self-driving stack is easily possible.

Augment Behavior Graph States Currently, the nodes of the behavior option graph only store the geometry and dynamics information of the vehicles. These are used for collision checking. Without adding complexity, additional information can be stored and processed, e.g., the compliance to a set of traffic rules [EGK20]. By doing so, the computed cooperative maneuvers will fit better into traffic. Examples of further information to persist and optimize for would be gap distances in a platoon or traffic light phases.

7.4.3 Further Research Directions for MINIVAN

The following points are specific to MINIVAN.

Warmstarting We discussed that warmstarting the optimization is essential for real-time execution. With in-depth solver knowledge, the performance can be further optimized. CPLEX offers several heuristics to improve a given, valid solution. Such features were not examined in the current model. Also, recent works exist that aim to learn an (initial) solution of a MIP model, and afterward use the mixed-integer model formulation to prove optimality using a few steps [Cau+20]. Here, the combinatorial part of the problem is transferred into a learning step that is executed offline. During runtime, only a small optimization is performed with comparable solution quality.

Environment Decomposition Road geometries are usually non-convex and therefore have to be decomposed into several convex sub-polygons. Our implemented algorithm relies on a triangulation based on a Voronoi diagram and a subsequent merging step of the triangles in a greedy fashion. This algorithm can result in suboptimal decomposition with too many polygons,

significantly decreasing the runtime. A more advanced convexification algorithm, obtaining a minimal set of convex sub-polygons, can speed up the optimization. Also, knowledge of the topology of the geometry can be used to reduce the number of polygons.

Fitting We can trade accuracy for runtime with the number of regions in the polygonal fits. Few regions result in a fast model with few integer variables and constraints. More regions yield a better approximation of the vehicle shape and a better approximation of the nonlinear terms. Not only the number of regions but also the minimal and maximal speed in the fitting process control the accuracy and complexity of the fit.

The fitting model cannot be changed during runtime in the current implementation. As the model is executed in a receding-horizon fashion, it is possible to exchange the fit from one timestep to another to use the best-suited fit in a scenario-adaptive fashion. With this strategy, the trade-off between accuracy and performance could be chosen dynamically and situation-adaptive.

Verification Usage The proposed model was developed for online usage as a planning component of an autonomous vehicle. When changing the parameterization from a low computation time to prove optimality, the model can also be applied as a tool for offline verification of behavior plans as it can rate how far a given trajectory is away from the global optimal solution.

Pre-processing of the Planning Problem For an application in generic scenarios with an arbitrary number of interacting traffic participants, the planning problem has to be efficiently pre-processed to avoid including unnecessary objects. While in this work, a simple Field of View (FOV) filter was implemented, a more advanced strategy, e.g., taking the road geometry into account, would be beneficial. The model formulation can handle multiple agents with an own intention, (dynamic) obstacles that follow a fixed prediction, or both. A trade-off must be found which object to denote as an obstacle and which to handle as an interacting agent. The possible number of interacting agents is limited, whereas a high number of obstacles can be handled in real-time. Truly interactive maneuvers can only be planned in the multi-agent formulation.

7.4.4 Further Research Directions in Applying Open-Source Software Stacks

Usage of HD-Maps, Map Generation, and Localization A main technical issue in this work was the robust generation and maintenance of the HD-Map and a reliable localization within this map using a differential Global Navigation Satellite System (GNSS). To ease the integration of effort when extending to new locations, algorithms should be developed and integrated that generate a map representation online, either using fused sensor data [Kes+18], lidar pointclouds, or raw video footage. Also, the ego localization should not be purely based on GNSS but on other sensor data, such as lidar, as even a high-end device could, in numerous situations, not provide robust enough localization data.

Benchmark Apollo vs. Autoware.Auto The design decision to base the software stack on Apollo dates back to the year 2018. When the performance of the Autoware platform reaches the mature performance of Apollo, this has to be re-evaluated, also concerning the openness of system design and ecosystem. As of now, 2021, Autoware.Auto has caught up and also claims to offer a good baseline performance. Still, the operational design domain is limited to valet parking. Due to the more open ecosystem of Autoware, this stack is appealing for research like in this thesis. A quantitative comparison of the performance of both stacks would be highly beneficial for the research community. A qualitative analysis of the effort to integrate additional components and adapt the stack to a custom prototype vehicle compared to the procedure described here for the Apollo driving stack is needed.

Validate Methodology by Integrating a Second Planner MINIVAN operates seamlessly in the Apollo driving stack. It is suspected that the same methodology and source code can be applied to integrate a second planner, such as ACCORD. If this second planner is based on a different technology than mixed-integer optimization, e.g., a learning-based approach, our framework and software interface would be validated to be generally applicable.

Application in a Connected Car Setting For this research, one autonomous vehicle prototype was available. To demonstrate the applicability of the proposed planners for groups of CAVs, the transfer of the stack to another vehicle is necessary and the setup of a connected car setting. As we did for the perception pipeline, a stable minimal realistic communication setup is needed. The question has to be answered how to include minimal additional complexity in the system while still being able to demonstrate realistic scenarios.

7.5 Concluding Remarks

The result of this thesis is twofold. First, based on mathematically sound formulations, this thesis developed two general, scenario-agnostic optimal cooperative planning algorithms using MIP. The resulting trajectories are kinematically valid, collision-free, and without any form of randomness. This property is undoubtedly a favorable feature for validation and certification. The introduced cooperation factor effectively leverages the level of cooperation among groups of autonomous or non-autonomous vehicles. Both model formulations consider model, perception, and prediction errors of objects interacting with the autonomous vehicle. This thesis showed that the implementations are real-time capable with a limited number of agents, and the complexity analysis revealed where exponential scaling can be avoided. With the models developed in this work, it is now possible to generate not only reactive but truly cooperative behavior for autonomous vehicles based on mathematical optimization, resulting in fair, smooth, safe, valid, and admissible trajectories.

Second, this thesis brought the third-party, open-source autonomous driving stack Apollo to a full-size German vehicle experimental platform. The additional source code is also published as open-source software [EK21]. This work documented every implementation step, assessed the strengths, benefits, and shortcomings of this setup, and elaborated the lessons learned. These extensions and experiences will allow research groups working on a similar vehicle setup not to have to adapt Apollo from scratch. Real-road driving experiments showed the applicability of the full software stack, especially for evaluating behavior and motion planning algorithms. The applied procedure is transferable to other autonomous driving stacks. With the description of the hard- and software setup, the developed methodology, and open-source software code, this work aims to enable other research groups to demonstrate their algorithms in reality easier to reveal shortcomings and catch up to companies in the field.

As of 2022, production-grade highly or fully automated driving has often been promised but never reached European roads. By evaluating research algorithms in closed-loop with the real world, one can accelerate this development by making research results easier transferable into an industrial context. MIP-based cooperative planning, as an evolution of nowadays human driver behavior, can be one part of the unstoppable transformation process from manual to autonomous driving – probably the most significant change in traffic history since the replacement of horses by cars.

