Master Thesis

Dahlem Center for Machine Learning and Robotics
Department of Mathematics and Computer Science

Graph-Based Speed Planning for Autonomous Driving

Till-Julius Krüger
till.krueger@fu-berlin.de

Reviewer:

Prof. Dr. Daniel Göhring

Prof. Dr. Dr. (h.c.) habil. Raúl Rojas

Advisor:

Fritz Ulbrich

9th August 2019

Eidesstattliche Erklärung

Ich versichere hiermit an Eides statt, dass diese Arbeit von niemand anderem als meiner Person verfasst worden ist. Alle verwendeten Hilfsmittel wie Berichte, Bücher, Internetseiten oder ähnliches sind im Literaturverzeichnis angegeben, Zitate aus fremden Arbeiten sind als solche kenntlich gemacht. Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungskommission vorgelegt und auch nicht veröffentlicht.

Berlin, 09.August 2019

Till-Julius Krüger

Abstract

Motion planning in autonomous driving defines the task of planning the desired movement of a vehicle through a dynamically moving environment. A plan is stored as trajectory, saving spatial and temporal information about the future vehicle movement. Path-Speed decomposition is a planning method for finding such a trajectory. A path is planned in a first step, followed by an according speed profile.

This thesis aims to implement and evaluate a planner for finding a rough speed profile in a discretized search space. A graph is created and a single source shortest paths algorithm is used to find the optimal speed profile within the limited search space, evaluated by cost functions representing the requirements of speed planning. The rough speed profile can serve as initial solution for numerical optimization, which is not part of this thesis.

The implemented approach is evaluated in simulation of various urban traffic scenarios, showing promising collision free and low-jerk trajectories. It is able to find a speed profile in real-time. Therefore, the planner seems useful for practical application in an autonomous driving vehicle.

Acknowledgment

I want to thank Dr. Göhring for supervising this thesis and Fritz Ulbrich for enlightening discussions and feedback. I would also like to thank my colleagues at TomTom, especially Stefan Kaiser and Christian Klauer for their helpful advice and support.

Special thanks go to my friends and family, in particular Michaela, Marina, Horst and Jan, for their great love and backup in complicated times.

Nomenclature

$\mathbb{S}$	Station configuration space
$\mathbb{X}$	Vehicle configuration space
P	Path of the vehicle
$\mathbb{T}$	Time configuration space
L	Solution space of speed planning
v	Vehicle speed
a	Vehicle acceleration
j	Vehicle jerk
J	Cost function
C	Constraint function space
rsp	Reference speed profile
A	Set of agents
τ	Time sampling parameter
σ	Station sampling parameter
$\mathbb{S}_\sigma$	Sampled station configuration space
$\mathbb{T}_\tau$	Sampled time configuration space
$h_{station}$	Station horizon
h_{time}	Time horizon
$n_{station}$	Station horizon index
n_{time}	Time horizon index
$V_{\tau\sigma}$	Nodes of the station-time graph
$E_{\tau\sigma}$	Edges of the station-time graph
$J_{E_{\tau\sigma}}$	Edge cost function of station-time graph
$M_{\tau\sigma}$	Matrix representation of station-time graph
β	Path in station-time graph
π	Predecessor of node in station-time graph
SSSP	Single source shortest paths
DAG	Directed acyclic graph
$\gamma_\square$	Weight of cost function $\square$
J_V	Reference speed cost function
J_A	Acceleration cost function
J_J	Jerk cost function
J_A	Obstacle interaction cost function
v_{Ref}	Reference speed
α_1, α_2	Scaling parameters of J_V
$[a_{soft\min}, a_{soft\max}]$	Comfort acceleration range
ψ	Acceleration-speed relation threshold
d	Longitudinal safety distance to an obstacle

Contents

1 Introduction

An autonomous vehicle has to handle complex traffic scenarios. It has to consider dynamically acting traffic participants as well as traffic regulations. A trajectory, consisting of time and space information, defines the desired movement of a vehicle through a moving environment. Motion planning is responsible for finding such a trajectory that is feasible for the vehicle, comfortable for the passengers and avoids collisions with other traffic participants.

One approach for solving the motion planning problem is to split the problem into two elements. The path, defining *where* the vehicle will drive, is planned as a first step. Path planning takes road geometry and stationary obstacles into account. The second step is to determine, *when* the vehicle will pass a point on the path. This is defined by a time stamp connected with each point on the path and hence the speed that the vehicle needs to drive to reach the point in time. The time component is responsible for avoiding collisions with dynamic obstacles.

This thesis focuses on temporal speed planning for a given path. The search space is discretized and a graph search algorithm, which uses dynamic programming, is applied.

1.1 Motivation

> **Vision Zero** is a philosophy *that eventually no one will be killed or seriously injured within the road transport system* [1].

Having transportation is an essential need of a society. However, it comes with the burden of traffic accidents and pollution of the environment [2]. Worldwide, approximately 1.35 million people die each year because of traffic accidents. Road traffic injuries are the leading cause for deaths of humans aged 5-29 years [3]. In 2018, around 25100 fatalities due to road accidents happend in the EU. Germany has 39 road deaths per million inhabitants [4]. The pollution of the environment due to emissions of vehicles is immense. It becomes clear that it is a long way to reach **Vision Zero**.

The WHO names intelligent vehicles as a step to contribute to traffic injury prevention [6]. According to DVR [7], advanced driver assistance systems (**ADAS**) minimize the risk of road accidents. Since 94% of crashes in the U.S. are caused by the

driver [8], autonomous driving can contribute to the motives of **Vision Zero**, by excluding the factor of human failure from traffic. Homogenization of traffic by connected and autonomous vehicles can reduce the energy-consumption and therefore the amount of emissions [5, p. 55].

Motion planning is an important task of autonomous driving. Interacting with other vehicles is still an open problem for autonomous vehicles. Speed planning is necessary to avoid collisions with dynamic obstacles. Therefore, this thesis contributes a small step in the direction of autonomous driving and reaching **Vision Zero**.

The selected approach claims to find an optimal solution within the discretized search space in real-time, minimizing objective functions and comply with given constraints. Thereby, vehicle and road limitations are considered, as well as passenger safety and comfort. The domain of the approach is generic, hence is not limited to a certain environment (e.g. highway). The resulting trajectory serves as an initial solution for further optimization, which is not part of this thesis.

1.2 Contributions

The main contribution of this thesis is the implementation and evaluation of a graph-based speed planning approach in terms of functionality, performance and scalability. It provides a detailed description of transforming the speed planning problem to a shortest path problem in a graph. Cost functions for the discrete optimization were developed. The implemented algorithm was tested in various urban scenarios in simulation.

1.3 Outline

After giving a brief introduction, state of the art procedures for motion planning and especially speed planning are discussed in chapter 2. The next chapter will define the requirements and fundamentals for speed planning including a formal problem definition. Chapter 4 describes the implementation of optimizing a discrete speed planning problem. Therefore, the transformation of the speed planning problem to a shortest paths problem is described. A shortest path algorithm will be explained, followed by its application to discrete speed planning. In chapter 5, the algorithm will be evaluated using a set of common urban scenarios. The last chapter will provide conclusions and an outlook for future work.

2 Related Work

Motion planning is a complex and broad topic. Section 2.1 gives a short introduction to direct and decoupled motion planning by comparing existing solutions. The main focus is on the decoupled planning, since this thesis inspects a sub-problem of motion planning: speed planning.

After comparing motion planning designs, algorithms for speed planning are discussed. As Fan et al.[9] points out, the speed planning problem is a non-convex problem. An approach for attempting to find an optimal solution is to split the speed planning problem into two parts:

1. **Rough speed profile:** A discretizing approach is responsible for providing a *candidate solution*. The solution is calculated based on a discretized search space and therefore is most likely not optimal in the continuous search space. However, the aim is to find an optimal solution in the limited search space to serve as initial solution for numerical optimization.
2. **Optimized speed profile:** Based on the rough speed profile, a convex configuration space can be determined. This allows numerical optimization, resulting in a smoother and better trajectory in a non-discretized search space.

Combining both steps overcomes the problems of numerical optimization with potential local minima, resulting in a more optimal trajectory. Algorithms aiming to find a rough speed profile are discussed in section 2.2.

2.1 Motion Planning Architecture

The following quote visualizes the process of motion planning in an intuitive manner:

> Suppose that you arrive early at a cocktail party and the hall is sparsely filled. You desire a drink and glance around the hall to locate the bar. Except for avoiding an occasional piece of furniture, you walk straight toward it. However, as another person approaches your path, you slow down to avoid a collision and, after she passes, you speed up again [10].

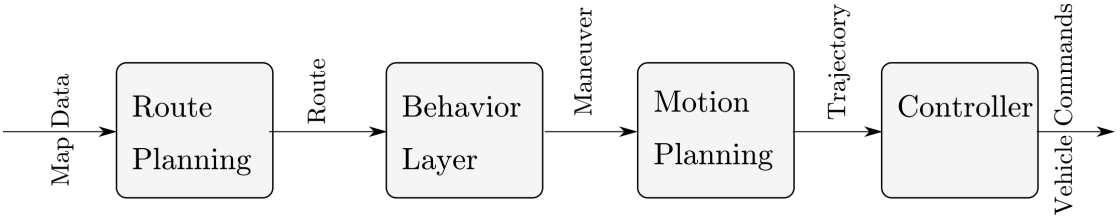


Figure 1: An example of a module architecture of an autonomous vehicle [11].

Motion planning for autonomous vehicles can be split in different modules [11]. The process through the modules is visualized in Fig. 1.

The starting module is the route planning which selects a preferred route through a network of roads, organized in a map. The route defines the global plan of the autonomous vehicle as a sequence of roads. Since the route planning doesn't take the local environment into account, local planning is proceeded in the following modules. The behavior layer is responsible for decision-making of an abstract maneuver, based on the given situation (for example, keep-lane, lane-change). To execute the abstract maneuver, a motion planning module is needed that defines a reference trajectory. The reference trajectory is then translated into vehicle commands by a controller module.

Direct Planning

Direct planning attempts to find an optimal trajectory in a single step. The authors of [12] use the A*-algorithm to find a trajectory in a three dimensional state graph. The graph adapts to the velocity of the vehicle by adjusting the spatial distance of the nodes, while the time differences are fix. The proposed algorithm shows online capability. On the downside, the algorithm has a very limited search space, resulting in a most probably non-optimal trajectory.

Another common choice is a spatiotemporal lattice graph that contains reachable states based on fixed motion primitives. Ziegler and Stiller [13] propose a real-time planner based on discrete spatiotemporal lattice graphs. The concept is extend by Matthew McNaughton et al. [14] by adding optional GPU support.

Direct planning usually suffers from the curse of dimensionality. Since a path and the corresponding speed profile is generated in one step, time and space have to be handled at once, leading to a multidimensional search space.

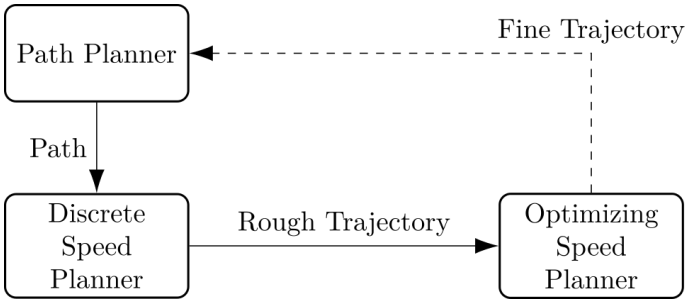


Figure 2: Simplified illustration of the iterative decoupled planning approach of [9].

Decoupled Planning

Decoupled planning splits the motion planning into parts and solves them consecutively. Because the parts are individually simpler to solve, decoupled planning takes advantage of the complexity reduction.

Gu and Dolan [15] proposed a planner that uses dynamic programming to find desirable maneuvers represented as a sequence of safe vehicle poses. Thereby, the search space is decreased. Based on the solution of the exploring step, paths and velocity profiles for the paths are sampled. The sampled trajectories are evaluated by a cost function.

Another idea incorporating decoupled planning is *path-velocity decomposition* [10]. Here, the path is calculated in the first place, respecting only static obstacles. Afterwards, the speed profile is calculated for the given path. Examples of decoupled approaches are discussed in section 2.2, setting the focus on the speed planning algorithms. A possible problem with decoupled planning is that the planned trajectory is not necessarily optimal, since path and speed planning depend on each other are not easy to separate [16].

To overcome this issue, the authors of [9] propose an iterative space-time decoupled approach. Within a planning cycle, path and speed planning is repeated to improve the latest solution. The procedure is visualized in Fig. 2.

2.2 Speed Planning Algorithms

Finding examples for direct motion planners is an easy task. On the other hand, decoupled planners with a two-step speed planning phase are rare. The following is an overview of speed planners used to find a rough speed profile.

A simple speed planner is proposed by Urmson et al. [17]. Based on the situation, a parameterized speed profile is selected from a pool of four different linear speed profiles. This limits the search space and is therefore rather not optimal.

The version of decoupled planning proposed by Gu and Dolan [15] uses speed as part of the dynamic programming cost function. An issue here is that they don't provide information about the used algorithm. The approach is extended by Gu, Dolan and Lee [18]: A multi phase framework decomposes the planning problems into smaller problems in each phase. The idea of using a reference speed profile for speed planning is proposed. A reference speed profile defines a reference speed for each point on the path, instead of having a general speed limit. It can incorporate static environment information as traffic lights, speed limits, road curvature and traffic signs. The information is translated into speed limitations for certain areas of the path. The main advantage of the use of a reference speed profile is another decoupling effect: the planner doesn't need to be aware of the given situation, since the reference speed profile abstracts it. In many motion planners a distinction is still made depending on the current environment situation (e.g. [19] focuses on highway driving). Therefore, a cluster of situation-based planners needs to be managed as described in [20].

The configuration space of speed planning can be pruned by using a visibility graph [21, 22]. Considering only speed profiles that touch start or end point of edges in the graph, the algorithm delivers in real-time. The blocked space by obstacles over time is approximated by a convex hull, allowing various prediction models for an obstacle, e.g. non-constant acceleration prediction. On the downside, the produced speed profile is very rough. Due to the pruning, only a limited number of candidates is inspected. The jerk of the speed limit is ignored, leading to an uncomfortable speed profile.

Another graph-based approach for speed planning can be found in Lim et al. [23]. The authors use decoupled planning to generate a rough trajectory in the first place. The rough trajectory is found by exploring a discretized search-space. In a second step, numerical optimization is used to smooth the trajectory and improve traveling comfort. The generation of the speed profile is supported by a reference speed profile. The authors claim to use a hybrid-A* algorithm, but they don't define the heuristic for estimating a solution, which is the key element of the algorithm. A defined space-horizon is used as termination criterion, which might lead to a huge variance in the algorithm run-time.

Fan et al. [9] decided to use a graph-based algorithm as well. In advance to the other approaches, they take all possible solution within the discretized search-space into ac-

count. The optimal solution depends on reference speed, obstacle avoidance and passenger comfort. The authors claim to use *dynamic programming* to find the optimal solution. Unfortunately, they don't give a detailed insight in the used algorithm.

Using a reference speed profile opens great opportunities for a situation-independent planning algorithm. Abstracting the static environment allows a single planner for multiple situations. The intention of the rough speed profile is to serve as initial solution for optimization. For this reason, it is not sufficient to find a good solution. The algorithm should find an optimal solution in the limited search-space, allowing the numerical optimization to achieve the best possible results. Therefore, the algorithm must respect traffic regulations, avoid collisions and be comfortable for passengers. The graph shouldn't be pre-pruned to provide an optimal solution in the discretized space. Hence, this thesis will follow the idea of an graph-based algorithm that evaluates an entire graph of [9].

3 Fundamentals for Speed Planning

After having compared approaches of related work in chapter 2, this chapter will present the fundamentals needed for speed planning. Section 3.1 describes the requirements to speed planning to achieve safe and comfortable driving. A formal problem definition is given in section 3.2, followed by an explanation of the input components of the speed planning algorithm in section 3.3 and 3.4.

3.1 Context of Safe and Comfortable Driving

Speed planning is part of trajectory planning. Thus, speed planning inherits the requirements from trajectory planning. Common requirements are briefly described below.

The most important requirement is the **feasibility** of a trajectory. The autonomous vehicle must be able to execute the planned trajectory and thereby the planned speed. In terms of speed planning, feasibility translates to respecting vehicle dynamics. There is a range of speed and acceleration that is achievable by the autonomous vehicle, depending on the engine.

The next constraint is the **passenger safety** as well for passengers of the autonomous vehicles as for other traffic participants. Collisions with other traffic participants and static objects must be avoided at all costs. Traffic regulations need to be respected as well.

Focusing on passengers, the next point is the **comfort** of the passengers. Humans feel discomfort when being exposed to jerk (e.g. roller coasters). To prevent high values of jerk, a sudden change of acceleration needs to be obviated.

A constraint that is directed to the algorithm, is to achieve real-time capability. Planning needs to take place online to react to the dynamic environment. Therefore section 5 measures the performance of the used planning algorithm.

3.2 Problem Definition

The following section will define the problem of speed planning. A path consists of positions combined with orientation of the vehicle which are called configurations. Hence, the path exists in space in a range between a start configuration and an end configuration.

Speed planning connects the space component of a path with time. Combining space and time creates a trajectory with positions, orientations and time stamps. As a result, a speed profile is generated which maps time stamps to space for a given path. Thereby, the speed profile decides *when* the autonomous vehicle should reach a point on the path.

The path can be represented by the arc length between these configurations. Therefore it can be parameterized in the range of the start and end configuration. This parameter is called station.

Considering that, the station domain of a path is modeled as:

$$\mathbb{S} := \{s \in \mathbb{R}_+ | s \in [0, s_{\max}]\}, \tag{1}$$

where $s_{\max}$ maps to the end configuration and 0 to the start configuration of a path. The configuration space of all configurations can be regarded as:

$$\mathbb{X} := \left\{ x \in \begin{bmatrix} \mathbb{R}^2 \\ \theta \end{bmatrix} \right\}, \tag{2}$$

where $\mathbb{R}^2$ is a two dimensional coordinate and θ an orientation angle.

Let $\mathbb{X}_{blocked}$ denote the configurations that are blocked by static environment elements and $\mathbb{X}_{free}$ the free to pass configurations by $\mathbb{X}_{free} := \mathbb{X} \backslash \mathbb{X}_{blocked}$. Static environment elements denotes obstacles that are not expected to move (e.g. trees, houses) and obstacles that have the potential to move, but are not expected to move (e.g. parked cars).