List of Figures

List of Tables

Acronyms

ACC Adaptive Cruise Control 25, 117, 119, 121

ACCORD Autonomous Car Coordination vii, 5, 10, 31, 32, 42–44, 51, 52, 58, 63, 143, 145–149, 151

ACCORD-P Autonomous Car Coordination for Valet Parking vii, 42, 43, 50–52, 64, 143

API Application Programming Interface 27, 42, 89, 90, 133

CAN Car Area Network 116, 118, 119, 121, 130, 132–134

CAV Connected Autonomous Vehicle 1, 3, 10, 11, 20, 31, 32, 39, 51, 102, 143, 145, 146, 148, 151

DARPA Defense Advanced Research Projects Agency 15, 16, 24–26

FOV Field of View 117, 118, 150

GCDC Grand Cooperative Driving Challenge 24, 25

GMSL Gigabit Multimedia Serial Link 118

GNSS Global Navigation Satellite System 25, 26, 117, 119, 134, 142, 150

HD-Map High Definition Map 26, 121, 122, 124, 133, 134, 144, 150

IMU Inertial Measurement Unit 26, 119, 134

INS Inertial Navigation System 117, 119

MCTS Monte Carlo Tree Search 17, 21

MILP Mixed-Integer Linear Programming 4, 13, 22–24, 31, 41, 43, 63, 143, 145, 148

MINIVAN Mixed Integer Interactive Planning 5, 10, 65, 66, 81–83, 85, 86, 90, 102–107, 110, 111, 113, 115, 123–129, 133, 134, 136, 140–145, 147–149, 151

MIP Mixed-Integer Progamming 1–5, 7, 12–15, 22–24, 41, 42, 46, 65, 66, 86, 88, 89, 91, 143, 146, 148, 149, 151

MIQP Mixed-Integer Quadratic Programming 4, 13, 17, 21–23, 65, 66, 69–71, 80–82, 89, 94, 97–101, 113, 123, 129, 137, 138, 141–143, 145

MPC Model-Predictive Control 4, 19, 20, 32, 115, 129, 132, 141–144

POMDP Partially Observable Markov Decision Process 19, 21

QP Quadratic Programming 22–24, 28, 29

ROI Region of Interest 125, 133

ROS Robot Operating System 26, 27, 122, 133

RTK Real-Time Kinematic 25, 117, 119

SQP Sequential Quadratic Programming 23, 24, 82, 83

V2V Vehicle-to-Vehicle 1, 9, 20, 25, 43

V2X Vehicle-to-Everything 9, 15, 20, 25, 58, 116

Bibliography

[AD14] M. Althoff and J. M. Dolan. "Online verification of automated road vehicles using reachability analysis." In: *IEEE Transactions on Robotics* 30.4 (2014), pp. 903–918 (cit. on p. 147).

[AD18] F. Altché and A. De La Fortelle. "Partitioning of the free space-time for on-road navigation of autonomous ground vehicles." In: *2017 IEEE 56th Annual Conference on Decision and Control, CDC 2017* (2018), pp. 2126–2133. arXiv: 1801.07961 (cit. on pp. 17, 24).

[AFV18] F. Ashtiani, S. A. Fayazi, and A. Vahidi. "Multi-Intersection Traffic Management for Autonomous Vehicles via Distributed Mixed Integer Linear Programming." In: *2018 Annual American Control Conference (ACC)*. 2018, pp. 6341–6346 (cit. on p. 24).

[AKM17] M. Althoff, M. Koschi, and S. Manzinger. "CommonRoad: Composable benchmarks for motion planning on roads." In: *2017 IEEE Intelligent Vehicles Symposium (IV)*. IEEE, June 2017, pp. 719–726 (cit. on pp. 26, 58).

[Ala+16] A. Alahi, K. Goel, V. Ramanathan, A. Robicquet, F.-F. Li, and S. Savarese. "Social LSTM : Human Trajectory Prediction in Crowded Spaces ." In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, pp. 961–971 (cit. on p. 120).

[Ara+15] M. Aramrattana, T. Larsson, J. Jansson, and C. Englund. "Dimensions of cooperative driving, its and automation." In: *2015 IEEE Intelligent Vehicles Symposium (IV)*. 2015, pp. 144–149 (cit. on p. 20).

[Ara+18] M. Aramrattana, J. Detournay, C. Englund, V. Frimodig, O. U. Jansson, T. Larsson, W. Mostowski, V. D. Rodriguez, T. Rosenstatter, and G. Shahanoor. "Team Halmstad Approach to Cooperative Driving in the Grand Cooperative Driving Challenge 2016." In: *IEEE Transactions on Intelligent Transportation Systems* 19.4 (2018), pp. 1248–1261 (cit. on pp. 20, 24, 25).

[ASA21] ASAM e. V. *ASAM OpenDrive*. https://www.asam.net/standards/detail/opendrive. 2021 (accessed September 30, 2021) (cit. on p. 121).

[Aut21a] Autoware Foundation. *Autoware.AI*. https://www.autoware.org/autoware-ai. 2021 (accessed September 30, 2021) (cit. on pp. 15, 26).

[Aut21b] Autoware Foundation. *Autoware.Auto*. https://www.autoware.org/autoware-auto, 2021 (accessed September 30, 2021) (cit. on p. 26).

[Bah+16] M. Bahram, A. Lawitzky, J. Friedrichs, M. Aeberhard, and D. Wollherr. "A Game-Theoretic Approach to Replanning-Aware Interactive Scene Prediction and Planning." In: *IEEE Transactions on Vehicular Technology* 65.6 (2016) (cit. on pp. 17, 18).

[Bai17] Baidu. *Apollo Auto: An open autonomous driving platform*. https://apollo.auto/. 2017 (cit. on pp. 3, 15, 25, 26, 118).

[Bai21a] Baidu. *The Apollo Planning Module*. https://github.com/fortiss/apollo/blob/r5.5.0/modules/planning/README.md. 2020 (accessed September 8, 2021) (cit. on p. 28).

[Bai21b] Baidu. *The Apollo Kernel*. https://github.com/ApolloAuto/apollo-kernel/blob/master/README.md. 2017 (accessed September 8, 2021) (cit. on p. 28).

[Bai21c] Baidu. *Apollo 3d obstacle perception*. `https://github.com/fortiss/apollo/blob/r5.5.0/docs/specs/3d_obstacle_perception.md`. 2019 (accessed September 8, 2021) (cit. on p. 120).

[Bai21d] Baidu. *Introduction into Cyber RT*. `https://github.com/fortiss/apollo/blob/r5.5.0/cyber/README.md`. 2019 (accessed September 8, 2021) (cit. on p. 27).

[Bai21e] Baidu. *The Apollo Perception module*. `https://github.com/fortiss/apollo/blob/r5.5.0/modules/perception/README.md`. 2019 (accessed September 8, 2021) (cit. on p. 120).

[Bai21f] Baidu. *The Apollo Prediction module*. `https://github.com/fortiss/apollo/blob/r5.5.0/modules/prediction/README.md`. 2019 (accessed September 8, 2021) (cit. on p. 120).

[Bai21g] Baidu. *Apllo: An open autonomous driving platform*. `https://github.com/ApolloAuto/apollo`. 2021 (accessed September 30, 2021) (cit. on p. 27).

[Bal+17] M.-F. Balcan, T. Dick, T. Sandholm, and E. Vitercik. "Learning to Branch." In: *Proceedings of the 35th International Conference on Machine Learning*. 2017 (cit. on p. 148).

[Bas+16] H. Bast, D. Delling, A. Goldberg, M. Müller-Hannemann, T. Pajor, P. Sanders, D. Wagner, and R. F. Werneck. "Route Planning in Transportation Networks." In: *Algorithm Engineering*. Vol. LNCS 9220. 2016, pp. 19–80. arXiv: `1504.05140` (cit. on p. 16).

[Ben+15] P. Bender, O. S. Tas, J. Ziegler, and C. Stiller. "The combinatorial aspect of motion planning: Maneuver variants in structured environments." In: *IEEE Intelligent Vehicles Symposium, Proceedings*. 2015, pp. 1386–1392 (cit. on pp. 17, 24).

[Ber+20] J. Bernhard, K. Esterle, P. Hart, and T. Kessler. "BARK: Open Behavior Benchmarking in Multi-Agent Environments." In: *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2020, pp. 6201–6208 (cit. on pp. 4, 90, 122, 133).

[Bha+20] R. P. Bhattacharyya, B. Wulfe, D. J. Phillips, A. Kuefler, J. Morton, R. Senanayake, and M. J. Kochenderfer. "Modeling Human Driving Behavior through Generative Adversarial Imitation Learning." In: *CoRR* abs/2006.06412 (2020). arXiv: `2006.06412` (cit. on p. 19).

[Bix12] R. E. Bixby. "A Brief History of Linear and Mixed-Integer Programming Computation." In: *Documenta Mathematica · Extra* ISMP (2012), pp. 107–121 (cit. on p. 13).

[BL18] C. Burger and M. Lauer. "Cooperative Multiple Vehicle Trajectory Planning using MIQP." In: *21st IEEE International Conference on Intelligent Transportation Systems (ITSC)*. 2018, pp. 602–607 (cit. on pp. 18, 22, 69, 84).

[BO98] T. Başar and G. J. Olsder. *Dynamic Noncooperative Game Theory*. Society for Industrial and Applied Mathematics, 1998 (cit. on p. 11).

[BOW11] L. Blackmore, M. Ono, and B. C. Williams. "Chance-constrained optimal path planning with obstacles." In: *IEEE Transactions on Robotics* 27.6 (2011), pp. 1080–1094 (cit. on p. 22).

[BPK19] J. Bernhard, S. Pollok, and A. Knoll. "Addressing Inherent Uncertainty: Risk-Sensitive Behavior Generation for Automated Driving using Distributional Reinforcement Learning." In: *2019 IEEE Intelligent Vehicles Symposium (IV)*. 2019, pp. 2148–2155 (cit. on p. 19).

[Bue+15] M. Buechel, J. Frtunikj, K. Becker, S. Sommer, C. Buckl, M. Armbruster, A. Marek, A. Zirkler, C. Klein, and A. Knoll. "An Automated Electric Vehicle Prototype Showing New Trends in Automotive Architectures." In: *IEEE Conference on Intelligent Transportation Systems, Proceedings, ITSC* (2015), pp. 1274–1279 (cit. on p. 25).

[Bue+19] M. Buechel, M. Schellmann, H. Rosier, T. Kessler, and A. Knoll. "Fortuna: Presenting the 5G-connected automated vehicle prototype of the project PROVIDENTIA." In: *prePrint* (2019) (cit. on pp. 4, 24, 118).

[Cal+18] L. Caltagirone, M. Bellone, L. Svensson, and M. Wahde. "LIDAR-based driving path generation using fully convolutional neural networks." In: *IEEE Conference on Intelligent Transportation Systems, Proceedings, ITSC* (2018), pp. 1–6. arXiv: 1703.08987 (cit. on p. 20).

[Cau+20] A. Cauligi, P. Culbertson, B. Stellato, D. Bertsimas, M. Schwager, and M. Pavone. "Learning Mixed-Integer Convex Optimization Strategies for Robot Planning and Control." In: *Proceedings of the IEEE Conference on Decision and Control*. 2020, pp. 1698–1705. arXiv: 2004.03736 (cit. on p. 149).

[CBD11] D.-S. Chen, R. G. Batson, and Y. Dang. *Applied Integer Programming: Modeling and Solution*. 2011 (cit. on pp. 12, 36).

[CE16] L. Chen and C. Englund. "Cooperative Intersection Management: A Survey." In: *IEEE Transactions on Intelligent Transportation Systems* 17.2 (2016), pp. 570–586 (cit. on p. 20).

[Cla+20] L. Claussmann, M. Revilloud, D. Gruyer, and S. Glaser. "A Review of Motion Planning for Highway Autonomous Driving." In: *IEEE Transactions on Intelligent Transportation Systems* 21.5 (2020), pp. 1826–1848 (cit. on p. 15).

[DGB17] K. Driggs-Campbell, V. Govindarajan, and R. Bajcsy. "Integrating Intuitive Driver Models in Autonomous Planning for Interactive Maneuvers." In: *IEEE Transactions on Intelligent Transportation Systems* 18.12 (2017), pp. 3461–3472 (cit. on p. 19).

[Dol+08] D. Dolgov, S. Thrun, M. Montemerlo, and J. Diebel. "Practical search techniques in path planning for autonomous driving." In: *Proceedings of the First International Symposium on Search Techniques in Artificial Intelligence and Robotics (STAIR-08)* (2008), pp. 32–37 (cit. on p. 16).

[Dos+17] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun. "CARLA: An Open Urban Driving Simulator." In: *Proceedings of the 1st Annual Conference on Robot Learning*. 2017, pp. 1–16. arXiv: 1711.03938 (cit. on p. 122).

[DP14] M. Düring and P. Pascheka. "Cooperative decentralized decision making for conflict resolution among autonomous agents." In: *IEEE International Symposium on Innovations in Intelligent Systems and Applications (INISTA)*. 2014, pp. 154–161 (cit. on pp. 18, 20, 55).

[EGK20] K. Esterle, L. Gressenbuch, and A. Knoll. "Formalizing traffic rules for machine interpretability." In: *2020 IEEE 3rd Connected and Automated Vehicles Symposium, CAVS 2020 - Proceedings* (2020). arXiv: 2007.00330 (cit. on pp. 148, 149).

[EK21] K. Esterle and T. Kessler. *Adaptions from fortiss to make Apollo work on Fortuna*. https://github.com/fortiss/apollo. 2021 (accessed September 30, 2021) (cit. on pp. 118, 151).

[EKK20] K. Esterle, T. Kessler, and A. Knoll. "Optimal Behavior Planning for Autonomous Driving: A Generic Mixed-Integer Formulation." In: *2020 IEEE Intelligent Vehicles Symposium (IV)*. 2020, pp. 1914–1921 (cit. on pp. 4, 15, 66–70, 75, 83, 85, 103).

[Ele21] Elektrobit. *EB robinos Automated driving software*. `https://www.elektrobit.com/products/automated-driving/eb-robinos/`. 2021 (accessed September 30, 2021) (cit. on p. 27).

[ES17] J. Eilbrecht and O. Stursberg. "Cooperative driving using a hierarchy of mixed-integer programming and tracking control." In: *2017 IEEE Intelligent Vehicles Symposium (IV)*. 2017, pp. 673–678 (cit. on pp. 18, 21, 22, 69).

[Est+18] K. Esterle, P. Hart, J. Bernhard, and A. Knoll. "Spatiotemporal Motion Planning with Combinatorial Reasoning for Autonomous Driving." In: *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*. 2018, pp. 1053–1060 (cit. on pp. 17, 23, 28).