Since a path is represented by configurations $x \in \mathbb{X}$ in respect to station $s \in \mathbb{S}$, there must exist a mapping from station to configuration:

$$P : \mathbb{S} \to \mathbb{X}_{free} \tag{3}$$

Therefore a path $p \in P$ is a continuous mapping from station to free configuration and therefore assumed to be free of static obstacles.

After covering the path, speed planning can be defined. Speed planning produces a speed profile that maps from time to station of a given path. This mapping can be expressed as a function using the following definition of time:

$$\mathbb{T} := \{t \in \mathbb{R}_+ | t \in [0, t_{\max}]\}, \tag{4}$$

where $t_{\max}$ is the maximum time horizon of the speed profile. Since planning with time is only reasonable for the future, assuming the lower bound by zero is sensible.

Given time and space, speed planning connects both in a mapping called speed profile [24, 25]. The speed profile can be expressed as a function that decides when the vehicle will be at which point of the path. On this basis L is the set of all speed profiles l:

$$L := \{l | l : \mathbb{T} \to \mathbb{S}\} \tag{5}$$

This thesis excludes backwards driving maneuvers. The reason for that is that backwards driving occurs in very small set of scenarios that need special care. One example is an emergency situation where a lane has to be cleared for an ambulance vehicle. Handling such scenario requires a specialized behavior that is not covered here.

Because of the giving limitation, the speed profile is required to be monotonically increasing, meaning that the speed planning will produce a profile without backwards driving:

$$\forall l \in L \wedge t, t' \in \mathbb{T} : t < t' \Rightarrow l(t) \leq l(t') \tag{6}$$

A full stop is an interval in the speed profile, where the vehicle is stopped in the future. In case of a full stop is the speed profile restricted to continue the full stop:

$$\exists t \in \mathbb{T}, \forall t' \in \mathbb{T} : t \neq 0 \wedge l(t) = 0 \wedge t' > t \Rightarrow l(t') = 0 \tag{7}$$

Therefore, the full stop needs to be executed by the vehicle, before a speed profile can be generated that continues after the full stop.

Δ is used to symbol a difference of two values. Equation 8 shows this content for an example symbol $\square$.

$$\Delta\square = \square' - \square \tag{8}$$

Given the speed profile, several basic vehicle dynamics can be approximated by the finite differences method:

$$v(\Delta s, \Delta t) = \frac{\Delta s}{\Delta t} \tag{9}$$

$$a(\Delta v, \Delta t) = \frac{\Delta v}{\Delta t} \tag{10}$$

$$j(\Delta a, \Delta t) = \frac{\Delta a}{\Delta t}, \tag{11}$$

where v is the average velocity / speed[1] in Δs, a the average acceleration in ΔV and j is the average jerk in Δa. In the following, *speed* will be used because it is the more intuitive and correct term. However, to prevent confusion with station, it is denoted as v.

3.2.1 Optimal Speed Planning

Optimal speed planning attempts to find not only a solution, but an optimal solution. A speed profile is evaluated by a cost function J:

$$J : L \to \mathbb{R}_+ \tag{12}$$

A solution l is optimal, if there is no solution with lower costs. Therefore multiple optimal solutions may exist:

$$L_{optimal} := \{l_{optimal} \in L | \forall l' \in L : J(l_{optimal}) \leq J(l')\} \tag{13}$$

As described in section 3.1, the speed profile has to fulfill a set of requirements. These requirements can be formulated as a set of functions that assesses whether a solution l satisfies a given constraint:

$$C := \{c | c : L \to \{0, 1\}\}, \tag{14}$$

where $c \to 1$ shows that the given constraint is fulfilled and $c \to 0$ a fail of the constraint.

Using the cost function and the requirements, speed planning can be formulated as an optimization problem:

$$\underset{l \in L}{\operatorname{argmin}} J(l) \tag{15}$$

$$\text{s.t. } \forall c \in C : c(l) = 1 \tag{16}$$

So the optimal speed profile is defined by the cost function and also meets the given constraints.

[1] Average speed is defined as traveled distance, divided by a time interval. Hence, speed expresses how fast an object is moving. In contrast, average velocity depicts speed in a certain direction and is therefore a vector. But since velocity is calculated based on station, it is one-dimensional and positive by definition. Thus, it is equivalent to speed in this context.

3.3 Reference Speed Profile

Knowledge and observation about the environment of the vehicle can be expressed as constraints to speed. The environment consists of two types: static and dynamic elements.

Static environments like road boundaries don't depend on time and can therefore be saved in a file. Relying on existing information about the static environment might be only sufficient in a closed track, but not in real world traffic. Adding observations from the dynamic environment enables the autonomous vehicle to react to events like traffic lights. Both data can be expressed as reference speed for the vehicle.

A reference speed profile is a mapping from station to a certain reference speed [26]. Formally speaking, it can be defined as a function $rsp : S \to \mathbb{R}$. The reference speed should be used as a suggestion for an upper limit of the speed. There might be situations where exceeding the given reference speed is reasonable. One can interpret the reference speed profile as an approach for modeling a driving scene, neglecting dynamic obstacles. Dynamic obstacles depend on time and can therefore not be modeled in a reference speed profile, since the speed at some station would depend on the speed at previous stations.

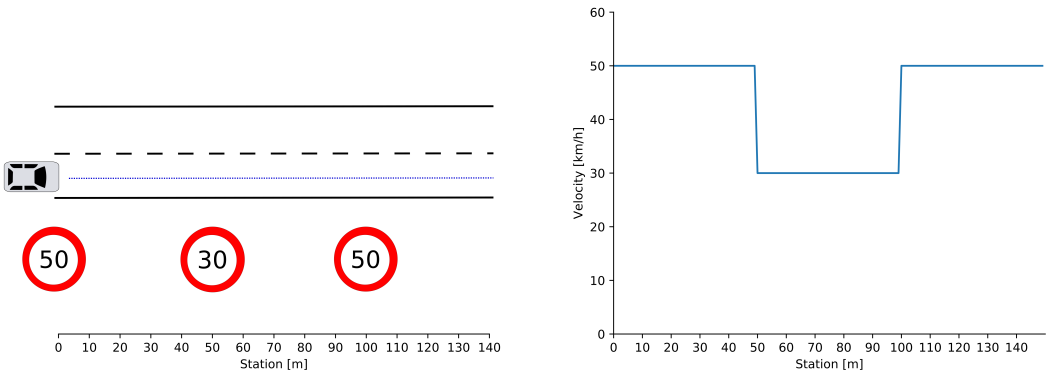
3.3.1 Constructing a Reference Speed Profile

A reference speed profile can be developed by a set of specialized agents. Each agent inspects a certain aspect of the scene and transforms it into a reference speed profile. Hence, an agent a of an agent set A can be described as a function from an input vector to a reference speed profile:

$$A := \{a | a : \mathbb{R}^n \to rsp\}, \tag{17}$$

where $\mathbb{R}^n$ describes an n-dimensional input vector that depends on the used agent.

A simple way of combining all the agent's profiles to a combined reference speed profile can then be attained by using the minimal speed of the profiles. This is a rather simple approach which assumes that each situation is unambiguous. In order to handle ambiguous situations like a traffic light with an additional traffic sign, a priority decider is needed. Defining such a decider is a rather complex task, since the prioritization depends on the situation.



(a) (b)

Figure 3: A simple scenario with the respective reference speed profile. On the left is the scene visualized: a lane-following car with different speed limits along the path. The path is represented as blue dots. The right hand image shows the related reference speed profile of the path based on the speed limit agent.

The following paragraphs describe a set of agents that are capable of handling a limited set of driving scenarios:

- Respect vehicle and road speed limitations
- Adjust speed to centripetal acceleration
- Respect traffic lights

To accomplish more complex driving scenarios, the set of agents needs to be extended .

Vehicle Limitations Agent A vehicle is bounded to its physical limitations. Depending on the engine, a vehicle has a valid speed range $[v_{\min}, v_{\max}]$ that it can achieve. The vehicle dynamics agent ensures that the reference speed profile doesn't exceed this limit. Therefore, it sets the the reference speed profile to $v_{\max}$. Note that this agent is not responsible for the valid acceleration range $[a_{\min}, a_{\max}]$.

Speed Limit Agent This agent is capable for limitation based on speed limits. The speed limit agent sets the reference speed profile to the given speed limit. An example of a traffic scenario with the according reference speed profile is given in Fig. 3. Speed limits may be detected during driving from the perception module or extracted from known

map data. A combination of both is favored, since static speed limits (e.g. inner city speed limit of $50km/h$) are not explicitly stated and can't be detected by perception. On the other hand, the perception based speed limit detection is essential to interact with temporary limitations (e.g. construction work).

Centripetal Acceleration Agent Due to passenger comfort and safety, it is appropriate to limit the centripetal acceleration which applies when driving curves. Without this limitation, the vehicle might start drifting in curves.

The curvature describes the rate of direction change of a curve. One can interpret a path $p \in \mathbb{P}$ as a sequence of small curves. These curves are defined by the position and orientation of two consecutive points on p.

So to calculate the curvature for a certain station value $s \in \mathbb{S}$ on p, the curvature for a curve defined by two points $p(s)$ and $p(s + \Delta s)$ has to be calculated. $\Delta s \in \mathbb{S}$ is the arc length and should be a small value to create a good approximation of the path.

To represent the curve, we can fit a circle that respects the orientation at $p(s)$ and $p(s+\Delta s)$. The circle has a radius R, which is called radius of curvature. The radius can be calculated by the difference between the orientations θ_s and $\theta_{\Delta s}$ at $p(s)$ and $p(s+\Delta s)$ divided by the arc length.

$$\kappa(s) = \frac{\theta_{\Delta s} - \theta_s}{\Delta s} \tag{18}$$

Using the curvature we can calculate the centripetal acceleration a_{cen} of the vehicle at the given station s based on the current speed v as defined in [26]:

$$a_{cen} = v^2 \cdot |\kappa(s)| \tag{19}$$

Since we want to limit the centripetal acceleration, we can reformulate this equation to get a maximal speed based on the curvature:

$$v = \sqrt{\frac{a_{cen}}{|\kappa(s)|}} \tag{20}$$

In case that there is no curvature, the speed isn't influenced and is not calculated. Figure 4 shows examples, how the maximum speed is limited for a given curvature and different centripetal acceleration values. Summarized, the centripetal acceleration agent

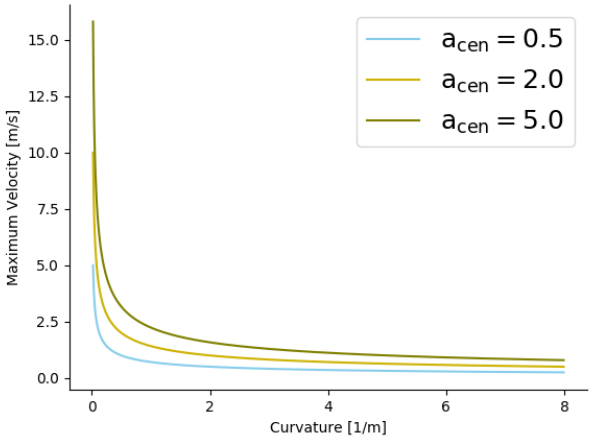


Figure 4: Certain maximal speed values that can be achieved with the given centripetal acceleration a_{cen} and curvature κ.

limits the reference speed of a station value, based on the curvature, so that a certain centripetal acceleration value is not exceeded.

Traffic Light Agent The traffic light agent is in charge of interacting with traffic lights. Information from the environment perception module about the state of the traffic light is transformed into a reference speed profile. A common traffic light has four states, which are represented by the colors *Green, Amber*, *Red* and both *Red and Amber* at the same time. Note that there are more types of traffic lights (e.g. right/left turn), which will require special agents. Though for simplicity, the described agent covers only the common traffic light case:

- *Green* and *Red and Amber*: No action required, the reference speed profile is not affected by the traffic light.
- *Red*: A full stop at the stop line of the traffic light is required. This translates to $0m/s$ at the stop line.
- *Amber*: Based on the current speed and distance to the stop line of the traffic light a decision is needed, whether the vehicle can respect the traffic light. In case that the deceleration for achieving a full stop is beyond a given limit, the vehicle needs to speed up to pass the stop line as soon as possible. Otherwise, the same action as in *Red* is performed.

It is assumed that the information is incomplete in terms of duration of the states. Therefore, one cannot make a statement about the time left for a state transition from *Red and Amber* to *Green* and *Green* to *Amber*.

3.4 Interacting with Obstacles

Interacting with dynamic obstacles is a crucial part of planning. The future movement of other traffic participants has to be taken into account to provide a collision free trajectory. Thus, a prediction module is needed that assigns each obstacle one or more possible trajectories. The prediction might change over time, based on the given situation.

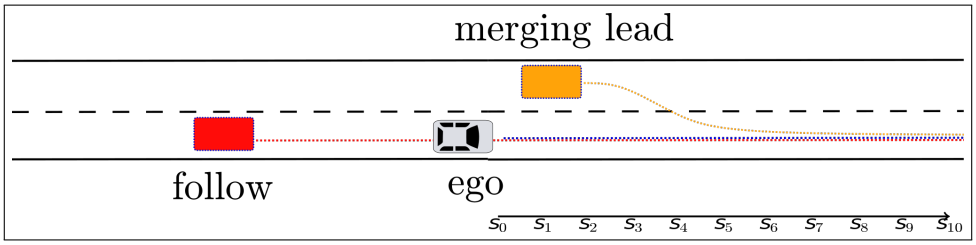
Assumption 1. *An obstacle is expected to follow one trajectory per lane most likely. The algorithm implemented in this thesis assumes this prediction to be true and will not take uncertainty into account.*

The obstacle prediction doesn't include acceleration, therefore all obstacles are predicted with constant speed. However, the implemented algorithm in this thesis is expected of being capable handling trajectories with non-constant acceleration.

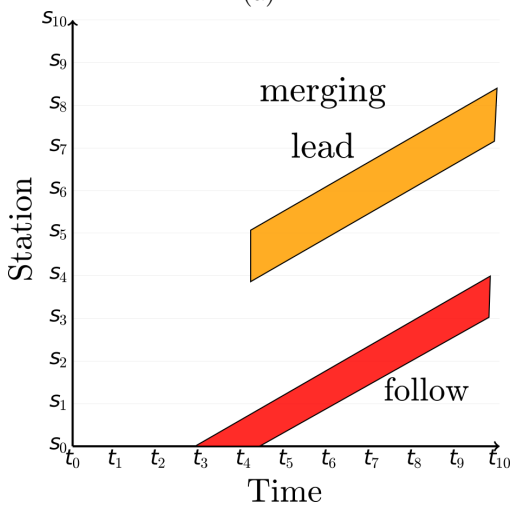
3.4.1 Station-Time Domain

While the reference speed profile is taking care of constraints formulated in the station-speed domain, dynamic obstacles need to be described with respect to time. The predicted obstacle might cross the path of the autonomous vehicle. It is important to know in which station range the obstacle will block the path and what the distance left to the obstacle is. Since dynamic obstacles travel over time, it is also important to notice, *when* the obstacle will block the path segment.

The station-time domain, also called space-time domain [23], provides this information. It is limited by time and station horizons as defined in section 3.2. An example is shown in Fig. 5. The station representing the path is on the vertical axis, while the time aspect is illustrated on the horizontal axis. Obstacles blocking the path are marked in red and orange. The *following* obstacle is located behind the autonomous vehicle. In case that the autonomous vehicle stands still, there will be a collision. A *leading* obstacle merges into the path of the autonomous vehicle at time $t = t_4$.



(a)



(b)

Figure 5: Example for understanding the station-time domain. Figure a shows the given scenario with two obstacles, which is visualized in the station-time domain in Fig. b. The obstacles are traveling with constant speed.

The obstacle interaction cost function described in section 4.3.2 makes use of the station-time domain for collision avoidance.

4 Discrete Optimization of Speed Profiles

This section aims to describe the selected approach to find an optimal speed profile on a discrete search space. Therefore section 4.1 describes the discretization of the search space by sampling. It also introduces a data structure for the sampled station-time. Section 4.2 continues with an explanation of the used graph-search algorithm, starting with the algorithm in general. Afterwards, the algorithm is adapted to the speed planning problem.

4.1 Discretizing the Station-Time Domain

Sampling the station-time domain creates a grid in the configuration space. Section 2 explained the motivation for discretizing the station-time domain by sampling: finding the optimum within a discretized search space yields not necessarily and allmost never to an optimal solution in the continous-search space, but it can serve as an initial guess for further convex optimization in continuous space.

This means that instead of solving the optimal speed planning problem in continuous station $\mathbb{S}$ and time $\mathbb{T}$ (as defined in section 3.2), a rough solution is found on a sampled station-time domain. Therefore, $\mathbb{S}$ and $\mathbb{T}$ are replaced by equations 21 and 22 respectively, using $\tau \in \mathbb{R}_+$ and $\sigma \in \mathbb{R}_+$ as sampling distances in station and time:

$$\mathbb{S}_\sigma := \{\sigma \cdot a | a \in [0, 1, ..., n_{station}]\} \tag{21}$$

$$\mathbb{T}_\tau := \{\tau \cdot b | b \in [0, 1, ..., n_{time}]\} \tag{22}$$

where $n_{station} \in \mathbb{N}_+$ and $n_{time} \in \mathbb{N}_+$ limits the number of sampling values on the grid in station and time. The maximum sampling value for station is calculated by:

$$n_{station} = \frac{h_{station}}{\sigma} - 1, \tag{23}$$

where $h_{station} \in \mathbb{R}_+$ is a horizon value that specifies the end of the planning space. The maximum sampling value for time n_{time} is determined in an identical manner by a time horizon $h_{time} \in \mathbb{R}_+$.

The resulting speed profile from the grid is a discrete mapping:

$$l_{\sigma\tau} : \mathbb{T}_\tau \to \mathbb{S}_\sigma \tag{24}$$

4.1.1 Station-Time Graph

The station-time graph represents the sampled station-time domain [9]. It consists of nodes $V_{\tau\sigma}$ and directed, weighted edges $E_{\tau\sigma}$. A node represents a tuple of station and time that is an option of reaching a station value at a certain time step. So one node stores the information about *when* the vehicle will be at a certain station value. Every combination of station and time is present in the nodes. Since the vehicle has to start planning at the current configuration, only station $s = 0$ is valid at $t = 0$.