[FB11] C. Frese and J. Beyerer. "A Comparison of Motion Planning Algorithms for Cooperative Collision Avoidance of Multiple Cognitive Automobiles." In: *IEEE Intelligent Vehicles Symposium (IV)*. 2011, pp. 1156–1162 (cit. on pp. 18, 22, 75).

[Fer+21] T. Fernando, S. Denman, S. Sridharan, and C. Fookes. "Deep Inverse Reinforcement Learning for Behavior Prediction in Autonomous Driving: Accurate Forecasts of Vehicle Motion." In: *IEEE Signal Processing Magazine* 38.1 (2021), pp. 87–96 (cit. on p. 19).

[FG20] F. Fabiani and S. Grammatico. "Multi-vehicle automated driving as a generalized mixed-integer potential game." In: *IEEE Transactions on Intelligent Transportation Systems* 21.3 (2020), pp. 1064–1073 (cit. on pp. 17, 18).

[For86] S. Fortune. "A Sweepline Algorithm for Voronoi Diagrams." In: *Proceedings of the Second Annual Symposium on Computational Geometry*. 1986, pp. 313–322 (cit. on p. 126).

[Fre+20] D. J. Fremont, E. Kim, Y. V. Pant, S. A. Seshia, A. Acharya, X. Bruso, P. Wells, S. Lemke, Q. Lu, and S. Mehta. "Formal Scenario-Based Testing of Autonomous Vehicles: From Simulation to the Real World." In: *2020 IEEE 23rd International Conference on Intelligent Transportation Systems (ITSC)* (2020) (cit. on p. 26).

[Fre12] C. Frese. *Planung kooperativer Fahrmanöver für kognitive Automobile*. 2012 (cit. on p. 22).

[FVL17] S. A. Fayazi, A. Vahidi, and A. Luckow. "Optimal scheduling of autonomous vehicle arrivals at intelligent intersections via MILP." In: *2017 American Control Conference (ACC)*. 2017, pp. 4920–4925 (cit. on p. 24).

[GC13] Y. Geng and C. G. Cassandras. "New "Smart Parking" System Based on Resource Allocation and Reservations." In: *IEEE Transactions on Intelligent Transportation Systems* 14.3 (2013), pp. 1129–1139 (cit. on p. 24).

[Geh+21] B. Gehrels, B. Lalande, M. Loskot, and A. Wulkiewicz. *Boost Geometry (Generic Geometry Library, GGL)*. `https://www.boost.org/doc/libs/1_72_0/libs/geometry/doc/html/index.html`. 2021 (accessed September 30, 2021) (cit. on p. 42).

[GGW17] B. Gutjahr, L. Gröll, and M. Werling. "Lateral Vehicle Trajectory Optimization Using Constrained Linear Time-Varying MPC." In: *IEEE Transactions on Intelligent Transportation Systems* 18.6 (2017) (cit. on p. 79).

[Gon+16] D. González, J. Pérez, V. Milanés, and F. Nashashibi. "A Review of Motion Planning Techniques for Automated Vehicles." In: *IEEE Transactions on Intelligent Transportation Systems* 17.4 (2016), pp. 1135–1145 (cit. on p. 15).

[Goo21] Google Operations Research. *GLOP*. `https://developers.google.com/optimization/`. 2021 (accessed September 30, 2021) (cit. on p. 148).

[Gra+21] M. J. Graf, O. M. Speidel, J. Ruof, and K. Dietmayer. "On-Road Motion Planning for Automated Vehicles at Ulm University." In: *IEEE Intelligent Transportation Systems Magazine* (2021), pp. 2–12 (cit. on p. 2).

[Gra19] C. W. Gran. *HD-Maps in Autonomous Driving*. 2019 (cit. on p. 122).

[GS21] J. D. Gammell and M. P. Strub. "Asymptotically Optimal Sampling-Based Motion Planning Methods." In: *Annual Review of Control, Robotics, and Autonomous Systems* 4.1 (2021), pp. 295–318 (cit. on p. 16).

[Gur21] Gurobi Optimization. *Gurobi*. `https://www.gurobi.com/`. 2021 (accessed September 30, 2021) (cit. on p. 148).

[Gut+16] B. Gutjahr, C. Pek, L. Gröll, and M. Werling. "Recheneffiziente Trajektorienoptimierung für Fahrzeuge mittels quadratischem Programm." In: *At-Automatisierungstechnik* 64.10 (2016), pp. 786–794 (cit. on pp. 17, 23).

[HCL19] M. Henaff, A. Canziani, and Y. LeCun. "Model-Predictive Policy Learning with Uncertainty Regularization for Driving in Dense Traffic." In: *CoRR* abs/1901.02705 (2019). arXiv: `1901.02705` (cit. on p. 19).

[He+21] R. He, J. Zhou, S. Jiang, Y. Wang, J. Tao, S. Song, J. Hu, J. Miao, and Q. Luo. "TDR-OBCA: A Reliable Planner for Autonomous Driving in Free-Space Environment." In: *Proceedings of the American Control Conference* (2021), pp. 2927–2934. arXiv: `2009.11345` (cit. on pp. 26, 29).

[Hei+20] A. Heilmeier, A. Wischnewski, L. Hermansdorfer, J. Betz, M. Lienkamp, and B. Lohmann. "Minimum curvature trajectory planning and control for an autonomous race car." In: *Vehicle System Dynamics* 58.10 (2020), pp. 1497–1527 (cit. on p. 110).

[HRK19] P. Hart, L. Rychly, and A. Knoll. "Lane-Merging Using Policy-based Reinforcement Learning and." In: *2019 IEEE Intelligent Transportation Systems Conference (ITSC)* (2019), pp. 3176–3181 (cit. on p. 17).

[Hub+17] 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: *IEEE Intelligent Vehicles Symposium, Proceedings* Iv (2017), pp. 1671–1678 (cit. on pp. 18, 19).

[Hub+18] C. Hubmann, J. Schulz, G. Xu, D. Althoff, and C. Stiller. "A Belief State Planner for Interactive Merge Maneuvers in Congested Traffic." In: *IEEE Conference on Intelligent Transportation Systems, Proceedings, ITSC* (2018), pp. 1617–1624 (cit. on pp. 18, 19).

[Int21a] International Business Machines Corporation (IBM). *ILOG CPLEX Optimization Studio 12.9.0 Documentation*. 2021 (cit. on p. 12).

[Int21b] International Business Machines Corporation (IBM). *ILOG CPLEX Optimization Studio*. `https://www.ibm.com/de-de/products/ilog-cplex-optimization-studio`. 2021 (accessed September 30, 2021) (cit. on pp. 42, 89).

[KA17] M. Koschi and M. Althoff. "SPOT: A tool for set-based prediction of traffic participants." In: *IEEE Intelligent Vehicles Symposium, Proceedings* (2017), pp. 1686–1693 (cit. on p. 58).

[Kat+18] S. Kato, S. Tokunaga, Y. Maruyama, S. Maeda, M. Hirabayashi, Y. Kitsukawa, A. Monrroy, T. Ando, Y. Fujii, and T. Azumi. "Autoware on Board: Enabling Autonomous Vehicles with Embedded Systems." In: *Proceedings - 9th ACM/IEEE International Conference on Cyber-Physical Systems, ICCPS 2018* (2018), pp. 287–296. arXiv: `1112.5154` (cit. on pp. 3, 25, 26).