More formally, the nodes of the directed station-time graph $G = (V_{\tau\sigma}, E_{\tau\sigma})$ describe a subset of the station-time domain:

$$V_{\tau\sigma} := \{(s,t) | (s,t) \in \mathbb{S}_\sigma \times \mathbb{T}_\tau\} \tag{25}$$

The root node, representing the start configuration at station $s = 0$ and time $t = 0$, is denoted by u_{start}. Accessing the station value of nodes $u \in V_{\tau\sigma}$ in general is depicted by $u.s$ and $u.t$ for time respectively. Nodes with $u.s = n_{station}$ or $u.t = n_{time}$ are denoted as *horizon nodes*.

The nodes in a station-time graph are connected by directed edges which show the reachability from one node to another. The set of edges $E_{\tau\sigma}$ can be stated as:

$$E_{\tau\sigma} \subseteq \{(u, u', w) | (u, u') \in V^2 \tag{26a}$$

$$\wedge\, u'.t - u.t = \tau \tag{26b}$$

$$\wedge\, u'.s \geq u.s \tag{26c}$$

$$\wedge\, w \in \mathbb{R}_+\} \tag{26d}$$

Node u has an edge to u' if they are in consecutive time steps and u' appears one time step after u. This is necessary, because a speed profile needs to be continuous in time. Another condition for an edge is that the head nodes station value $u.s$ is at most equal to the tail node $u'.s$, because the speed v needs to be non-negative as defined in equation 6. Equation 26b relates to time-continuity and line 26c enforces non-negative velocity.

An edge is weighted by the costs w. Costs of edges are calculated by the cost function J_E:

$$J_E : E_{\tau\sigma} \to \mathbb{R}_+ \tag{27}$$

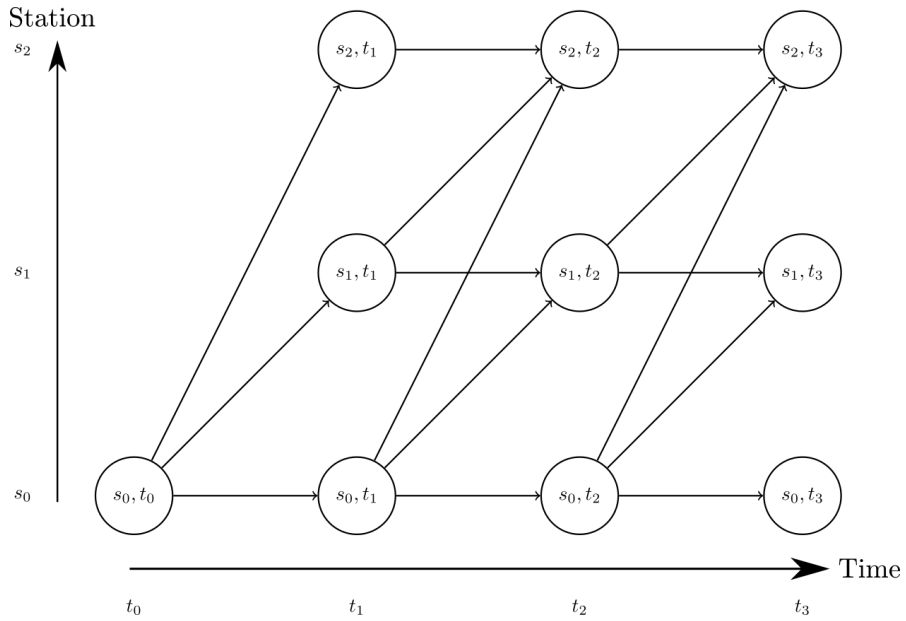


Figure 6: A small example that shows the representation of a station-time domain as station-time graph for certain values in time $t_0, .., t_3$ and station $s_0, .., s_2$.

Figure 6 shows a toy example of a station-time graph with $n_{time} = 3$ and $n_{station} = 2$.

The station-time graph is acyclic, due to equation 26b. The equation prevents cycles in the time domain which is natural, because time is a linear ordering. Equation 26c creates a linear ordering in the station domain.

Combining both equations creates a topological order on the graph based on the edges. A topological order allows a linearized processing order. One can consider a topological order of a graph to be a horizontal line of vertices where all edges go from left to right [27, p. 612]. An example for the linearization of the ordering on an station-time graph is shown in Figure 7.

It is important to note that a station-time graph is a directed acyclic graph (**DAG**). Since the cost function J_E used for edge weights assigns values in $\mathbb{R}_+$, it has also non-negative edge weights. These properties will be used in section 4.2.

Matrix Representation

The station-time graph G can be represented as matrix $M_{\sigma\tau}$ of dimensions $s_n \times t_n$. Each entry in the matrix represents one node $u \in V_{\tau\sigma}$. This representation will be used for

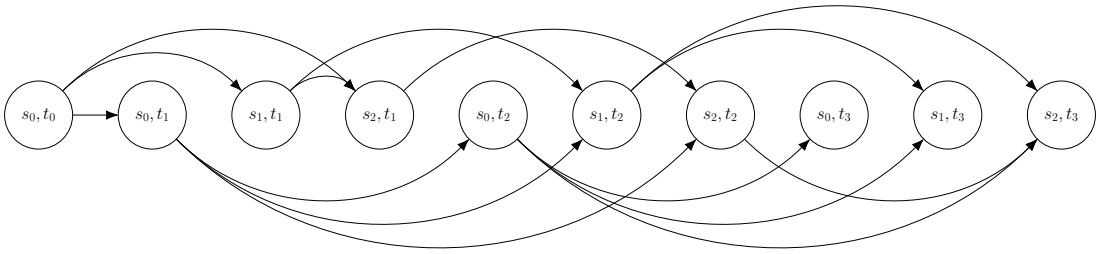


Figure 7: Topological ordering of the station-time graph of Fig. 6 by equation 26.

finding shortest paths in section 4.2. The following shows an example of the matrix representation of the station-time graph from Fig. 6:

$$M_{\tau\sigma} = \begin{pmatrix} (s_2,t_0) & (s_2,t_1) & (s_2,t_2) & (s_2,t_3) \\ (s_1,t_0) & (s_1,t_1) & (s_1,t_2) & (s_1,t_3) \\ (s_0,t_0) & (s_0,t_1) & (s_0,t_2) & (s_0,t_3) \end{pmatrix}$$

Matrix entries $(s_1,t_0),\ldots,(s_{n_{station}},t_0)$ are pseudo entries that are added in the column of the start configuration $u_{start} = s_0t_0$. Since all speed profiles need to start from the start configuration, the pseudo entries are ignored.

Edges are not represented in the matrix, because they can be derived from the element indices respecting equation 26c. Therefore, the matrix is not related to an adjacency matrix of a graph.

Note that the matrix has it's origin (s_0,t_0) in the bottom left corner, instead of the commonly used top left corner. The matrix is better comparable to the graph representation and station-time domain due to this conversion.

Retrieving Motion Properties

The motion properties are annotated in the nodes of the station-time graph. Since they interdepend on the predecessor, they need to be updated when the predecessor changes.

Using equation 9, one can derive the motion properties for an edge $(u, u') \in E_{\tau\sigma}$:

$$u'.v = v(u'.s - u.s, \tau) \tag{28}$$

$$u'.a = a(u'.v - u.v, \tau) \tag{29}$$

$$u'.j = j(u'.a - u.a, \tau) \tag{30}$$

These dynamics will be used in the cost functions in section 4.3.2.

The start configuration u_{start} needs to be initialized in the first planning iteration by the actual vehicle state. In following planning iterations, u_{start} is gained from the trajectory of the last planning cycle.

Parameters for Sampling and Horizon

Choosing the sampling distance parameters τ for the time and σ for the station domain is an important task. The parameters have a direct impact on the available speed and acceleration values, since both depend on station and time. One should keep in mind that reaching certain maximum speed becomes impossible with some τ and σ values. Therefore, τ and σ should be chosen with respect to expected maximum speed values.

Particular attention should be given to the horizon. Based on the speed of the vehicle, adjusting $h_{station}$ and h_{time} is necessary. For example, when driving with a speed of $v = 5m/s$, a station horizon of $h_{station} = 50m$ and time horizon of $h_{time} = 6s$ are sufficient, because the vehicle won't reach the station horizon in that time frame. But with a speed of $v = 14m/s$, meeting the station and time horizon is only reachable by braking.

An evaluation for finding appropriate τ and σ is presented in section 5.5.

4.1.2 Distance Calculation for Collision Detection

Avoiding a collision is the main task in planning with dynamic obstacles. To achieve this task, it is important to know the distances to obstacles. This section describes the used method for calculating the distances between obstacles and the autonomous vehicle at every point in time. A common approach to perform a collision check is the approximation of a vehicle as circles [28]. They replace the bounding box of the vehicle by a number of circles. Based on the number of circles, the approach tends to suffer from

an over- or underfitting of the bounding box. Another approach is using the bounding box directly, calculating the euclidian distance between bounding boxes that are shifted on the trajectory [29]. Both approaches don't differentiate between a longitudinal and a lateral safety distance. The problem is here that a car in a neighboring lane to the autonomous vehicle can be passed with a much lower safety distance than a leading car. Ignoring the differentiation leads to a poor compromise.

The calculation for the distance tries to separate obstacles into lateral and longitudinal obstacles. The method is split into several stages, from a rough estimation for filtering to exact distances:

Filtering non-interfering obstacles: As a first step, all obstacles are filtered so that only obstacles with a path-crossing trajectory are inspected further. Traffic participants are interpreted as points in this step. Obstacles and vehicles are shifted along their trajectories to their position at each sampled time step. In case of reaching a certain distance threshold that represents a lateral safety distance, the obstacle and its trajectory are marked for further investigation. Thereby, non-interfering obstacles are ignored in the following steps.

Circular Approximation: After filtering, only path-interfering obstacles are left over. The point representation is inflated to a circle, enclosing the bounding box of participant. Once again, the vehicle and the obstacles are shifted to their predicted positions. In case that the distance from the circles is lower than the longitudinal safety distance, the points are added for inspection in the next step.

Geometry Calculation: As a last step, the euclidean distance of the filtered positions is calculated based on the bounding boxes.

The described approach manages to calculate the distances to dynamic and static obstacles. However, it is a rather simple approach that has obvious flaws. The main flaw is the filtering in the first step: due to the point obstacle representations, the obstacles are expected to have a parallel orientation to the vehicle. This is needed to filter lane-following obstacles in neighbor lanes, but is a very limited approach in general. A rotation of the obstacle can't be detected, because of the point representation. Based on the time sampling τ, a point in time with a guaranteed collision can be specified. But the specified time step is not the exact time of collision, because of the sampling nature. For this reason, an obstacle that enters and leaves the path of the autonomous vehicle within a time $t < \tau$ will not be detected.

The calculated distance will be used in section 4.3.2 in the obstacle interaction cost function to avoid collisions.

4.2 Graph-Search with Dynamic Programming

After introducing the station-time graph, this section covers the algorithm for finding the optimal solution in the discretized search space. Note that an existing edge is not equivalent to a feasible path. The graph may contain unfeasible edges. One needs to clarify about solutions, feasible solutions and optimal solutions in terms of the station-time graph.

4.2.1 Shortest Paths in the Station-Time Graph

A path $\beta := [u_i, .., u_j]$ in the station-time graph is a list of nodes $u_i, .., u_j \in V_{\tau\sigma}$, $0 \leq i < j \leq n_{time}$ connected by edges, where u_i is the start node of the path and u_j the end node of the path. The nodes can be accessed by $\beta[r]$, where r is the r-st element of the path. Each node $u \in V_{\tau\sigma}$ maintains a predecessor $u.\pi \in V_{\tau\sigma}$ that is another node $u.\pi \neq u$. Both nodes have to be connected by an edge. Excluded is the root node u_{start}, because it has no predecessor and is therefore marked as $u_i.\pi = \text{NIL}$.

There are some definitions required for a path to meet the problem definition of section 3.2 and define an optimal feasible solution in terms of the discrete speed planning problem:

1. A solution is optimal, if there is no other solution with smaller costs.
2. Any path starting from the start configuration u_{start} and ending at a horizon node is a solution $l_{\tau\sigma}$.
3. A subsolution is any path starting with the start configuration u_{start}.
4. To meet the full stop planning perspective from section 3.2, a solution and subsolution must not contain a path with $u.s \neq 0$ and $u.\pi.s = 0$ and $u.\pi \neq u_{start}$.
5. A path is feasible, if all nodes are feasible and therefore fulfill the set of constraints C.

A path β and hence a solution $l_{\tau\sigma}$ can therefore be evaluated by the sum of edge weights of the given solution:

$$J_{\tau\sigma}(\beta) = c_{fromRoot}(\beta[0]) + \sum_{i=0}^{|\beta|-1} J_E(\beta[i], \beta[i+1]) \tag{31}$$

where the constraint $c_{fromRoot}$ evaluates the start node of the path $\beta[0]$ and ensures that a solution begins with the start configuration u_{start} [30, p. 49]:

$$c_{fromRoot}(u) = \begin{cases} \infty, & u \neq u_{start} \\ 0, & u = u_{start} \end{cases} \tag{32}$$

Consequently from equation 31, the optimal solution is a path from the start configuration to one of the valid end nodes with minimal costs. Thereby it encodes a speed profile since it allows a mapping from time to station.

To find the optimal solution, one needs to calculate all possible paths to the end nodes and choose the best. Therefore, it is obligatory to know the shortest paths to every node from the start node. For this reason, the speed planning problem in the station-time graph is a shortest paths problem, more precisely a single-source shortest paths (**SSSP**) problem.

4.2.2 Dynamic Programming Method

Dynamic Programming is a technique to solve complex optimization problems. It splits the complex problem into overlapping subproblems [27, p. 359]. The subproblems are solved optimally to build an optimal solution for the problem [27, p.379]. Solutions of overlapping subproblems are saved and used with a lookup table. Thereby, dynamic programming takes advantage of reoccurring subproblems, since a subproblem is solved only once. Looking up stored solutions is expected in constant time and therefore more efficient than solving the subproblem again.

A optimization problem needs two properties in order to be applicable for dynamic programming [27, p. 378ff.]:

1. The problem contains overlapping subproblems.

2. The problem has an optimal substructure.

An overlapping subproblem is a problem that has to be solved in an algorithm multiple times [27, p. 384], which differentiates dynamic programming from divide-and-conquer: Algorithms using divide-and-conquer like MergeSort tend to split problems into subproblems that won't occur multiple times.

A problem has an optimal substructure if the optimal solution incorporates optimal solutions to contained subproblems [27, p. 362]. Consequently, all subproblems that are covered by the optimal solution must have optimal subsolutions within the optimal solution.

There are two approaches for dynamic programming for shortest path problems in graphs:

1. Backward value iteration
2. Forward value iteration

Backward value iterations finds optimal paths to all start nodes based on an end node. Therefore, the end configuration is fixed. Forward value iteration is used to find an optimal path in a graph from a start configuration to all other nodes in the graph [30, p. 48].

The next section will explain an algorithm that makes use of dynamic programming to solve the shortest paths problem.

4.2.3 Single-Source Shortest Paths Algorithm

Solving the single-source shortest paths problem in a directed acyclic graph is a well known problem. Calculating naivly all possible solutions is exhaustive. Therefore dynamic programming is used to evaluate all valid solutions. This section describes an approach to solve the **SSSP** problem in a directed acyclic graph.

Every **SSSP** problem with a graph $G = (V, E)$ with non-negative edge weights and directed edges can be solved by the Dijkstra algorithm [27, p. 658] in $O(V \cdot logV + E)$. But due to the topological order of the **DAG** it can be done faster, resembling breadth-first search with dynamic programming.

Algorithm 1: DAG Single Source Shortest Paths [27, p. 655ff.]

Data: Graph $G(V, E)$, Cost Function J_E, StartVertex u_{source}
Result: Shortest Paths from u_{source} to each $u \in G.V$

```
1 SortTopological(G.V)
2 Initialize(G, u_source)
3 for each vertex u taken in topologicalSortedOrder(G) do
4 |   for each u' ∈ G.adjacent(u) do
5 |   |   Relax(u, u', J_E)
```

The shortest paths problem in a graph has an optimal substructure [27, p. 645] which is needed for dynamic programming. Computing the shortest path to an end node requires to evaluate the same paths of the graph over and over again. Since an optimal solution depends on all end nodes, the paths reoccur even more often. It becomes clear that evaluating paths are overlapping subproblems of the station-time graph. Therefore dynamic programming can be used to calculate the **SSSP** efficiently.

Algorithm 1 is proven to find the shortest paths from a start node s to all other nodes in the graph [27, p. 656]. Using the properties of the **DAG** and dynamic programming, it creates a shortest path tree from a root node in $\Theta(V + E)$ [27, p. 655].

For searching in the station-time graph forward value iteration is required, because the start configuration of the car is fixed and a solution reaching either the time or station horizon is needed.

The main idea of algorithm 1 is to process each node exactly once in a topological order. Thus, the graph is sorted in topological order in line 1. Initializing the nodes of the graph in line 2 is carried out to algorithm 2. Each node $u \in V$ from the graph has a predecessor that is initialized with NIL. The predecessor is updated during the algorithm so that it represents the shortest path from $u.\pi$ to u.

The costs to reach u from $u.\pi$ are regarded as $u.w$ and denote a current estimate for the shortest path costs to u. Both will be updated by algorithm 3.

Algorithm 2: Initialize [27, p. 648]

Data: Graph G, StartVertex s
Result: Initialized Graph G

1 **foreach** $u \in G.V$ **do**
2 $\quad u.w = \infty$;
3 $\quad u.\pi = \text{NIL}$;
4 $s.w = 0$;

Algorithm 3: Relax [27, p. 649]

Data: Vertex u, AdjacentVertex u', CostFunction J_E
Result: relaxed edge (u, u')

1 **if** $u.w + J_E(u,u') < u'.w$ **then**
2 $\quad u'.w = u.w + J_E(u, u')$;
3 $\quad u'.\pi = u$;

After initializing the nodes, each node of the graph is examined once. Thereby all nodes of the graph are processed in topological order (line 3). All adjacent nodes u' of a node u are relaxed in algorithm 3. Relaxing an edge means to test whether u contributes to a shorter path to u' than the ones previously found. In case of a shorter path is the *cost-to-reach* and the predecessor updated to represent $u'.w$ and $u'.\pi$ respectively. Note that the algorithm selects in case of two paths with equal cost the one that was found first.

To obtain the shortest path to a node u , one can simply traverse backwards using the predecessors $u.\pi$.

4.3 Finding Optimal Speed Profiles in the Station-Time Graph

The basic algorithm 1 is customized in the following section to adjust to the discrete speed planning problem. The *Initialize* function is slightly adapted to initialize a matrix instead of a graph, but the functionality is identical (see algorithm 5). The *Relax* function is reused (see algorithm 6) with the addition to update the motion properties as defined in paragraph 4.1.1.