[KEK20] T. Kessler, K. Esterle, and A. Knoll. "Linear Differential Games for Cooperative Behavior Planning of Autonomous Vehicles Using Mixed-Integer Programming." In: *2020 59th IEEE Conference on Decision and Control (CDC)*. 2020, pp. 4060–4066 (cit. on pp. 4, 15, 66, 78, 104, 110, 112, 113).

[KEK22] T. Kessler, K. Esterle, and A. Knoll. "Mixed-Integer Motion Planning on German Roads within the Apollo Driving Stack." In: *Accepted for publication in the IEEE Transactions of Intelligent Vehicles* (2022). In reference to IEEE copyrighted material which is used with permission in this thesis, the IEEE does not endorse any of TUM's products or services. Internal or personal use of this material is permitted. If interested in reprinting/republishing IEEE copyrighted material for advertising or promotional purposes or for creating new collective works for resale or redistribution, please go to `http://www.ieee.org/publications_standards/publications/rights/rights_link.html` to learn how to obtain a License from RightsLink. If applicable, University Microfilms and/or ProQuest Library, or the Archives of Canada may supply single copies of the dissertation. (cit. on pp. 4, 15, 115, 119, 123, 127, 128, 135–141).

[Ken+19] A. Kendall, J. Hawke, D. Janz, P. Mazur, D. Reda, J. M. Allen, V. D. Lam, A. Bewley, and A. Shah. "Learning to drive in a day." In: *Proceedings of the IEEE International Conference on Robotics and Automation* (2019), pp. 8248–8254. arXiv: `1807.00412` (cit. on p. 20).

[Kes+17] T. Kessler, P. Minnerup, D. Lenz, and A. Knoll. "Systematically comparing control approaches in the presence of actuator errors." In: *IEEE Intelligent Vehicles Symposium, Proceedings*. 2017, pp. 353–358 (cit. on pp. 4, 131).

[Kes+18] T. Kessler, P. Minnerup, K. Esterle, C. Feist, F. Mickler, E. Roth, and A. Knoll. "Road-graph Generation and Free-Space Estimation in Unknown Structured Environments for Autonomous Vehicle Motion Planning." In: *21st IEEE International Conference on Intelligent Transportation Systems (ITSC)*. 2018, pp. 2831–2838 (cit. on pp. 4, 122, 150).

[Kes+19] T. Kessler, J. Bernhard, M. Buechel, K. Esterle, P. Hart, D. Malovetz, M. Truong Le, F. Diehl, T. Brunner, and A. Knoll. "Bridging the Gap between Open Source Software and Vehicle Hardware for Autonomous Driving." In: *2019 IEEE Intelligent Vehicles Symposium (IV)*. 2019, pp. 1612–1619 (cit. on pp. 4, 15, 25, 115, 117, 131).

[KEZ18] K. Kurzer, F. Engelhorn, and J. M. Zöllner. "Decentralized Cooperative Planning for Automated Vehicles with Continous Monte Carlo Tree Search." In: *21st International Conference on Intelligent Transportation Systems (ITSC)*. 2018, pp. 452–459. arXiv: `1807.09530` (cit. on pp. 18, 21).

[KF11] S. Karaman and E. Frazzoli. "Sampling-based algorithms for optimal motion planning." In: *International Journal of Robotics Research* 30.7 (2011), pp. 846–894 (cit. on p. 16).

[KK17] T. Kessler and A. Knoll. "Multi Vehicle Trajectory Coordination for Automated Parking." In: *IEEE Intelligent Vehicles Symposium (IV)*. 2017, pp. 661–666 (cit. on pp. 4, 15, 31, 42, 50, 62, 63).

[KK19] T. Kessler and A. Knoll. "Cooperative Multi-Vehicle Behavior Coordination for Autonomous Driving." In: *IEEE Intelligent Vehicles Symposium (IV)*. 2019 (cit. on pp. 4, 8, 15, 31, 32, 34, 36, 39, 116).

[Krä+22] A. Krämmer, C. Schöller, D. Gulati, and A. Knoll. "Providentia–A Large Scale Sensing System for the Assistance of Autonomous Vehicles and Its Evaluation." In: *J. Field Robot.* 2 (2022), pp. 1156–1176 (cit. on pp. 1, 58).

[KZ86] K. Kant and S. W. Zucker. "Toward Efficient Trajectory Planning: The Path-Velocity Decomposition." In: *International Journal of Robotics Research* 5.3 (1986), pp. 72–89 (cit. on p. 16).

[KZZ18] K. Kurzer, C. Zhou, and J. M. Zöllner. "Decentralized Cooperative Planning for Automated Vehicles with Hierarchical Monte Carlo Tree Search." In: *IEEE Intelligent Vehicles Symposium (IV)*. 2018, pp. 529–536. arXiv: 1809.03200 (cit. on pp. 18, 21).

[LaV06] S. LaValle. *Planning Algorithms*. Cambridge University Press, 2006 (cit. on pp. 11, 42, 43, 121, 147).

[Len+17] D. Lenz, F. Diehl, M. T. Le, and A. Knoll. "Deep neural networks for Markovian interactive scene prediction in highway scenarios." In: (2017), pp. 685–692 (cit. on pp. 16, 19).

[Lev+11] J. Levinson, J. Askeland, J. Becker, J. Dolson, D. Held, S. Kammel, J. Z. Kolter, D. Langer, O. Pink, V. Pratt, M. Sokolsky, G. Stanek, D. Stavens, A. Teichman, M. Werling, and S. Thrun. "Towards fully autonomous driving: Systems and algorithms." In: *IEEE Intelligent Vehicles Symposium, Proceedings* (2011), pp. 163–168 (cit. on p. 24).

[LGW18] B. Lehmann, H.-J. Günther, and L. Wolf. "A Generic Approach towards Maneuver Coordination for Automated Vehicles." In: *21st International Conference on Intelligent Transportation Systems (ITSC)*. 2018, pp. 3333–3339 (cit. on pp. 18, 20).

[Lin+10] S. Lin, B. De Schutter, Y. Xi, and H. Hellendoorn. "Model Predictive Control for urban traffic networks via MILP." In: (2010), pp. 2272–2277 (cit. on p. 24).

[LKK15] D. Lenz, T. Kessler, and A. Knoll. "Stochastic Model Predictive Controller with Chance Constraints for Comfortable and Safe Driving Behavior of Autonomous Vehicles." In: *IEEE Intelligent Vehicles Symposium (IV)*. 2015, pp. 292–297 (cit. on pp. 4, 34, 132).

[LKK16] D. Lenz, T. Kessler, and A. Knoll. "Tactical cooperative planning for autonomous highway driving using Monte-Carlo Tree Search." In: *2016 IEEE Intelligent Vehicles Symposium (IV)*. 2016, pp. 447–453 (cit. on p. 17).

[LL20] A. Liniger and J. Lygeros. "A Noncooperative Game Approach to Autonomous Racing." In: *IEEE Transactions on Control Systems Technology* 28.3 (2020). arXiv: 1712.03913 (cit. on pp. 17, 18).

[LR17] M. W. Levin and D. Rey. "Conflict-point formulation of intersection control for autonomous vehicles." In: *Transportation Research Part C: Emerging Technologies* 85.9 (2017), pp. 528–547 (cit. on pp. 22, 24).