Algorithm 4 shows the pseudo code to find an optimal solution in a station-time graph. The matrix representation of the station-time graph $M_{\sigma\tau}$ is used, because it is a compact data structure and provides a simple adjacency mechanism. Note that the algorithm works with the discrete step indices rather than real station and time values when accessing values from the matrix.

Since equation 26 defined a topological order for the station-time graph, an explicit ordering step is not required. The initialization of the matrix in line 1 works as in

Algorithm 4: Optimal Solution in Station-Time Graph

Data: Station-Time Graph $M_{\sigma\tau}(n_{station} \times n_{time})$, Cost Function J_E
Result: Shortest Path to Leaf in $M_{\sigma\tau}$

```
1  InitializeMatrix(M_στ);
2  for ( i = 0; t ≤ n_time − 1; i = i + 1 ) {
3      for ( j = 0; j ≤ n_station; j = j + 1 ) {
4          u = M_στ(j, i);
5          if u.w == ∞ then
6              Continue;
7          if u.s − u.π.s > 0 then
8              for ( k = j; k ≤ n_station; k = k + 1 ) {
9                  u′ = M_στ(k, i + 1);
10                 RelaxMatrix(u, u′, J_E);
11         else
12             u′ = M_στ(j, i + 1);
13             RelaxMatrix(u, u′, J_E);
14 return Backtrace(M_στ);
```

Algorithm 5: InitializeMatrix

Data: Station-Time Matrix $M_{\sigma\tau}(n_{station} \times n_{time})$
Result: Initialized Matrix $M_{\sigma\tau}$

```
1  for i = 0; i ≤ n_time; i = i + 1 do
2      for j = 0; j ≤ n_station; j = j + 1 do
3          u = M_στ(j, i);
4          u.w = ∞;
5          u.π = NIL;
6  u = M_στ(0, 0);
7  u.w = 0;
```

algorithm 2 with the exception that the start node is always the start configuration (see algorithm 5). Hence, an explicit start node is not required anymore.

The next step is to relax the edges in the station time graph. Thus, each node and edge is inspected once. The first two *for-loops* in line 2 and 3 are responsible for visiting all

Algorithm 6: RelaxMatrix

Data: Vertex u, AdjacentVertex u', CostFunction J_E
Result: relaxed edge (u, u')

```
1 if u.w + J_E(u,u') < u'.w then
2     u'.w = u.w + J_E(u, u');
3     u'.π = u;
4     updateMotionProperties(u, u');
```

nodes, except for the last time step. Hence, the last column of the matrix is skipped here. This happens because these nodes are the leafs of the graph.

Line 4 retrieves the currently inspected node u.

Infinity costs of u stand for a node u that wasn't successfully relaxed in the last time iteration. It is not necessary to continue searching from an unrelaxed node, because no path with reasonable costs to u exists and therefore no path from u to a leaf can exist (line 5 and 6).

Line 7 tests, whether u plans a full stop.

In case that u is not planning a full stop, all edges of u are relaxed to find shortest path to adjacent nodes u'. This is represented in line 8-10. The *for-loop* in line 8 enforces equation 26c since $u'.s - u.s$ must be positive for a positive velocity.

On the other hand, one needs to ensure the requirements of full stop handling from section 3.2. The only valid relaxation is therefore a resumption of the full stop. Hence, a relaxing of the edge with equal station value to u is made (line 12 and 13).

4.3.1 From Shortest Paths to Speed Profiles

After the calculation of all shortest paths, a backtrace is made in algorithm 7 to gain the shortest path to a leaf from $M_{\tau\sigma}$. The first step is to find the minimal cost of all leafs. Thus, the cheapest leaf from station and time horizon is selected.

The abstract function *minCost* returns the cheapest leaf from the top row of $M_{\tau\sigma}$. If the top row doesn't have an entry with valid costs, the previous row is inspected. Thereby

Algorithm 7: Backtrace

Data: Station-Time Graph as Matrix $M_{\sigma\tau}(n_{station} \times n_{time})$
Result: Shortest Path β from u_{start} to cheapest Leaf in $M_{\sigma\tau}$

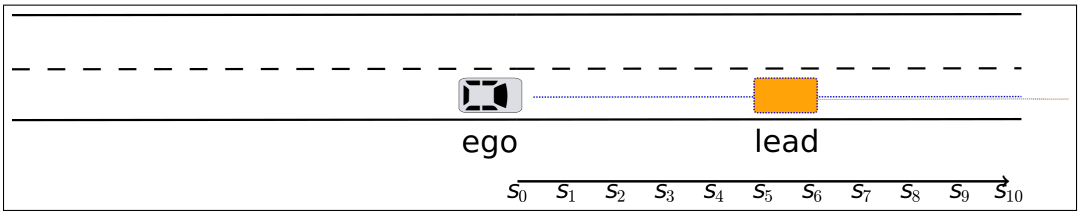
```
1  min_s = minCost(M_στ);
2  min_t = minCost(M_στ^T);
3  if min_s.s == n_station then
4      u_στ = minCost(min_s, min_t);
5  else
6      u_στ = min_t;
7  β = [u_στ];
8  while u_στ.π ≠ M_στ(0,0) do
9      u_στ = u_στ.π;
10     β.appendFront(u_στ);
11 return β;
```

is min_s as cheapest station horizon leaf calculated by $minCost(M_{\tau\sigma})$ in line 1 and min_t as cheapest time horizon leaf accordingly by $minCost(M_{\tau\sigma}^T)$ (line 2).

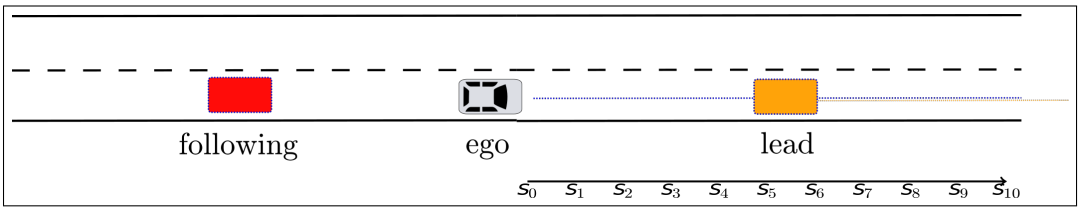
In case that min_s is not reaching the station horizon $n_{station}$, it should be ignored to reach at least one horizon (line $3-5$). An example of an unreachable station horizon is given in Fig. 8a.

Note that β might not be a solution but a subsolution, because all solutions are unfeasible. Figure 8b shows a scenario, where neither the station nor the time horizon can serve as a solution, since all paths to the horizon nodes are infeasible. This is an exceptional situation, where further reaction is needed. Based on the legitimization of the autonomous vehicle, a decision for situation handling has to be made. The vehicle might signal the driver to hand over, so that the responsibility for resolving the situation is in human competence. Another option is that the vehicle executes the suboptimal solution to react to the situation but signals warnings.

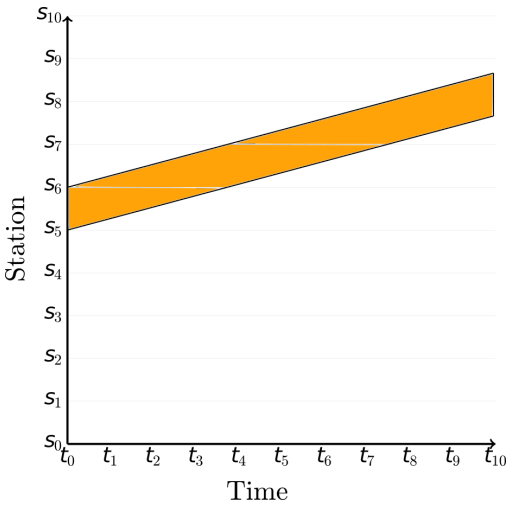
Finally, the shortest path β is created by the node $u_{\tau\sigma}$ and its shortest path predecessors (line $8-10$).



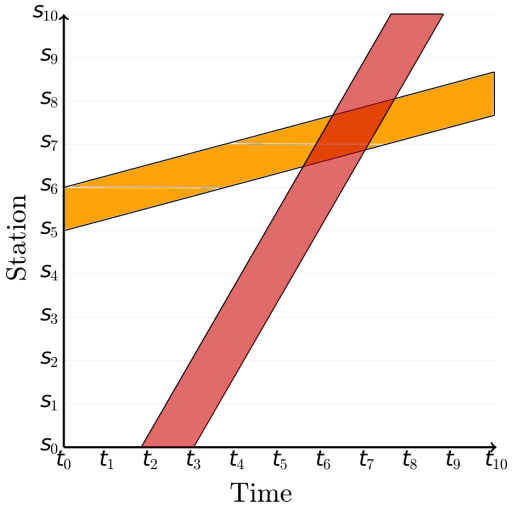
(a)



(b)



(c) Station-Time domain of (a).



(d) Station-Time domain of (b)

Figure 8: Visualization of two scenarios where horizons (s_{10} and t_{10}) can't be reached. Figure a shows a situation of a slow driving leading obstacle that prevents the ego vehicle from reaching the station horizon. The station horizon can't be reached without colliding with the leading obstacle. Figure b intensifies the situation even more, because a fast driving following obstacle would cause a collision before reaching the time horizon.

Post-Processing

The shortest path to a leaf is equal to an optimal speed profile $l_{\tau\sigma}$. Linear interpolation between the extracted nodes helps to define a continuous speed profile l based on $l_{\tau\sigma}$. A reverse mapping from $l_{\tau\sigma}$ to the space path P annotates configurations with time stamps

from the speed profile. The resulting speed profile can be used for further optimization.

4.3.2 Cost Functions

In order to evaluate costs to reach one node from another and for entire node paths, a cost function is needed. The cost function J is abstractly defined in equation 12. Choosing cost functions is an important task, since they define the optimality of a solution.

This section shows the design of the used cost function in the presented algorithm. To evaluate solutions and subsolutions, the cost function is split into several terms that represent requirements of section 3.1:

$$J_E(u, u') = \gamma_V \cdot J_V + \gamma_A \cdot J_A + \gamma_J \cdot J_J + \gamma_O \cdot J_O \tag{33}$$

Thereby is $\gamma_\square$ a weighting variable associated with a cost term $J_\square$. It is used to change the impact of each element of the cost function. The cost terms are:

- J_V: Reference Speed Cost Function
- J_A: Longitudinal Acceleration Cost Function
- J_J: Longitudinal Jerk Cost Function
- J_O: Obstacle Interaction Cost Function

These terms are explained in detail in the following paragraphs. The constraints C from section 3.1 are used in the cost functions to evaluate feasibility. The cost functions translate unfeasible solutions that violate the given constraints into infinite costs.

Reference Speed Cost Function

The speed cost function J_V is responsible for punishing discrepancy between the planned speed v of the vehicle and the reference speed v_{Ref} at a given station. In general, the vehicle should gain the exact reference speed to travel most efficiently, by making the most possible progress. The speed is gained from the tail node $v = u'.v$, while the reference speed is collected from the reference speed profile at the tail nodes station $v_{Ref} = rsp(u'.s)$.

One common cost function is defined as $J_V(v, v_{ref}) = |v_{Ref} - v|$ in [23, 31]. It is simple to understand and can be used in a linear optimizer. However, the function lacks of differentiation between exceeding and undershooting of the reference speed.

Using a free-form cost function provides the advantage of modelling various cases [23]. Modelling J_V as free-form cost function can be used to distinguish between overshooting and falling short of reaching the reference speed[9].

Therefore is $v_{so} = \frac{v - v_{Ref}}{v_{Ref}}$ used to calculate the offset between the planned and reference speed. Scaling the offset allows to relativize the offset based on the reference speed. For example, based on $30m/s$, an offset of $3m/s$ might be tolerated. On the other hand, when driving with $5m/s$, the offset is huge compared the actual speed. Because of this the offset is scaled by the reference speed which introduces a singularity in case of a full stop request. Respecting a full stop is important, therefore it is a hard constraint. This is stated in equation 35b.

Since the reference speed is a suggestion for an upper limit of speed, overshooting should imply more costs than undershooting. That's why equation 35c, which represents overshooting, is quadratic and equation 35d, which represents undershooting, is only linear. To adjust these factors, two additional weight parameters α_1 and α_2 are introduced.

Equation 35a encodes a constraint $c_{validVelocity} \in C$ which verifies that the planned speed is within the physical limitations $[v_{\min}, v_{\max}]$ of the vehicle:

$$c_{validVelocity}(v) = \begin{cases} 1, & \text{if } v_{\min} \leq v \leq v_{\max} \\ 0, & \text{else} \end{cases} \tag{34}$$

Putting all properties together, the cost function for vehicle speed can be regarded as:

$$J_V(v, v_{Ref}) = \begin{cases} \infty, & \text{if } c_{validVelocity(v)} = 0 & \text{(35a)} \\ \infty, & \text{else if } v_{Ref} = 0 \wedge v \neq 0 & \text{(35b)} \\ \alpha_1 \cdot v_o^2, & \text{else if } v_o > 0 & \text{(35c)} \\ \alpha_2 + v_o, & \text{else if } v_o \leq 0 & \text{(35d)} \end{cases}$$

Longitudinal Acceleration Cost Function

The longitudinal acceleration cost function J_A interacts as a counterbalance to the reference speed cost function. It receives the acceleration from a tail node: $a = u'.a$.

Using a free-form cost function allows to split the valid acceleration range of the vehicle $[a_{\min}, a_{\max}]$. So the acceleration range is divided into an inner interval between $[a_{Soft\,\min}, a_{Soft\,\max}]$, where $[a_{\min} < a_{Soft\,\min} < a_{Soft\,\max} < a_{\max}]$. The inner interval can be used as a soft-constraint which defines the limits of comfortable driving. Exceeding the comfortable might be indispensable in some situations like an emergency stop, but this shouldn't happen without a strong reason.

A constraint $c_{validAcceleration} \in C$ similar to $c_{validVelocity}$ prevents an unfeasible speed profile:

$$c_{validAcceleration}(a) = \begin{cases} 1, & \text{if } a_{\min} \leq a \leq a_{\max} \\ 0, & \text{else} \end{cases} \tag{36}$$

To gain a smooth starting and stopping of the vehicle, acceleration a in the comfort area is also affected by the speed v. At lower speed, acceleration is more expensive than at higher speed. This requirement is shown in equation 37b, where the acceleration is divided by the maximum of the speed and a threshold $\psi \in \mathbb{R}+$. The threshold is used to cap the costs for a certain speed to prevent extreme costs with very low speed.

Leaving the comfort range should correspond to high costs, since it shouldn't be used on a regular basis. Therefore is equation 37c an exponential function.

The entire longitudinal acceleration cost function can now be formulated as:

$$J_A(a, v) = \begin{cases} \infty, & \text{if } c_{validAcceleration}(a) = 0 & \text{(37a)} \\ \dfrac{|a|}{max(v, \psi)}, & \text{else if } a_{Soft\,\min} \leq a \leq a_{Soft\,\max} & \text{(37b)} \\ e^{|a|}, & \text{else} & \text{(37c)} \end{cases}$$

The properties of the acceleration cost function are visualized in Fig. 9. Figures 9a and 9c focus on the comfort area of $a_{Soft\,\min} = 0.7$ and $a_{Soft\,\max} = 1.0$ to demonstrate the common function behavior. On the other hand Fig. 9b and 9d point out the cost gap when leaving the comfort area.

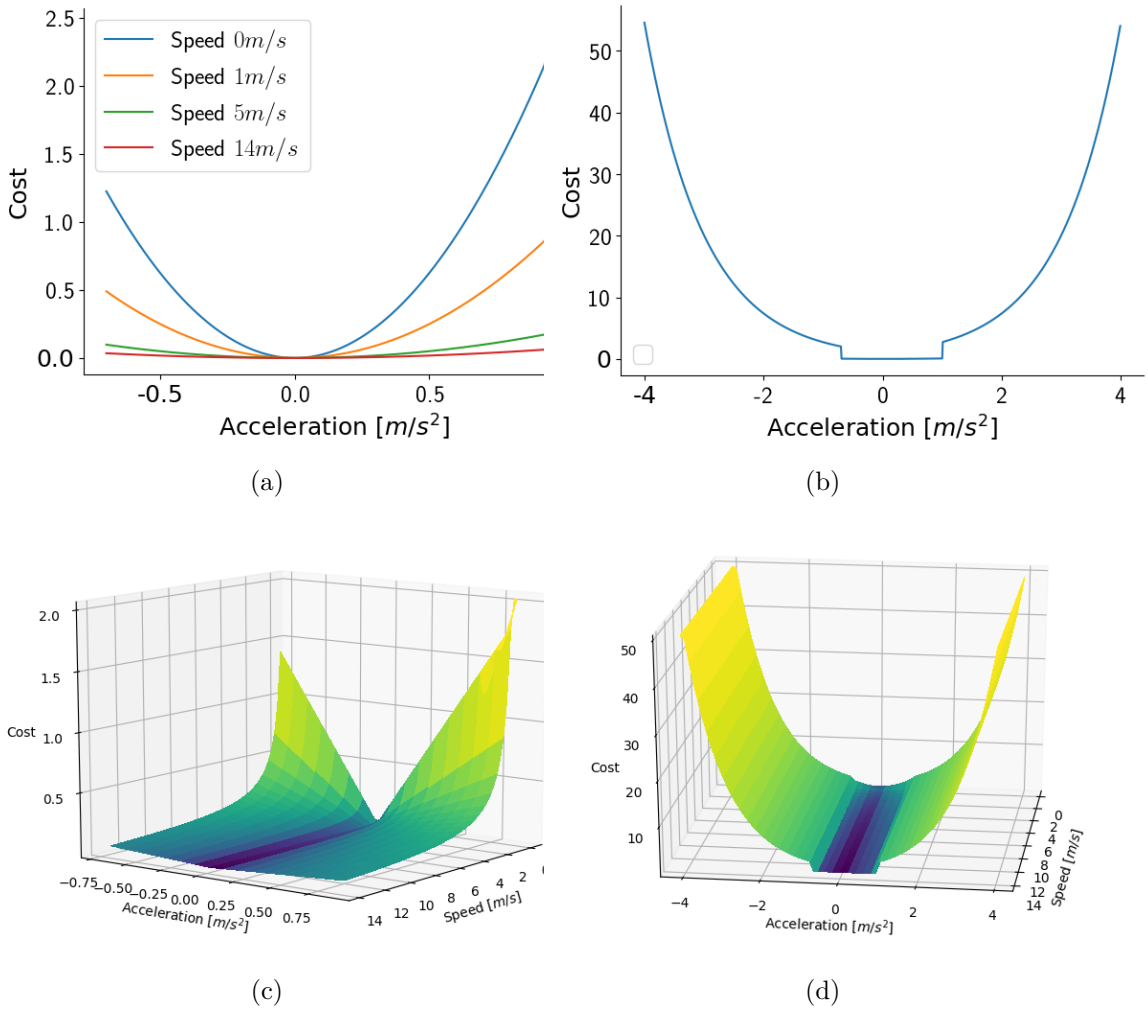
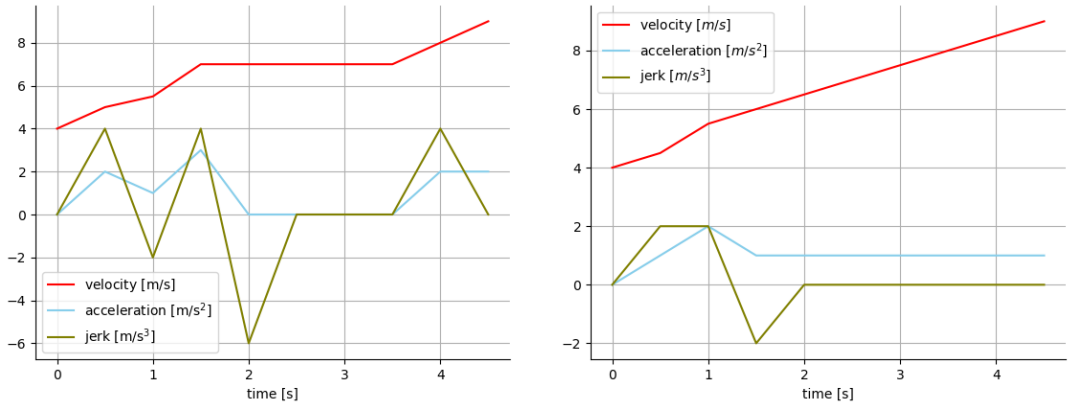


Figure 9: Visualization of the acceleration cost function. Colors correlate to costs. Fig. 9a shows the cost in the comfort area $[-0.7m/s^2, 1m/s^2]$ for different speed values, while Fig. 9b gives an overview of the entire function in with $a_{\min} = -4m/s^2$ and $a_{\max} = 4m/s^2$ with a single speed. The surface plots in Fig. 9c and 9d demonstrate the costs for the comfort area and the complete cost function for typical speed for inner city driving of $v \in [0, 14]m/s$.

Longitudinal Jerk Cost Function

Jerk denotes the rate of change of acceleration. Humans feel discomfort with jerk [32]. For example, approaching with a vehicle like a roller coaster is unpleasant and includes risks of health damages.

Figure 10 shows two different speed profiles for the same start and end speed with



(a) Speed profile with high jerk.

(b) Speed profile with low jerk.

Figure 10: Two speed profiles accelerating from $4m/s$ to $9m/s$ with the same amout of acceleration but different jerk.

different jerk values. The amount of acceleration is equal in both images, while the jerk varies. In the first image the jerk is a lot higher than in the second image.

The jerk is retrieved from tail node: $j = u'.j$. To comply the human need for comfort, the jerk cost function penalizes jerk in general:

$$J_J(j) = j^2 \tag{38}$$

Since there aren't any constraints to consider, it is quite simple. Having another look at the example in Fig. 10, the longitudinal jerk cost function evaluates the first solution with costs of 88 which exceeds the costs of the second solution of 12 by far.

Obstacle Interaction Cost Function

Preventing collisions with obstacles is a challenging task. The obstacle cost function J_O is responsible for obstacle avoidance and keeping a certain longitudinal distance d for safety to other traffic participants. It uses the distances of the autonomous vehicle to other obstacles, represented as list Q. The distances are retrieved by the method

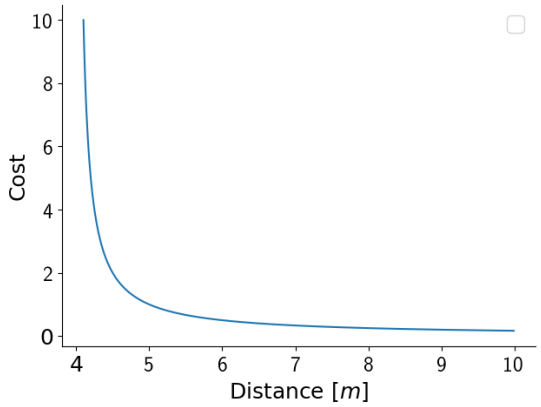


Figure 11: Obstacle cost function for a single obstacle with a safety distance $d = 4$ and $\gamma_O = 1$.

described in section 4.1.2.

$$J_O(Q) = \begin{cases} \infty, & \text{if } \exists q \in Q : q - d \leq 0 \quad (39a) \\ \sum_{q \in Q} \frac{1}{q - d} & \text{else} \quad (39b) \end{cases}$$

Entering the safety distance to an obstacle is treated as collision, to give a human safety driver time to react in case of a collision. As a result, the costs are infinity if the autonomous vehicle enters the safety distance. Because of this hardconstraint treatment, it is necessary to keep enough space to the safety distance. Equation 39b is displayed in Fig. 11. One can see that the costs increase significant when approaching the safety distance.

5 Evaluation

One of the open problems of motion planning is finding an accurate metric for evaluation. A common approach to evaluate a planner is the presentation of examples of the resulting trajectories in certain traffic scenarios [24]. The traffic scenario is visualized with the path of the autonomous vehicle, combined with images from the speed profile. One could name this method as a *snapshot* of the planner.

Others use three-dimensional plots to indicate the traveled trajectory of the road. Time is visualized by opacity [18], color [33] or as z-component [34]. However, most of these graphics lack of an understandable message, since the time dimension is hard to read and can be only guessed.

Althoff et al. [35] proposed an evaluation framework for motion planning called *CommonRoad*. The authors provide an extendable pool of traffic scenarios. The benchmarks are standardized by cost functions, vehicle parameters, vehicle model and obstacle predictions. Results of motion planning algorithms can be published in a reproducible way with an integrated visualization.

Unfortunately, *CommonRoad* doesn't support the used cost function structure used in this thesis. In addition, implementing the required adapters from the thesis to the framework would have exceeded the expenditures. Therefore, the method of *snapshots* is used.

The algorithm is implemented in `C++` using `ROS`[2]. All experiments are executed on an `Dell Precision 3520` with a `Intel® Xeon® CPU E3-1505M v6` processor with `3GHz`. The implementation is evaluated with a set of traffic scenarios in simulation. Based on the scenario, the properties of the speed profile are inspected. The resulting speed profile is compared to a *expected human-like speed profile*.

The first experiment evaluates the algorithm in the absence of obstacles. The focus is the aspect of comfortable driving, since no obstacles are involved. Scenarios with basic obstacles are considered in experiment 5.2. Starting with regular lane-keeping traffic, a first interaction with dynamic obstacles is evaluated. More complex obstacle interactions follow in experiment 5.3. A scenario of an obstacle entering the path of the vehicle will be evaluated, as well as an active lane change of the autonomous vehicle. Experiment 5.4 shows, how the algorithm responds to a scenario with an unavoidable

[2] https://www.ros.org/

collision. The experiment should evaluate whether the situation is detected correctly and how the resulting speed profile supports the decision.

Parameters of the algorithm used in simulation are listed in Tab. 1. The parameters are selected from empirical experimenting. All experiments use the same set of parameters.

h_{time}	$h_{station}$	τ	σ	γ_V	γ_A	γ_J	γ_O
$8s$	$125m$	0.5	0.125	0.575	0.2	0.2	0.05
α_1	α_2	a_{min}	$a_{soft\,\min}$	$a_{soft\,\max}$	$a_{\max}$	ψ	d
4	0.5	-7	-1	1	4	0.4	2.5

Table 1: Algorithm parameters used in simulation. A nomenclature can be found on page iv.

In addition, scenario 5.1.1 and 5.3.3 evaluate multiple parameter sets of σ and τ. Both sampling parameters are listed in Tab. 2. The time sampling parameter τ is modified to test different time resolutions in scenario 5.1.1 and 5.3.3. The runtime analysis in section 5.5 uses these parameter sets as well. The station sampling parameter σ is adjusted to τ, so that the planner can reach a minimal acceleration value of $0.5m/s^2$, when starting from a full stop.

Param Id	A	B	C	D	E	F	G	H	I
τ	1.0	0.9	0.8	0.7	0.6	0.5	0.4	0.3	0.2
σ	0.5	0.405	0.32	0.25	0.18	0.125	0.08	0.045	0.02

Table 2: Different time τ and station σ sampling parameter sets.

Visualizations of the station-time domain mark the blocked area of the obstacle plus the safety distance in red and the planned solution for the autonomous vehicle in blue. Thus, touching the red area doesn't lead to a violation of the safety distance.

5.1 Experiment: Obstacle-Free Driving

This experiment evaluates the planned dynamics of the autonomous vehicle in the absence of obstacles. The main focus is the interaction with the reference speed profile.

5.1.1 Scenario: Brake for Speed Limit

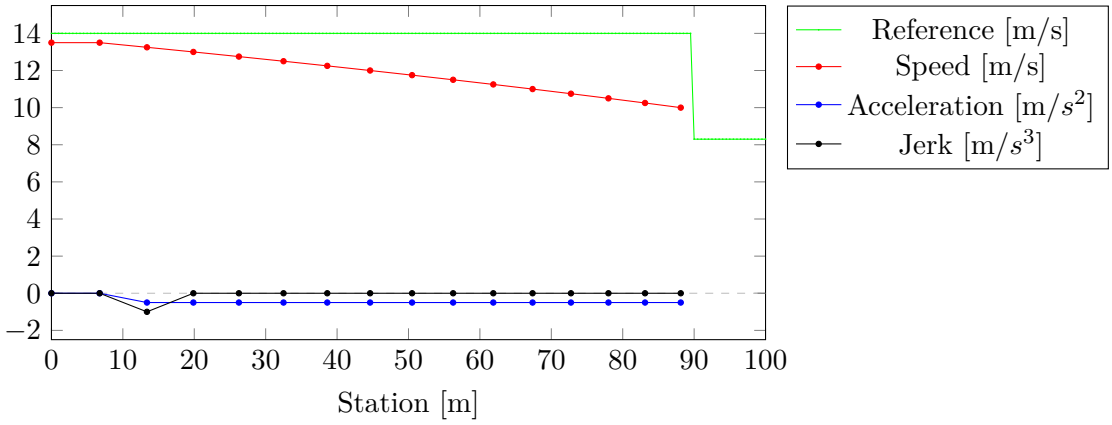
In this section, a braking maneuver due to a change of the speed limit is investigated. The vehicle is initially driving with a constant speed of $13.5m/s$ on a straight road. The reference speed profile describes a change of the reference speed from $14m/s$ to $8.3m/s$ within $90m$. Figure 12a visualizes the given scenario.

A speed profile is generated that ends in front of the descending edge in $90m$, not facing the adjustment of speed. The resulting speed profile can be found in Fig. 12b. The intention of a braking maneuver is visible, but the speed profile doesn't fully adjust to the change in the reference speed profile. However, a speed profile that considers the reference speed profile drop from the very beginning is preferable.

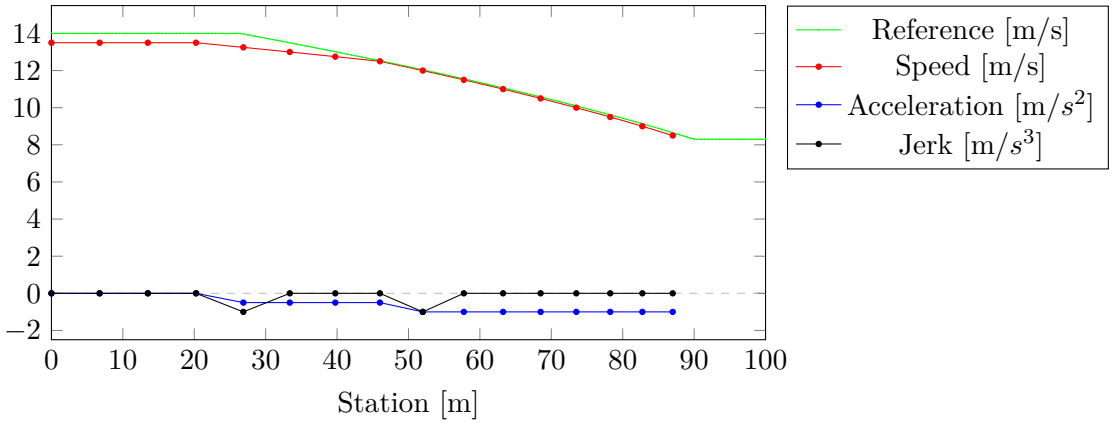
The algorithm is postponing an *evil* event, by using the flexible station horizon due to temporal planning. The evil event is related with high costs, which is reacting to the drop in the reference speed profile in this case: The cost function J_V punishes deviation from the reference speed profile. Let rsp_1 the reference speed profile from $0m$ to $90m$ and rsp_2 from $90m$ and further. Since the vehicle needs to slow down to prevent overshooting in rsp_2, it has to deviate from rsp_1. The algorithm can plan with a time horizon, hence it can adjust the length of the solution in station. A solution that reaches over a station of $90m$ is punished in rsp_1 and rsp_2, since the vehicle can't change its speed instantaneously (and shouldn't because of the acceleration cost function J_A). Therefore, a solution that will get punished only on rsp_1 is preferred by the algorithm. At some point in a future planning cycle, the algorithms will run out of such solutions and has to include solutions that are continued in rsp_2, so that the *evil* speed drop is handled. The effect of using the flexible station horizon to avoid a expensive action is called *postpone the evil* in the following.

A partial solution that weakens the *postpone the evil* effect, is to pre-process the reference speed profile. In the following experiments a smoothing algorithm is applied to the reference speed profile, similar to the one presented by Gu et al. [26]. It uses the comfort acceleration limits from section 4.3.2 as a heuristic. A result with a smoothed reference speed profile is visualized in Fig. 12c. Supporting the implemented algorithm by the smoothing heuristic improves the calculated speed profile. The speed profile provides a better solution, since the speed is closer to the reference speed profile and represents the change in speed much better.

(a) Traffic scenario



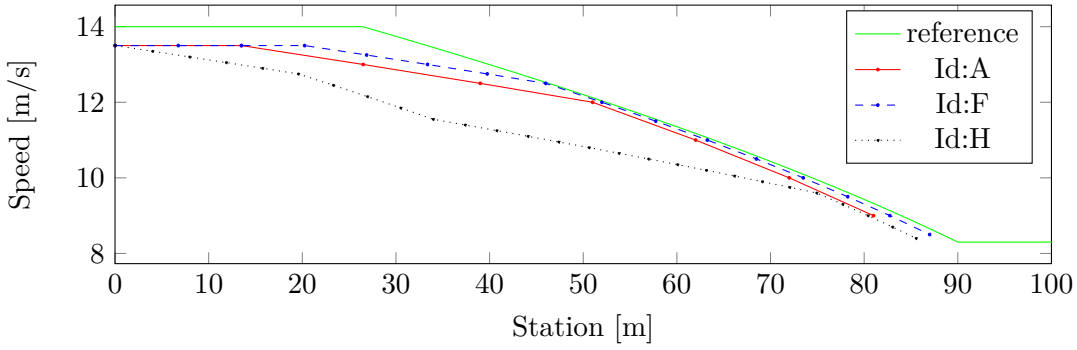
(b) Speed profile without smoothed reference speed profile



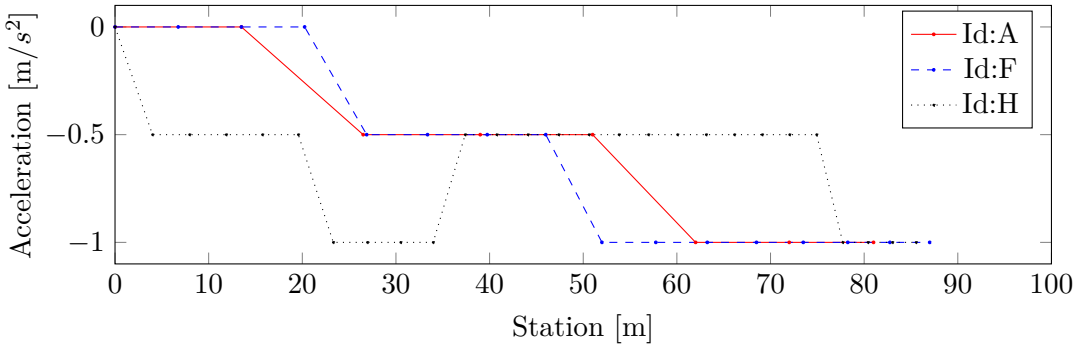
(c) Speed profile with smoothed reference speed profile

Figure 12: Speed profiles with and without smoothed reference speed profile of scenario 5.1.1. The scenario is visualised in Fig. a.

Figure 13 shows the outcome of the algorithm for three different sampling distances in station and time. Parameter sets with Id A and Id F stick quite close to the reference



(a) Speed for different sampling parameters



(b) Acceleration for different sampling parameters

Figure 13: Results of the **brake for speed limit** scenario for different sampling parameters of Tab. 2

speed profile (green), while the finest sampling distance, defined by parameter set with Id H, starts to decelerate earlier. Parameter set Id H is trading deviation from reference speed profile with less sudden deceleration.

5.1.2 Scenario: Exceeding the Reference Speed Profile

The autonomous vehicle can find itself in a situation, in which the current vehicle speed is significant above the reference speed profile. This scenario is a special case to the algorithm, since the initial configuration is not within the usual range. Such a situation may happen in case of a misdetection by the environment perception module. The vehicle has to adjust to the huge deviation from the reference speed profile by slowing down. The initial constant speed is $13.5m/s$, the reference speed profile requests a constant speed of $8.3m/s$.