[LSM21] S. Le Cleac'h, M. Schwager, and Z. Manchester. "ALGAMES: a fast augmented Lagrangian solver for constrained dynamic games." In: *Autonomous Robots* (2021) (cit. on p. 19).

[LVL14] S. Lefèvre, D. Vasquez, and C. Laugier. "A survey on motion prediction and risk assessment for intelligent vehicles." In: *ROBOMECH Journal* 1.1 (2014), p. 1 (cit. on p. 16).

[MA18] S. Manzinger and M. Althoff. "Tactical Decision Making for Cooperative Vehicles Using Reachable Sets." In: *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*. 2018, pp. 444–451 (cit. on pp. 18, 21).

[MCS15] N. Murgovski, G. R. de Campos, and J. Sjöberg. "Convex modeling of conflict resolution at traffic intersections." In: *2015 54th IEEE Conference on Decision and Control (CDC)*. 2015, pp. 4708–4713 (cit. on p. 18).

[Mer+20] J. C. Mertens, C. Knies, F. Diermeyer, S. Escherle, and S. Kraus. "The Need for Cooperative Automated Driving." In: *Electronics* 9.5 (2020) (cit. on pp. 1, 20).

[Min+16] P. Minnerup, D. Lenz, T. Kessler, and A. Knoll. "Debugging Autonomous Driving Systems Using Serialized Software Components." In: *IFAC-PapersOnLine* 49.15 (2016), pp. 44–49 (cit. on pp. 4, 130).

[Mon+09] M. Montemerlo, J. Becker, S. Bhat, H. Dahlkamp, D. Dolgov, S. Ettinger, D. Hähnel, T. Hilden, G. Hoffmann, B. Huhnke, D. Johnston, S. Klumpp, D. Langer, A. Levandowski, J. Levinson, J. Marcil, D. Orenstein, J. Paefgen, I. Penny, A. Petrovskaya, M. Pflueger, G. Stanek, D. Stavens, A. Vogt, and S. Thrun. "Junior: The Stanford Entry in the Urban Challenge." In: *The DARPA Urban Challenge: Autonomous Vehicles in City Traffic, George Air Force Base, Victorville, California, USA*. Ed. by M. Buehler, K. Iagnemma, and S. Singh. Vol. 56. Springer Tracts in Advanced Robotics. Springer, 2009, pp. 91–123 (cit. on pp. 15, 16, 25, 26).

[MPA18] C. Miller, C. Pek, and M. Althoff. "Efficient Mixed-Integer Programming for Longitudinal and Lateral Motion Planning of Autonomous Vehicles." In: (2018), pp. 1954–1961 (cit. on p. 23).

[MT21] T. Marcucci and R. Tedrake. "Warm Start of Mixed-Integer Programs for Model Predictive Control of Hybrid Systems." In: *IEEE Transactions on Automatic Control* 66.6 (2021), pp. 2433–2448 (cit. on p. 89).

[Nau+20] 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: *2020 IEEE International Conference on Robotics and Automation (ICRA)*. 2020, pp. 5481–5487 (cit. on p. 149).

[Nil+16] J. Nilsson, M. Brännström, J. Fredriksson, and E. Coelingh. "Longitudinal and Lateral Control for Automated Yielding Maneuvers." In: *IEEE Transactions on Intelligent Transportation Systems* 17.5 (2016), pp. 1404–1414 (cit. on pp. 22, 23).

[Nun+12] E. van Nunen, M. R. J. A. E. Kwakkernaat, J. Ploeg, and B. D. Netten. "Cooperative Competition for Future Mobility." In: *IEEE Transactions on Intelligent Transportation Systems* 13.3 (2012), pp. 1018–1025 (cit. on pp. 20, 25).

[NVI21] NVIDIA. *DriveWorks*. `https://developer.nvidia.com/drive/driveworks`. 2021 (accessed September 30, 2021) (cit. on p. 26).

[Ope21] OpenMp. *The OpenMP API specification for parallel programming*. `https://www.openmp.org/`. 2021 (accessed September 30, 2021) (cit. on p. 42).

[PA05] J. Peng and S. Akella. "Multiple Robots with Kinodynamic Constraints Along Specified Paths." In: *The International Journal of Robotics Research* 24.4 (2005), pp. 295–310 (cit. on p. 18).

[Pad+16] B. Paden, M. Čáp, S. Z. Yong, D. Yershov, and E. Frazzoli. "A Survey of Motion Planning and Control Techniques for Self-Driving Urban Vehicles." In: *IEEE Transactions on Intelligent Vehicles* 1.1 (2016), pp. 33–55 (cit. on pp. 14, 15).

[Pap+03] M. Papageorgiou, C. Diakaki, V. Dinopoulou, A. Kotsialos, and Y. Wang. "Review of road traffic control strategies." In: *Proceedings of the IEEE* 91.12 (2003), pp. 2043–2067 (cit. on p. 24).

[Pek+20] C. Pek, S. Manzinger, M. Koschi, and M. Althoff. "Using online verification to prevent autonomous vehicles from causing accidents." In: *Nature Machine Intelligence* 2.9 (2020), pp. 518–528 (cit. on p. 147).

[Pen+17] S. D. Pendleton, H. Andersen, X. Du, X. Shen, M. Meghjani, Y. H. Eng, D. Rus, and M. H. Ang. "Perception, planning, control, and coordination for autonomous vehicles." In: *Machines* 5.1 (2017), pp. 1–54 (cit. on p. 15).

[PGP17] B. Piot, M. Geist, and O. Pietquin. "Bridging the gap between imitation learning and inverse reinforcement learning." In: *IEEE Transactions on Neural Networks and Learning Systems* 28.8 (2017), pp. 1814–1826 (cit. on pp. 148, 149).

[Pog+18] F. Poggenhans, J. H. Pauls, J. Janosovits, S. Orf, M. Naumann, F. Kuhnt, and M. Mayr. "Lanelet2: A high-definition map framework for the future of automated driving." In: *2018 21st International Conference on Intelligent Transportation Systems (ITSC)* (2018), pp. 1672–1679 (cit. on p. 122).

[Qia+16] X. Qian, F. Altché, P. Bender, C. Stiller, and A. de La Fortelle. "Optimal trajectory planning for autonomous driving integrating logical constraints: An MIQP perspective." In: *19th IEEE International Conference on Intelligent Transportation Systems (ITSC)*. 2016, pp. 205–210 (cit. on pp. 22, 66, 69, 84).

[Raj12] R. Rajamani. *Vehicle Dynamics and Control*. Springer Science & Business Media, 2012 (cit. on p. 125).

[RD12] R. Reghelin and L. V. R. De Arruda. "A centralized traffic controller for intelligent vehicles in a segment of a multilane highway." In: *IEEE Intelligent Vehicles Symposium, Proceedings*. 2012, pp. 135–140 (cit. on p. 24).

[Rei+09] C. Reinholtz, D. Hong, A. Wicks, A. Bacha, C. Bauman, R. Faruque, M. Fleming, C. Terwelp, T. Alberi, D. Anderson, S. Cacciola, P. Currier, A. Dalton, J. Farmer, J. Hurdus, S. Kimmel, P. King, A. Taylor, D. Van Covern, and M. Webster. "Odin: Team VictorTango's entry in the DARPA Urban Challenge." In: *Springer Tracts in Advanced Robotics*. Vol. 56. 2009, pp. 125–162 (cit. on pp. 15, 16).