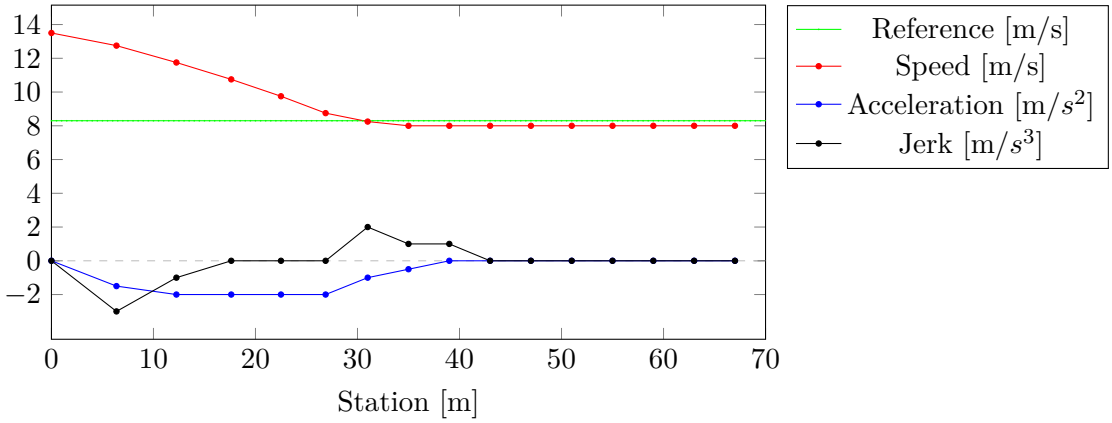


Figure 14: Planned speed profile in the **exceeding the reference speed profile** scenario.

The resulting speed profile is shown in Fig. 14. At first, a plan is generated that causes a slow down with acceleration of $-2m/s^2$ until it stops overshooting the reference speed profile. The planner is thereby leaving the acceleration comfort zone, trading a faster reaction with high acceleration. Once it is below the reference speed profile, the planner suggests to keep a constant acceleration of $0m/s^s$, keeping a slightly smaller speed than the reference speed profile. The vehicle needs $31m$ and $3s$ to reach the reference speed profile, closing a gap of $5.2m/s$.

The speed profile exceeds the acceleration comfort value range by using an acceleration of $-2m/s^2$. Based on the given situation, this seems acceptable to achieve a state that respects the traffic regulations. Slowing down is introduced and completed with a transition phase to reduce jerk. This works better in the completion phase.

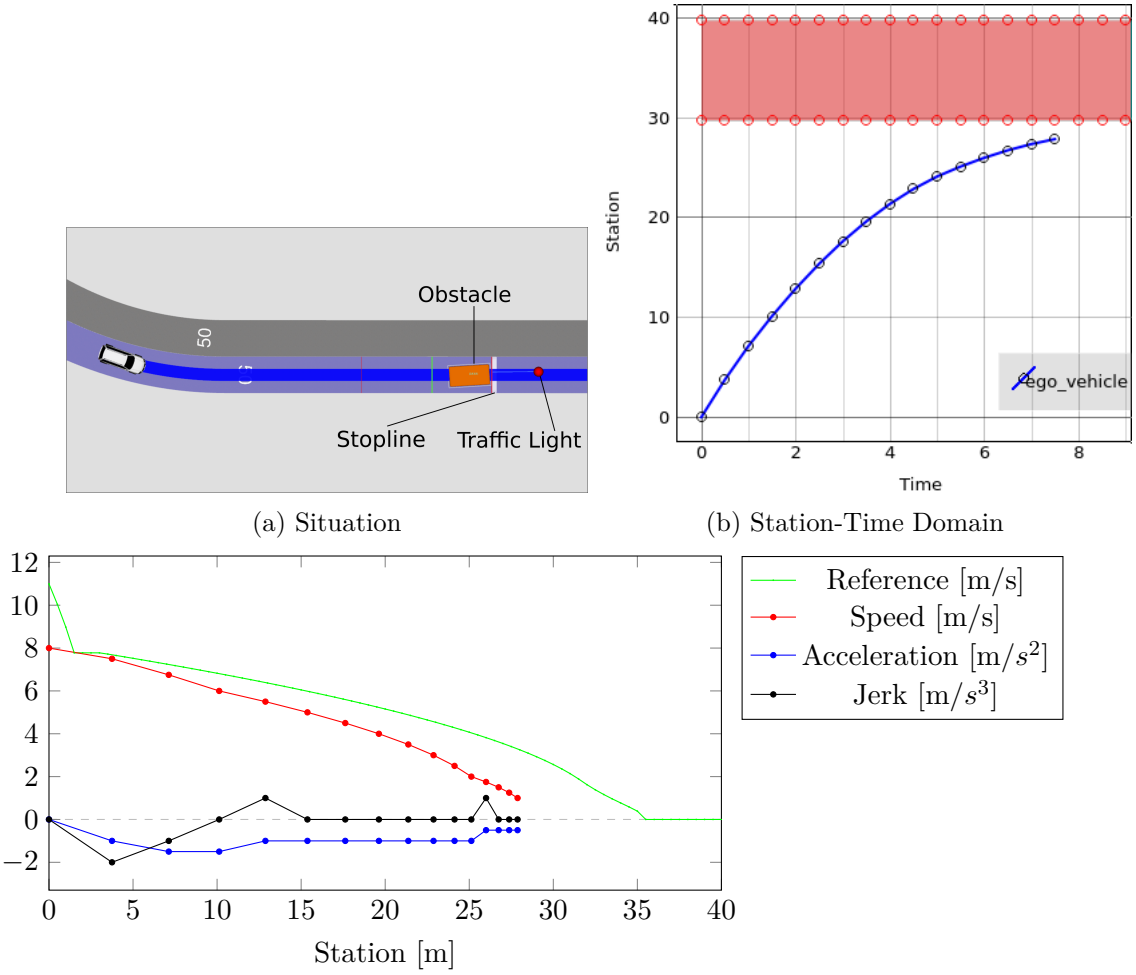
5.2 Experiment: Adapting to a Leading Obstacle

After evaluating the interaction with the reference speed profile, the next group of experiments aims to evaluate simple obstacle interaction. This includes obstacles that follow the path of the autonomous obstacle. An adaption to the obstacle's speed is required, yielding a conflict between reaching the reference speed profile and avoiding a collision.

5.2.1 Scenario: Stop at Traffic Light behind Obstacle

This scenario adds a standing obstacle waiting in front of a red traffic light. The autonomous vehicle approaches the traffic light with a constant initial speed of $8m/s$ and has to stop behind the obstacle. The obstacle is in a distance of $30m$ to the autonomous vehicle, the traffic light in a distance of $35.5m$. Therefore, the autonomous vehicle has to deviate from the reference speed profile to avoid a collision, since the reference speed profile isn't aware of obstacles. The scenario evaluates whether the planner is able to manage collision avoidance by neglecting the reference speed profile. The scene is visualized in Fig. 15a. The red ball is a symbol for the traffic light, the orange box represents the standing obstacle.

Figure 15b shows the station-time domain. The planning algorithm avoids a collision and lets the vehicle approach while ensuring safety distance. According to the planned speed profile illustrated in Fig. 15c, the vehicle won't perform a full stop within the given time horizon, but a slow speed of $1m/s$. The acceleration comfort area is exceeded to perform the braking maneuver. The intention of respecting the obstacle is clearly visible. The planner neglects the reference speed profile to prevent a collision.



(a) Situation

(b) Station-Time Domain

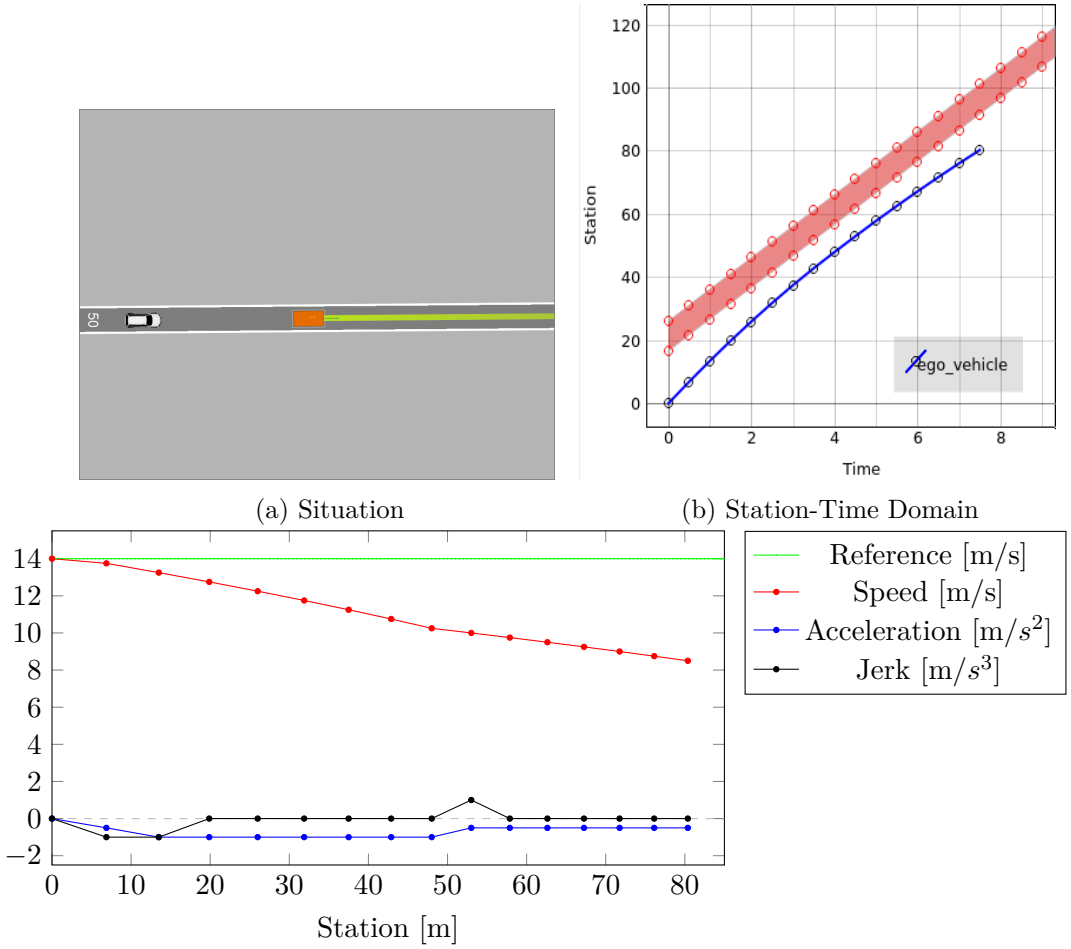
(c) Speed profile

Figure 15: Results of scenario 5.2.1. The planner neglects the reference speed profile to prevent a collision.

5.2.2 Scenario: Adjusting to a Slower Leading Obstacle

Following a leading obstacle is a common use case in urban environments. This scenario requires the autonomous vehicle to adjust its speed to a leading obstacle on a straight road. The autonomous vehicle travels initially with a constant velocity of $14m/s$. A leading obstacles drives with $10m/s$.

In this scenario, the planner avoids a collision, as visualized in Fig. 16b. The planner suggests to approach the obstacle until $t = 5s$ with a distance of $7.3m$. Afterwards, the distance is slightly increased to $8m$ at $t = 7.5s$ The planned speed profile is shown in



(a) Situation

(b) Station-Time Domain

(c) Speed profile

Figure 16: Results of scenario 5.2.2.

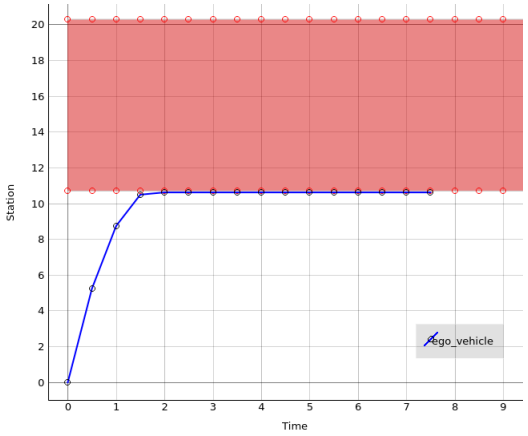
Fig. 16c.

The expectation for a human-like optimal trajectory requires to keep a constant speed, once a certain distance to the obstacle is established. Instead of keeping the speed of the obstacle of $10m/s$, the planner suggests to brake even further to increase the distance to the obstacle. Therefore, the planned speed profile doesn't match the human-like optimal one. Increasing the cost function weight γ_V to 1.275 would align the speed profile, but interferes the smoothness of scenario 5.1.2.

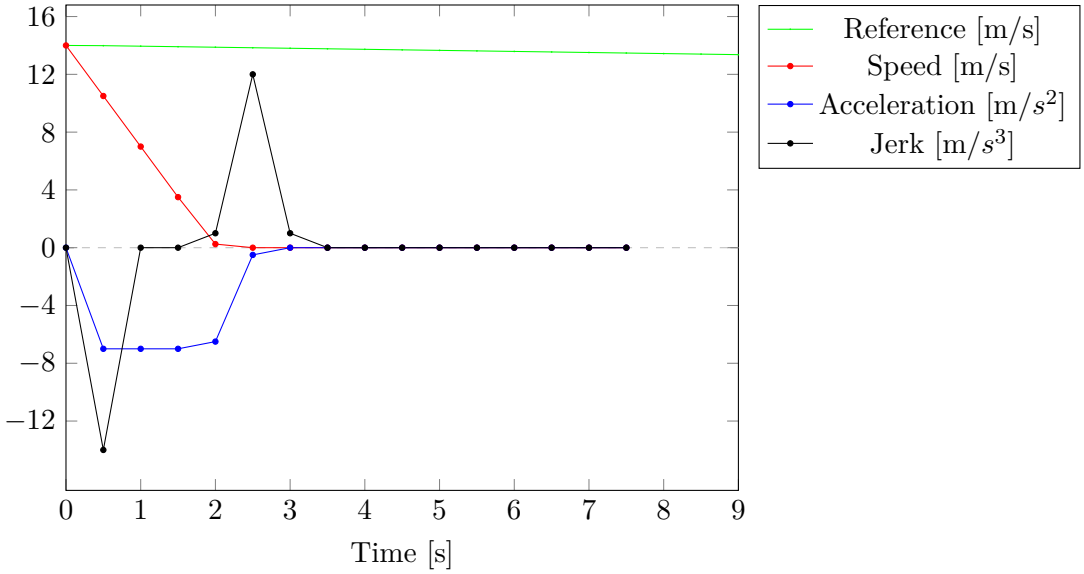
5.2.3 Scenario: Reacting to an Unexpected Obstacle

This scenario considers a misdetection of an obstacle. The road situation is equal to the last scenario: The autonomous vehicle is on a straight road with constant initial speed of $14ms$. However, this scenario adds a standing obstacle within $11m$ in front of the vehicle. This scenario can occur because of various reasons: Either something was detected as an obstacle (like grass from the road environment) or a real obstacle is in front of the autonomous vehicle and wasn't detected until now. Another reason for the emergence of this scenario is a sudden switch in the predicted motion of an obstacle. For example, a pedestrian crossing the street decides suddenly to stand still halfway, before continuing to cross the street (also known as *Oh I lost my phone!*). In both cases, reaction from the planner is required, an emergency brake is the only option to prevent a collision.

The planned solution is given in Fig. 17. The autonomous vehicle will get close to the safety distance of the obstacle, but avoids a collision as visualized in Fig. 17a. Slowing down by $a_{min} = -7m/s^2$ introduces a high amount of jerk (see Fig. 17b). Because the emergency maneuver followed by a full stop can be executed, the planner doesn't signal an unavoidable collision.



(a) Station-Time Domain

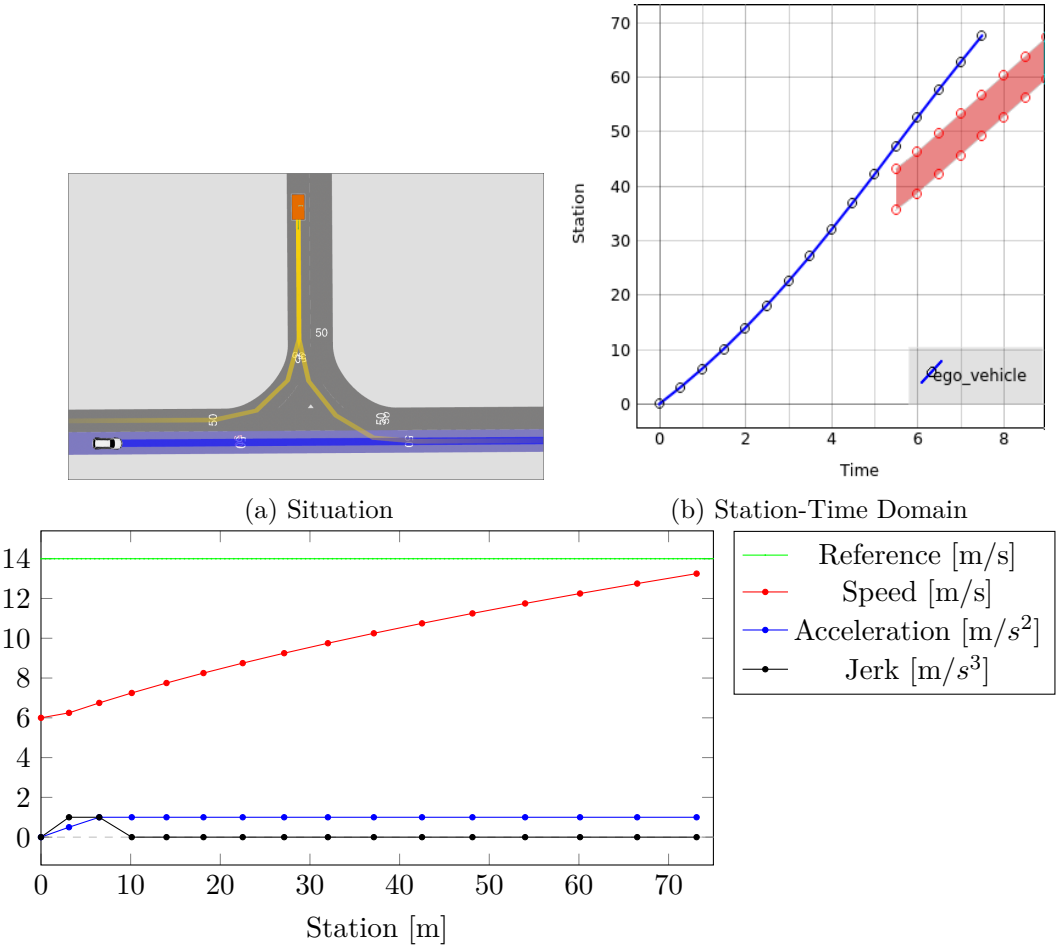


(b) Speed profile

Figure 17: Results of scenario 5.2.3.

5.3 Experiment: Merging Traffic

After handling interactions with obstacles that need an adaption to the given scenario, a more complex group of scenarios is inspected: handling merging traffic. Merging traffic includes obstacles with predicted trajectories which intend to enter or leave the path of the autonomous vehicle at some point in the future. In addition to avoid a collision, a decision has to be made, whether the autonomous vehicle should brake for the obstacle or speed up to get in front of it.



(a) Situation

(b) Station-Time Domain

(c) Speed profile

Figure 18: Results of scenario 5.3.1.

5.3.1 Scenario: Merging Obstacle

The first scenario of this experiment models an obstacle with the option to merge into the path of the autonomous vehicle at a speed of $7m/s$ (see Fig. 18a). The autonomous vehicle follows the blue path with an initial constant speed of $6m/s$. Traffic rules like *right of way* are not respected in this scenario. The obstacle either can merge or turn right without interfering the path (obstacle trajectories are marked in yellow). Both options of the obstacle are treated as most likely. Thus, the planner has to react to the merge option. The self-driving car can either keep the current speed to let the obstacle pass in front of it or speed up. The obstacle will merge into the path after $5.5s$ and $35m$ (see Fig. 18b).

The planner decides for the latter option. Figure 18c displays the speed profile. The acceleration is in the comfortable range, providing a smooth speed profile. In case that the obstacle would be $2.2m$ closer to the autonomous vehicle, the first option is preferred, since the required amount of acceleration reaches the uncomfortable value range.

5.3.2 Scenario: Crossing Pedestrian

Traffic participants can not only travel alongside the road, but also cross it. This scenario presents a pedestrian (turquoise box) crossing the street at a distance of $22m$ with a constant speed of $2m/s$. The predicted trajectory of the pedestrian is visualized in orange. The scene is visualized in Fig. 19a.

The pedestrian blocks station $22m$ to $28m$ from $t = 1.5s$ to $t = 2.5s$. The vehicle needs to slow down from its initial speed of $9m/s$ to prevent a collision with the pedestrian.

The result of the planner is shown in Fig. 19b and 19c. The planned trajectory leads to a braking maneuver to let the pedestrian pass. Thereby, it differentiates from the reference speed profile to avoid a collision. Afterwards, it starts to increase the speed to reach the reference speed profile. The planner provides a human-like behavior in this scenario.

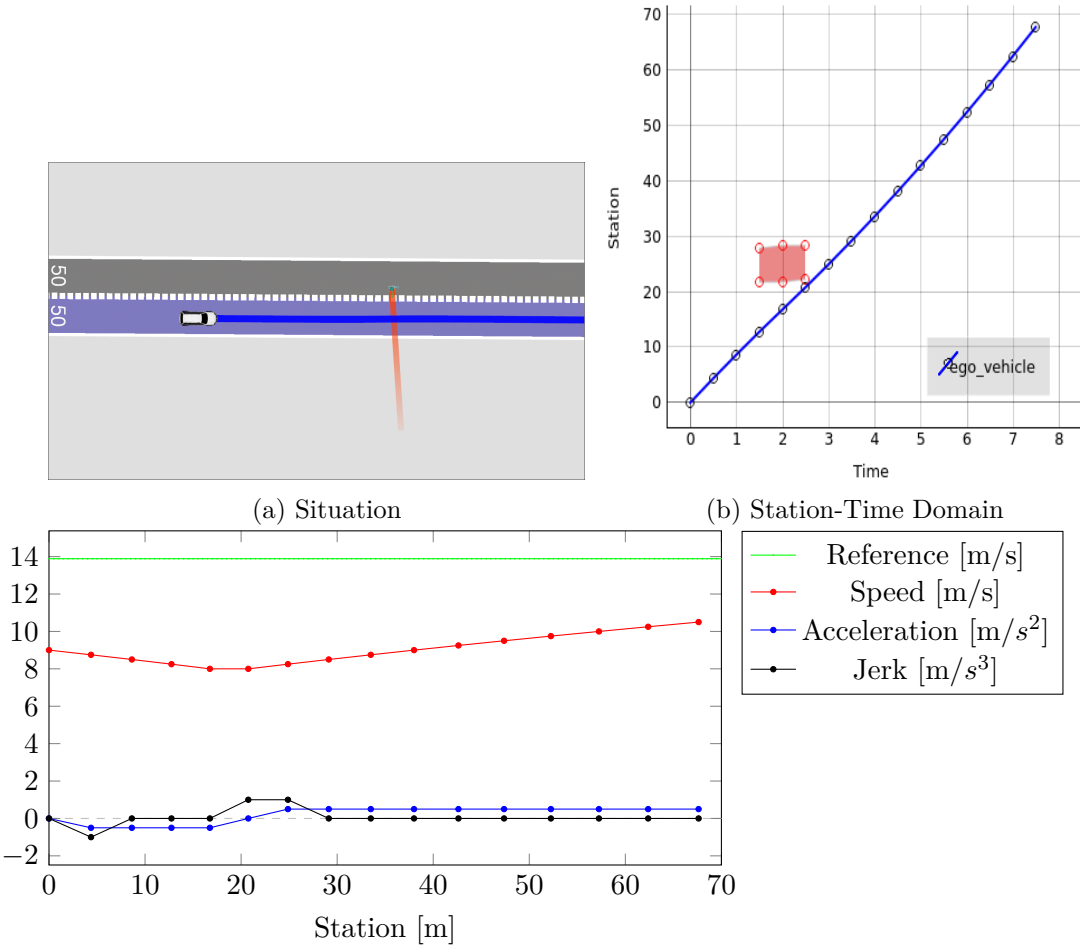
5.3.3 Scenario: Lane Change

Changing lanes is a complex driving maneuver. This scenario covers a lane change on a path from the right lane to the left lane in an urban environment. The scene is demonstrated in Fig. 20a. While the autonomous vehicle follows the path highlighted in blue, the obstacles keep their current lane. Obstacles are annotated with ids for distinction. A list of the speeds of the obstacles can be found in Tab. 3.

The autonomous vehicle is traveling with an initial constant speed of $8m/s$. Table 3 shows the constant speed of the given obstacles. To reach the gap of $16m$ between obstacle Id 0 and Id 1, the autonomous vehicle has to speed up to adapt to the obstacles.

Obstacle Id	0	1	2	3	4	5	6	7
Speed $[m/s]$	12	12	8	8	13	11.5	7	8.5

Table 3: Speeds of the obstacles from scenario 5.3.3.



(a) Situation

(b) Station-Time Domain

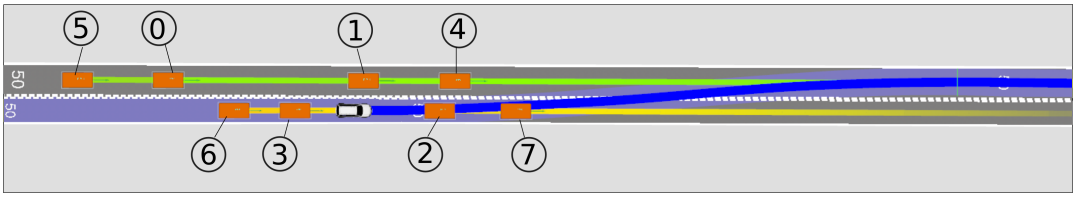
(c) Speed profile

Figure 19: Results of scenario 5.3.2.

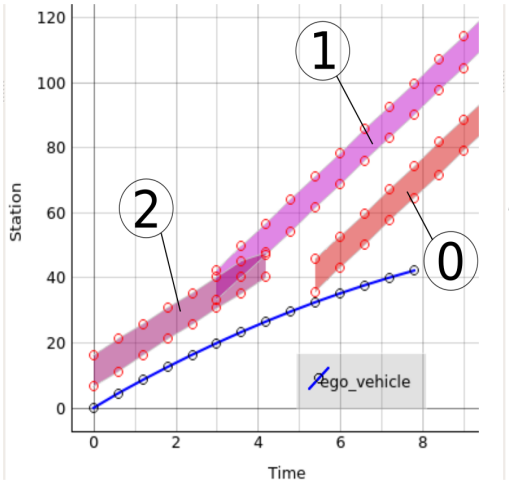
This scenarios is evaluated with obstacles 0 to 2, since it offers the option to take the gap between obstacle 0 and 1 or to brake and merge behind obstacle 0 into the left lane. The scenario is evaluated with different parameter sets from Tab. 2, to compare the quality of various sampling distances.

Parameter set Id A to E prefer to let both obstacles pass before entering the left lane. On the other hand, parameter Id F to I tend to take the gap. Parameter Id E and F are compared in the following, since they represent the changing point candidates of this phenomenon.

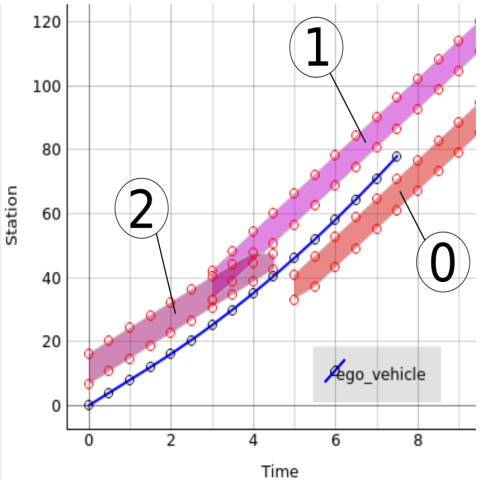
Figures 20b and 20e show the station-time plot of both parameter sets. Parameter set Id F takes the gap, getting close to obstacle 2 after 4.5s. The costs of both solutions are



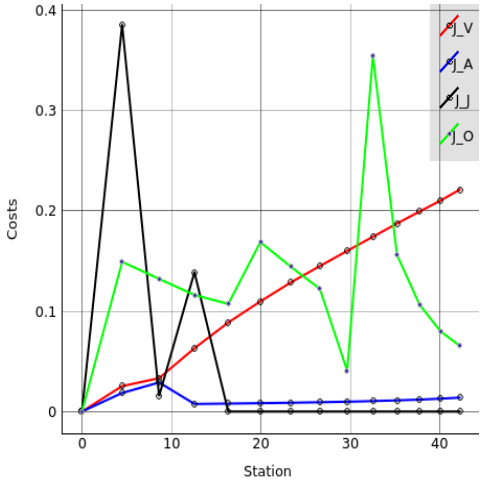
(a) Situation with annotated obstacle Ids.



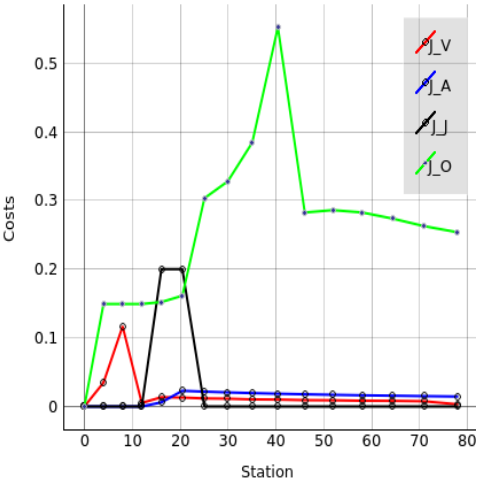
(b) Station-Time Domain of Param Id E



(c) Station-Time Domain of Param Id F



(d) Costs of Param Id E



(e) Costs of Param Id F

Figure 20: Obstacle interaction of scenario 5.3.3. Figure b and c show the station-time domain with three obstacles for two different sampling parameter sets. The costs of both solutions are visualized in Fig. d and e.

visualized in Figures 20d and 20e, showing the costs for each node of the shortest path. It is conspicuous that both solutions have a peak at the overlapping obstacles (station of

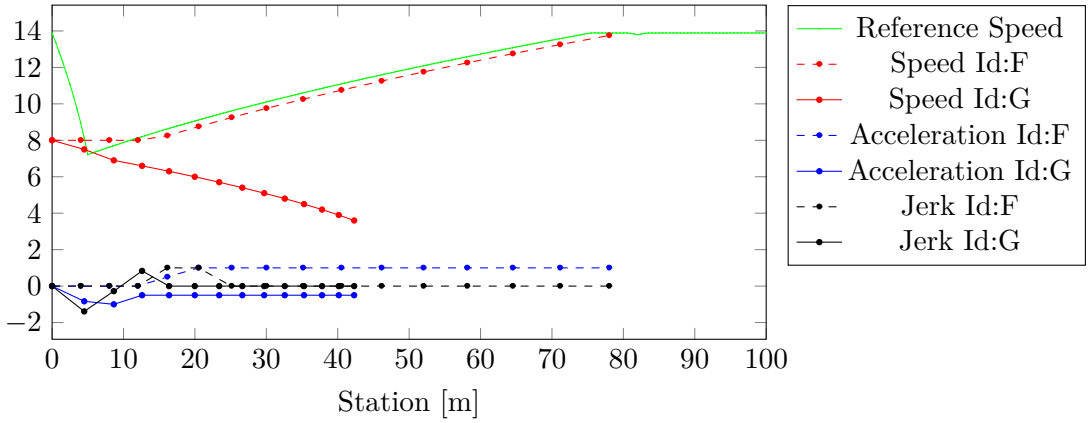


Figure 21: Speed profile of scenario 5.3.3 with parameter sets 6 and 7.

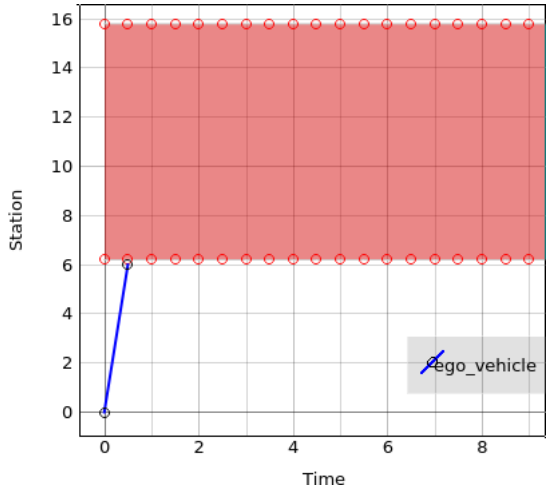
$30 - 40m$). This seems natural, due to the described obstacle interaction cost function. Since an overlap in the station-time domain leads to a collision of obstacles, it might be reasonable to avoid this position. However, this doesn't explain why the sampling distance has an impact on the decision of the planner.

The resulting trajectory is visualized in Fig. 21. The drop in reference speed is related to the curvature of the path.

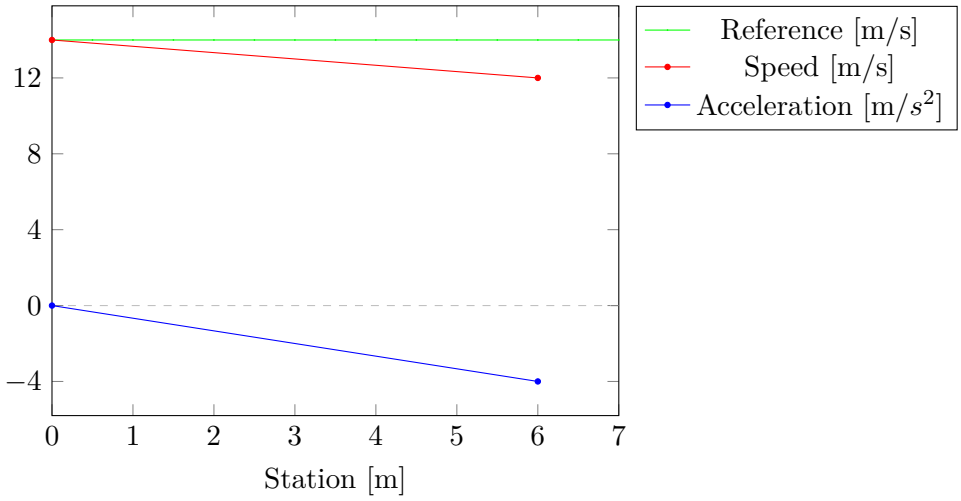
5.4 Experiment: Unavoidable Collision

A last scenario tests a situation with an unavoidable collision. This experiment is an extension of scenario 5.2.3: An obstacle appears on the path of the autonomous vehicle in a short distance, so that a collision is not avoidable. As visualized in Fig. 22a, the solution of the planner is a subsolution consisting of a single node. The subsolution consists of braking with $-4m/s^2$ (see Fig. 22b).

Because of the selection of a subsolution as optimal speed profile, the plan is detected correctly as an emergency situation. This requires human intervention, since a collision can't be avoided. The subsolution is a tendency in the right direction, but it isn't an emergency brake, which should use a maximum deceleration of $-7m/s^2$. The obstacle cost function J_O doesn't include the vehicle speed at the point of collision. Therefore, the braking maneuver with the lowest jerk without causing a collision is selected from the planner, which leads to a collision with higher speed than necessary.



(a) Station-Time Domain



(b) Speed profile

Figure 22: Results of scenario 5.4.

5.5 Run-time Analysis

The run-time performance measures the time needed to generate a speed profile, excluding the time spend on obstacle distance calculation. Thereby, the focus is on the run-time of the implemented planning algorithm. The measured run-time is based on a sequence of 100 planning cycles.

Run-Time of Experiments

Figure 23 shows the measured run-times of the presented scenarios with sampling parameter set F (see Tab. 1). The minimum, maximum and average run-time is measured. All scenarios have an average run-time value that is closer to the minimum than to the maximum. Scenarios of experiment 5.1 take the longest run-time, followed by scenario 5.3.1.

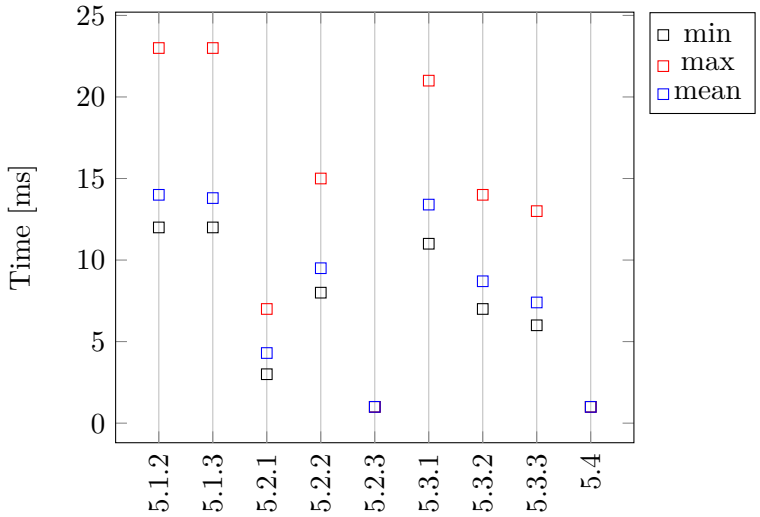


Figure 23: Run-times of the presented experiments as a result of 100 planning iterations.