[Rei+21] R. Reiter, M. Kirchengast, D. Watzenig, and M. Diehl. "Mixed-integer optimization-based planning for autonomous racing with obstacles and rewards." In: *IFAC-PapersOnLine* 54.6 (2021). 7th IFAC Conference on Nonlinear Model Predictive Control NMPC 2021, pp. 99–106 (cit. on p. 23).

[Rei79] J. H. Reif. "Complexity of the mover's problem and generalizations." In: *Proceedings - Annual IEEE Symposium on Foundations of Computer Science, FOCS* (1979), pp. 421–427 (cit. on p. 14).

[RFS13] G. Rodrigues de Campos, P. Falcone, and J. Sjöberg. "Autonomous cooperative driving: A velocity-based negotiation approach for intersection crossing." In: *16th International IEEE Conference on Intelligent Transportation Systems (ITSC)*. 2013, pp. 1456–1461 (cit. on pp. 18, 21).

[RGL19] V. M. Raju, V. Gupta, and S. Lomate. "Performance of Open Autonomous Vehicle Platforms: Autoware and Apollo." In: *2019 IEEE 5th International Conference for Convergence in Technology, I2CT 2019* (2019), pp. 5–9 (cit. on p. 26).

[RM17] J. Rios-Torres and A. A. Malikopoulos. "A Survey on the Coordination of Connected and Automated Vehicles at Intersections and Merging at Highway On-Ramps." In: *IEEE Transactions on Intelligent Transportation Systems* 18.5 (2017), pp. 1066–1077 (cit. on p. 20).

[Rod+17] G. Rodrigues de Campos, P. Falcone, R. Hult, H. Wymeersch, and J. Sjöberg. "Traffic Coordination at Road Intersections: Autonomous Decision-Making Algorithms Using Model-Based Heuristics." In: *IEEE Intelligent Transportation Systems Magazine* 9.1 (2017), pp. 8–21 (cit. on p. 18).

[RS16] H. Rewald and O. Stursberg. "Cooperation of autonomous vehicles using a hierarchy of auction-based and model-predictive control." In: *IEEE Intelligent Vehicles Symposium, Proceedings*. 2016, pp. 1078–1084 (cit. on pp. 18, 20).

[SA20] L. Shimanuki and B. Axelrod. “Hardness of 3D Motion Planning Under Obstacle Uncertainty.” In: *Algorithmic Foundations of Robotics XIII*. 2020, pp. 852–867 (cit. on p. 14).

[SAR18] W. Schwarting, J. Alonso-Mora, and D. Rus. “Planning and Decision-Making for Autonomous Vehicles.” In: *Annual Review of Control, Robotics, and Autonomous Systems* 1.1 (2018), pp. 187–210 (cit. on pp. 1, 15, 19).

[Sch+01] T. Schouwenaars, B. De Moor, E. Feron, and J. How. “Mixed integer programming for multi-vehicle path planning.” In: *2001 European Control Conference (ECC)*. 2001, pp. 2603–2608 (cit. on p. 22).

[Sch+21] W. Schwarting, A. Pierson, S. Karaman, and D. Rus. “Stochastic Dynamic Games in Belief Space.” In: *IEEE Transactions on Robotics* 37.6 (2021), pp. 2157–2172 (cit. on pp. 17, 18).

[She+15] X. Shen, J. C. Zhuang, S. Pendleton, W. Liu, B. Qin, G. M. J. Fu, and M. H. Ang. “Multi-vehicle motion coordination using V2V communication.” In: *2015 IEEE Intelligent Vehicles Symposium (IV)*. 2015, pp. 1334–1341 (cit. on p. 18).

[SLL21] J. Siek, L.-Q. Lee, and A. Lumsdaine. *The Boost Graph Library (BGL)*. `https://www.boost.org/doc/libs/1_72_0/libs/graph/doc/index.html`. 2021 (accessed September 30, 2021) (cit. on p. 41).

[SP14] W. Schwarting and P. Pascheka. “Recursive Conflict Resolution for Cooperative Motion Planning in Dynamic Highway Traffic.” In: *International Conference on Intelligent Transportation Systems (ITSC)*. 2014, pp. 1039–1044 (cit. on pp. 18, 21).

[SS15] G. Sierksma and G. Sierksma. *Linear and Integer Optimization: Theory and Practice*. Chapman and Hall/CRC, 2015 (cit. on p. 12).

[SSS17] S. Shalev-Shwartz, S. Shammah, and A. Shashua. “On a Formal Model of Safe and Scalable Self-driving Cars.” In: *CoRR* abs/1708.06374 (2017). arXiv: `1708.06374` (cit. on p. 147).

[Tas+18] O. S. Tas, N. O. Salscheider, F. Poggenhans, S. Wirges, C. Bandera, M. R. Zofka, T. Strauss, J. M. Zollner, and C. Stiller. “Making Bertha Cooperate-Team AnnieWAY’s Entry to the 2016 Grand Cooperative Driving Challenge.” In: *IEEE Transactions on Intelligent Transportation Systems* 19.4 (2018), pp. 1262–1276 (cit. on pp. 25, 118).

[TD96] F. Thomanek and E. D. Dickmanns. “Autonomous road vehicle guidance in normal traffic.” In: *Recent Developments in Computer Vision. ACCV 1995. Lecture Notes in Computer Science*. Vol. 1035. 1996 (cit. on pp. 24, 25).

[The21a] The MathWorks, Inc. *Automated Driving Toolbox*. `https://de.mathworks.com/products/automated-driving.html`. 2021(accessed September 30, 2021) (cit. on p. 27).

[The21b] The MathWorks, Inc. *MATLAB*. `https://de.mathworks.com/products/matlab.htmlo`. 2021 (accessed September 30, 2021) (cit. on p. 89).

[Tog+21] B. Toghi, R. Valiente, D. Sadigh, R. Pedarsani, and Y. P. Fallah. “Altruistic Maneuver Planning for Cooperative Autonomous Vehicles Using Multi-agent Advantage Actor-Critic.” In: *CoRR* abs/2107.05664 (2021). arXiv: `2107.05664` (cit. on p. 21).

[Ulb+15] S. Ulbrich, S. Grossjohann, C. Appelt, K. Homeier, J. Rieken, and M. Maurer. “Structuring Cooperative Behavior Planning Implementations for Automated Driving.” In: *2015 IEEE 18th International Conference on Intelligent Transportation Systems*. 2015, pp. 2159–2165 (cit. on pp. 2, 20).

[UM13] S. Ulbrich and M. Maurer. "Probabilistic online POMDP decision making for lane changes in fully automated driving." In: *IEEE Conference on Intelligent Transportation Systems, Proceedings, ITSC* Itsc (2013), pp. 2063–2070 (cit. on p. 16).