The measurements indicate that obstacles contribute to a better run-time. Obstacles prune the station-time graph, since some nodes are blocked by obstacles and are not reachable anymore.

Varying Sampling Distances

In the following, two traffic scenarios are tested with different values of the sampling distance parameters τ and σ using the sets of Tab. 2. Varying the sampling distances can be used to find a set of parameters with online capability. The run-time is tested in the absence of obstacles with scenario 5.1.1, where a braking reaction is tested for a change in the reference speed profile. The test is repeated in the presence of obstacles in the lane change scenario 5.3.3. A different number of obstacles are added to the scene. Thereby, the scalability in terms of obstacle presence is evaluated.

The run-time performance for scenario 5.1.1 is displayed in Tab. 4. Having a look at the different parameter sets, it becomes clear that Id I is not suitable for achieving real-time, since it exceeds $100ms$ on average. The other sets are below the $100ms$ mark. The run-time correlates clearly with the number of cells in the matrix of the station-time graph.

Param Id	A	B	C	D	E	F	G	H	I
Runtime [ms]	1	1	2	4	9	14	33	89	324
# Matrix Cells	2,000	2,781	3,910	6,000	9,730	16,000	31,260	75,006	250,000

Table 4: Average runtime performance for **brake for speed limit** scenario. Param Id refers to Tab. 2.

Table 5 shows the execution time of the planner in the lane change scenario from section 5.3.3. The run-time is evaluated with a different number of obstacles (for example, the row with obstacle Id 3 contains obstacles 0, 1, 2 and 3).

Obstacle Id \ Param Id	A	B	C	D	E	F	G	H	I
without obstacles	1	1	2	4	8	13	30	83	304
0	1	1	2	4	8	13	29	83	289
1	1	1	2	4	7	12	27	75	173
2	1	1	1	3	5	7	16	45	155
3	1	1	1	1	2	3	6	16	56
4	1	1	1	1	2	3	7	18	68
5	1	1	1	1	2	3	7	18	69
6	1	1	1	1	2	3	7	20	72
7	1	1	1	1	2	3	7	19	70

Table 5: Average run-times of the planner in ms of scenario 5.3.3. Obstacle Id (see Fig. 20a) include the Ids of previous rows. Param Id refers to Tab. 2.

Comparing the performance to the row without obstacles, it becomes clear that the implemented algorithm benefits from obstacles. Interaction with more than three obstacles increases the run-time, but only slightly. The algorithm performs best with four obstacles, because they surround the path of the autonomous vehicle and apply the most pruning in the station-time graph. Adding more obstacle starts to increase the runtime again, since the station-time graph is already maximal pruned, but the obstacle handling requires some time.

Parameter set I is clearly not suitable for real-time in the absence of obstacles, but it

might be used with four or more obstacles. One might use parameter set H, but it is still close to the $100ms$ boundary. On the other hand, the parameter sets define a trade between accuracy and and run-time performance. Therefore, a suggestion is to use parameter set F or G, because they ensure online capability with a resolution of $0.5s$ (and $0.4s$) respectively, which can be sufficient for a rough speed profile.

For completeness, runtimes of the distance calculation for collision detection can be found in Tab. 6 in the appendix.

6 Conclusions & Future Work

A graph-based speed planning algorithm was implemented in this thesis. The performance of the algorithm was evaluated in various urban scenarios. The effect of *post-poining the evil* was identified and a solution to weaken the effect was proposed.

The algorithm showed promising results in the evaluation. Most speed profiles met the requirements stated by the cost functions. Experiment 5.1 demonstrated that the planner is able to create comfortable speed profiles with foresight to the reference speed profile. Aside from collision avoidance, the speed profiles were jerk-reduced and therefore comfortable for the passengers. The planner was capable of collision avoidance in every scenario of experiment 5.3 and 5.2. A comprehensible decision for deceleration or picking up speed was made, taking the reference speed profile and other traffic participants into account. Especially scenario 5.3.2 and 5.3.1 demonstrated a speed profile that is similar to human-behavior. The algorithm performed collision free with all obstacles in the complex maneuver of a lane change, indicating a good scaling behavior with even more obstacles. A critical situation of an unexpectedly appearing obstacle was successfully resolved in scenario 5.2.3 by planning a braking maneuver. Furthermore, in experiment 5.4 where a collision is intentionally caused, the planner successfully detects the unavoidable collision with the given path so that a further strategy can be triggered.

The scalability and run-time performance of the algorithm was evaluated with multiple obstacles in scenario 5.5. The implemented approach was able to finalize a single planning cycle in less than $24ms$ in all scenarios. As a general rule it is sufficient to finish a planning cylcle within $100ms$ [36]. Hence, it can be concluded that the planner is capable to plan in real-time.

On the other hand, scenarios 5.2.2 and 5.3.3 showed that the selection of sampling parameters and cost function parameters is crucial to match a human-like speed profile. It was observed that under some conditions the solution found by the planner is sensible to the sampling parameters. A solution can present different driving maneuvers based on the sampling parameters. The abstraction from the environment situation into a reference speed profile comes also with costs: a set of parameters must be found, which can handle all situations. It is easy to overtune the algorithm for a specific scenario, neglecting the idea of a scenario-independent approach. Changing parameters is an exhausting process that needs to take place with representative scenarios. Another conclusion is that the cost functions formulate a vague problem, so that the reasons for

the optimality of a solution can be hard to comprehend. The cost function should be more specific about what is actually preferred.

All in all, the implemented planner was able meet the requirements in the context of safe and comfortable driving. As always, there is room for improvements. The following points present suggestions for future work based on this thesis:

Re-Thinking the Safety Distance Metric

Experiment 5.4 showed a problem of the obstacle cost function: Since the costs are based on the distance to the obstacle, all collisions are treated equally. This is probably wrong: A collision with higher speed might cause more damage than one with lower speed. Thus, an option for improvement is the integration of a metric that measures the damage of a collision. This metric can then be used in the cost function in case of an unavoidable collision to create a less fatal collision.

In addition, using a fixed distance as metric doesn't scale with different speeds. The safety distance to a leading vehicle should be extended with higher speed values, e.g. on highways. Including the speed of the autonomous vehicle in the obstacle distance calculation also helps with the evaluation of collisions.

Adding Uncertainty

The implemented approach uses a single obstacle trajectory for prediction. In real traffic, obstacles will most likely not follow the prediction for the entire time. Therefore, it may be an advantage to take not only a single prediction but a probability distribution into account. An example for probabilistic planning can be found in [34]. The authors propose a planner using a Gaussian Mixture Regression algorithm to estimate the probability density of obstacle positions, aiming to optimize the probability of collision free driving. This thesis algorithm should be capable of handling uncertainty, given an adjustment to the obstacle interaction cost function.

Within the process of adding uncertainty, a next step in the direction of increased safety is the constant planning of an emergency trajectory. The emergency trajectory is activated, when an obstacle behaves in an unexpected way. Pek and Althoff [37] designed a planner that plans such an emergency trajectory in addition to the optimal trajectory. The same

could be applied in speed planning: a speed profile that expects that an obstacle follows the prediction and a more defensive profile in case of unexpected obstacle movement.

Stability of Solutions

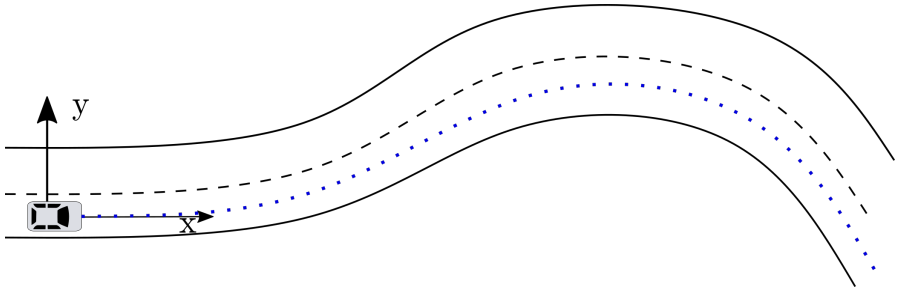
Another interesting improvement is the supplement of another cost function to stabilize the speed profiles over time. Based on the cost functions, the planner might switch between solutions with similar costs, causing discomfort for the passengers. A cost term could be added that compares the newly planned solutions to the solution from the last planning cycle. The last solution has to be shifted and clamped by the elapsed time since the last planning cycle. One option is proposed by Lim et al. [23], who add a consistency cost term that compares the speed at each time step to the speed of the last solution. The consistency term could also be used in the selection of the best solution to save some run-time.

Sampling in Speed and Time

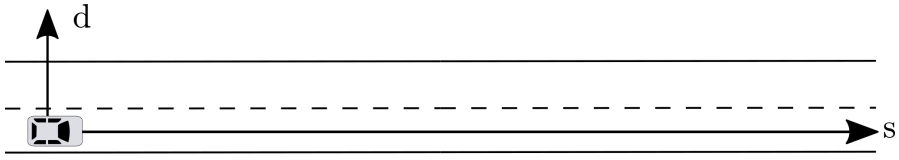
The used sampling in station and time formed the station-time graph. One idea to improve the run-time performance is to replace station by speed. In most cases, the planner needs to reach or is limited by a certain speed or acceleration value, as opposed to station values. Therefore, the planning could be made in the speed-time domain. This means that a non-uniform sampling could be applied, e.g. more dense in slow speed ranges. The crux for speed-time planning is to find a suitable sampling distance for accurate obstacle interaction.

Frenét-Frame

Another improvement in the context of obstacle collision test and planning is the use of a Frenét-Frame. The Frenét-Frame transforms a road in a curvilinear manner, using a transformation from cartesian coordinates (x, y) to coordinates (s, d) with s as station of the path of the autonomous vehicle and d as lateral deviation from the path. An example of a simple traffic scenario is given in Figure 24 to demonstrate the difference between the Cartesian coordinate system and the Frenét-Frame. Werling et al. [36] proposed a planner making use of the Frenét-Frame.



(a) Scenario in Cartesian coordinate frame



(b) Representation of Fig. a in the Frenét-Frame.

Figure 24: Example of the Frenét-Frame

List of Figures

List of Tables

References

[1] Claes Tingvall and Narelle Haworth. "Vision Zero: an ethical approach to safety and mobility". In: *6th ITE international conference road safety & traffic enforcement: beyond*. Vol. 1999. 2000, pp. 6–7.

[2] Miltos Kyriakidis et al. "The deployment of advanced driver assistance systems in Europe". In: *Available at SSRN 2559034* (2015).

[3] *Road traffic injuries*. `https://www.who.int/news-room/fact-sheets/detail/road-traffic-injuries`. Accessed: 2010-06-28.

[4] *Road safety: Data show improvements in 2018 but further concrete and swift actions are needed*. `http://europa.eu/rapid/press-release_IP-19-1951_de.htm`. Accessed: 2010-06-28.

[5] G. Becher et al. "Automatisiert, Vernetzt, Elektrisch. Potenziale innovativer Mobilitätslösungen für Baden-Württemberg". In: *e-mobil BW GmbH* (2015).

[6] World Health Organization. *World report on road traffic injury prevention*. `https://apps.who.int/iris/bitstream/handle/10665/42871/9241562609.pdf;jsessionid=A7881FF073D9924ACC1EFA5BDC7CAE0D?sequence=1`. Accessed: 2019-06-28. 2004.

[7] *EU-Kommission: Bis 2050 keine Verkehrstoten mehr – Fahrerassistenzsysteme sollen helfen*. `https://www.dvr.de/presse/informationen/eu-kommission-bis-2050-keine-verkehrstoten-mehr--fahrerassistenzsysteme-sollen-helfen_id-4741.html`. Accessed: 2010-06-28.

[8] Santokh Singh. *Critical reasons for crashes investigated in the national motor vehicle crash causation survey*. Tech. rep. National Center for Statistics and Analysis, 2015.

[9] Haoyang Fan et al. "Baidu apollo em motion planner". In: *arXiv preprint arXiv:1807.08048* (2018).

[10] Kamal Kant and Steven W Zucker. "Toward efficient trajectory planning: The path-velocity decomposition". In: *The international journal of robotics research* 5.3 (1986), pp. 72–89.

[11] Brian Paden et al. "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.

[12] Zahra Boroujeni et al. “Flexible unit A-star trajectory planning for autonomous vehicles on structured road maps”. In: *2017 IEEE International Conference on Vehicular Electronics and Safety (ICVES)*. IEEE. 2017, pp. 7–12.

[13] Julius Ziegler and Christoph Stiller. “Spatiotemporal state lattices for fast trajectory planning in dynamic on-road driving scenarios”. In: *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE. 2009, pp. 1879–1884.

[14] Matthew McNaughton et al. “Motion planning for autonomous driving with a conformal spatiotemporal lattice”. In: *2011 IEEE International Conference on Robotics and Automation*. IEEE. 2011, pp. 4889–4895.

[15] Tianyu Gu and John M Dolan. “On-road motion planning for autonomous vehicles”. In: *International Conference on Intelligent Robotics and Applications*. Springer. 2012, pp. 588–597.

[16] Philipp Bender et al. “The combinatorial aspect of motion planning: Maneuver variants in structured environments”. In: *2015 IEEE Intelligent Vehicles Symposium (IV)*. IEEE. 2015, pp. 1386–1392.

[17] Chris Urmson et al. “Autonomous driving in urban environments: Boss and the urban challenge”. In: *Journal of Field Robotics* 25.8 (2008), pp. 425–466.

[18] Tianyu Gu, John M Dolan, and Jin-Woo Lee. “On-road trajectory planning for general autonomous driving with enhanced tunability”. In: *Intelligent Autonomous Systems 13*. Springer, 2016, pp. 247–261.

[19] Laurène Claussmann et al. “A Review of Motion Planning for Highway Autonomous Driving”. In: *IEEE Transactions on Intelligent Transportation Systems* (2019).

[20] Michael Montemerlo et al. “Junior: The stanford entry in the urban challenge”. In: *Journal of field Robotics* 25.9 (2008), pp. 569–597.

[21] Jeff Johnson and Kris Hauser. “Optimal longitudinal control planning with moving obstacles”. In: *2013 IEEE Intelligent Vehicles Symposium (IV)*. IEEE. 2013, pp. 605–611.

[22] Jeff Johnson and Kris Hauser. “Optimal acceleration-bounded trajectory planning in dynamic environments along a specified path”. In: *2012 IEEE International Conference on Robotics and Automation*. IEEE. 2012, pp. 2035–2041.

[23] Wonteak Lim et al. “Hierarchical trajectory planning of an autonomous car based on the integration of a sampling and an optimization method”. In: *IEEE Transactions on Intelligent Transportation Systems* 19.2 (2018), pp. 613–626.

[24] Changliu Liu, Wei Zhan, and Masayoshi Tomizuka. "Speed profile planning in dynamic environments via temporal optimization". In: *2017 IEEE Intelligent Vehicles Symposium (IV)*. IEEE. 2017, pp. 154–159.

[25] Jordi Pérez Talamino and Alberto Sanfeliu. "Anticipatory kinodynamic motion planner for computing the best path and velocity trajectory in autonomous driving". In: *Robotics and Autonomous Systems* 114 (2019), pp. 93–105.

[26] Tianyu Gu et al. "Focused trajectory planning for autonomous on-road driving". In: *2013 IEEE Intelligent Vehicles Symposium (IV)*. IEEE. 2013, pp. 547–552.

[27] Thomas H Cormen et al. *Introduction to algorithms*. MIT press, 2009.

[28] Wenda Xu et al. "A real-time motion planner with trajectory optimization for autonomous vehicles". In: *2012 IEEE International Conference on Robotics and Automation*. IEEE. 2012, pp. 2061–2067.

[29] Silvia Magdici and Matthias Althoff. "Fail-safe motion planning of autonomous vehicles". In: *2016 IEEE 19th International Conference on Intelligent Transportation Systems (ITSC)*. IEEE. 2016, pp. 452–458.

[30] Steven M LaValle. *Planning algorithms*. Cambridge university press, 2006.

[31] Shai Shalev-Shwartz, Shaked Shammah, and Amnon Shashua. "Safe, multi-agent, reinforcement learning for autonomous driving". In: *arXiv preprint arXiv:1610.03295* (2016).

[32] David Eager, Ann-Marie Pendrill, and Nina Reistad. "Beyond velocity and acceleration: jerk, snap and higher derivatives". In: *European Journal of Physics* 37.6 (2016), p. 065008.

[33] Y. Zhang et al. "Hybrid Trajectory Planning for Autonomous Driving in Highly Constrained Environments". In: *IEEE Access* 6 (2018), pp. 32800–32819. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2018.2845448.

[34] J. Schlechtriemen, K. P. Wabersich, and K. Kuhnert. "Wiggling through complex traffic: Planning trajectories constrained by predictions". In: *2016 IEEE Intelligent Vehicles Symposium (IV)*. 2016, pp. 1293–1300. DOI: 10.1109/IVS.2016.7535557.

[35] Matthias Althoff, Markus Koschi, and Stefanie Manzinger. "CommonRoad: Composable benchmarks for motion planning on roads". In: *2017 IEEE Intelligent Vehicles Symposium (IV)*. IEEE. 2017, pp. 719–726.

[36] Moritz Werling et al. “Optimal trajectory generation for dynamic street scenarios in a frenet frame”. In: *2010 IEEE International Conference on Robotics and Automation*. IEEE. 2010, pp. 987–993.

[37] Christian Pek and Matthias Althoff. “Computationally efficient fail-safe trajectory planning for self-driving vehicles using convex optimization”. In: *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*. IEEE. 2018, pp. 1447–1454.

Appendix

Table 6: Runtimes in *ms* for collision test of scenario 5.3.3. Obstacle Id rows include the Ids of previous rows (see Fig. 20a for an Id mapping).

Obstacle Id \ Config Id	1	2	3	4	5	6	7	8	9
0	1	1	1	1	2	2	3	4	7
1	2	3	3	4	4	5	7	10	16
2	4	5	5	6	7	9	12	15	24
3	7	8	9	11	12	15	19	26	39
4	10	12	13	15	17	21	25	36	51
5	10	13	15	17	19	24	31	39	57
6	13	17	20	21	25	41	40	50	74
7	14	17	19	22	24	29	38	51	74