[Urm+08] C. Urmson, J. Anhalt, D. Bagnell, C. Baker, R. Bittner, M. N. Clark, J. Dolan, D. Duggins, T. Galatali, C. Geyer, M. Gittleman, S. Harbaugh, M. Hebert, T. M. Howard, S. Kolski, A. Kelly, M. Likhachev, M. Mcnaughton, N. Miller, K. Peterson, B. Pilnick, R. Rajkumar, P. Rybski, B. Salesky, Y. W. Seo, S. Singh, J. Snider, A. Stentz, W. R. Whittaker, Z. Wolkowicki, J. Ziglar, H. Bae, T. Brown, D. Demitrish, B. Litkouhi, J. Nickolaou, V. Sadekar, W. Zhang, J. Struble, M. Taylor, M. Darms, and D. Ferguson. "Autonomous driving in urban environments: Boss and the urban challenge." In: *Journal of Field Robotics* 25.8 (2008), pp. 425–466 (cit. on pp. 15, 16, 25).

[Val+17] C. Vallon, Z. Ercan, A. Carvalho, and F. Borrelli. "A machine learning approach for personalized autonomous lane change initiation and control." In: (2017), pp. 1590–1595 (cit. on p. 19).

[Via+21] I. B. Viana, H. Kanchwala, K. Ahiska, and N. Aouf. "A Comparison of Trajectory Planning and Control Frameworks for Cooperative Autonomous Driving." In: *Journal of Dynamic Systems, Measurement, and Control* 143.7 (Feb. 2021) (cit. on p. 21).

[VJ16] K. G. Vamvoudakis and S. Jagannathan. *Control of Complex Systems Theory and Applications*. Butterworth-Heinemann, 2016 (cit. on p. 14).

[Wan+18] Z. Wang, Y. Zheng, S. E. Li, K. You, and K. Li. "Parallel Optimal Control for Cooperative Automation of Large-scale Connected Vehicles via ADMM." In: *21st IEEE International Conference on Intelligent Transportation Systems (ITSC)*. IEEE, 2018, pp. 1633–1639 (cit. on p. 20).

[Wan+19] M. Wang, Z. Wang, J. Talbot, J. Christian Gerdes, and M. Schwager. "Game Theoretic Planning for Self-Driving Cars in Competitive Scenarios." In: *Robotics: Science and Systems 2019*. 2019 (cit. on pp. 18, 19).

[Wan+20] Y. Wang, S. Jiang, W. Lin, Y. Cao, L. Lin, J. Hu, J. Miao, and Q. Luo. "A Learning-Based Tune-Free Control Framework for Large Scale Autonomous Driving System Deployment." In: *CoRR* abs/2011.04250 (2020). arXiv: `2011.04250` (cit. on pp. 26, 30).

[Wan+21] T. Wang, A. Amini, W. Schwarting, I. Gilitschenski, S. Karaman, and D. Rus. "Learning Interactive Driving Policies via Data-driven Simulation." In: *CoRR* abs/2111.12137 (2021). arXiv: `2111.12137` (cit. on pp. 2, 20).

[WDL13] J. Wei, J. M. Dolan, and B. Litkouhi. "Autonomous vehicle social behavior for highway entrance ramp management." In: (2013), pp. 201–207 (cit. on p. 19).

[Wil90] G. Wilfong. "Graphs with variable edge costs: a model for scheduling a vehicle subject to speed and timing restraints." In: *EEE International Workshop on Intelligent Robots and Systems, Towards a New Frontier of Applications*. Vol. 1. 1990, pp. 73–79 (cit. on p. 148).

[Wol+17] P. Wolf, C. Hubschneider, M. Weber, A. Bauer, J. Härtl, F. Dürr, and J. M. Zöllner. "Learning how to drive in a real world simulation with deep Q-Networks." In: (2017), pp. 244–250 (cit. on p. 20).

[WRA20] X. Wang, A.-K. Rettinger, and M. Althoff. "Coupling Apollo with the CommonRoad Motion Planning Framework." In: *FISITA World Congress* (2020) (cit. on p. 26).

[WZT10] M. Werling, J. Ziegler, and S. Thrun. "Optimal Trajectory Generation for Dynamic Street Scenarios in a Frenet Frame." In: *IEEE International Conference on Robotics and Automation*. 2010, pp. 987–993 (cit. on pp. 16, 29, 123, 131).

[Xu+19] J. Xu, Q. Luo, K. Xu, X. Xiao, S. Yu, J. Hu, J. Miao, and J. Wang. "An Automated Learning-Based Procedure for Large-scale Vehicle Dynamics Modeling on Baidu Apollo Platform." In: *IEEE International Conference on Intelligent Robots and Systems* (2019), pp. 5049–5056 (cit. on p. 26).

[Xu+20] K. Xu, X. Xiao, J. Miao, and Q. Luo. "Data Driven Prediction Architecture for Autonomous Driving and its Application on Apollo Platform." In: (2020), pp. 175–181 (cit. on p. 26).

[Yu16] J. Yu. "Intractability of optimal multirobot path planning on planar graphs." In: *IEEE Robotics and Automation Letters* 1.1 (2016), pp. 33–40. arXiv: 1504.02072 (cit. on p. 14).

[Yur+20] E. Yurtsever, J. Lambert, A. Carballo, and K. Takeda. "A Survey of Autonomous Driving: Common Practices and Emerging Technologies." In: *IEEE Access* 8 (2020), pp. 58443–58469. arXiv: 1906.05113 (cit. on p. 15).

[Zha+19] Y. Zhang, H. Sun, J. Zhou, J. Hu, and J. Miao. "Optimal Trajectory Generation for Autonomous Vehicles Under Centripetal Acceleration Constraints for In-lane Driving Scenarios." In: *2019 IEEE Intelligent Transportation Systems Conference (ITSC)* (2019), pp. 3619–3626 (cit. on p. 28).

[Zha+20] Y. Zhang, H. Sun, J. Zhou, J. Pan, J. Hu, and J. Miao. "Optimal Vehicle Path Planning Using Quadratic Optimization for Baidu Apollo Open Platform." In: *2020 IEEE Intelligent Vehicles Symposium (IV)*. 2020, pp. 978–984 (cit. on p. 28).

[Zho+20] J. Zhou, R. He, Y. Wang, S. Jiang, Z. Zhu, J. Hu, J. Miao, and Q. Luo. "DL-IAPS and PJSO: A Path/Speed Decoupled Trajectory Optimization and its Application in Autonomous Driving." In: *CoRR* abs/2009.11135 (2020). arXiv: 2009.11135 (cit. on p. 29).

[Zie+14a] J. Ziegler, P. Bender, T. Dang, and C. Stiller. "Trajectory planning for Bertha - A local, continuous method." In: *2014 IEEE Intelligent Vehicles Symposium Proceedings* (2014), pp. 450–457 (cit. on pp. 2, 17, 23, 24, 68, 69, 81).

[Zie+14b] J. Ziegler, P. Bender, M. Schreiber, H. Lategahn, T. Strauss, C. Stiller, T. Dang, U. Franke, N. Appenrodt, C. G. Keller, E. Kaus, R. G. Herrtwich, C. Rabe, D. Pfeiffer, F. Lindner, F. Stein, F. Erbs, M. Enzweiler, C. Knoppel, J. Hipp, M. Haueis, M. Trepte, C. Brenk, A. Tamke, M. Ghanaat, M. Braun, A. Joos, H. Fritz, H. Mock, M. Hein, and E. Zeeb. "Making bertha drive-an autonomous journey on a historic route." In: *IEEE Intelligent Transportation Systems Magazine* 6.2 (2014), pp. 8–20 (cit. on pp. 17, 24, 25).