Accelerated Dynamic Programming for Trajectory Planning of Automated Vehicles

Jona Ruof* and Klaus Dietmayer

Abstract: For the real-world deployment of automated vehicles general trajectory planning methods are required. The most versatile planning approaches, such as dynamic programming, consider many distinct options, which increases their computational effort significantly. Therefore, previous works often use heuristic search or significantly limit the amount of behavior options. However, due to recent advances in graphic processing units (GPUs) the available computational resources have increased tremendously. Thus, this paper reevaluates the use of the versatile dynamic programming method under consideration of recent hardware. Additionally, an exemplary implementation and evaluation on challenging scenarios such as unsignalized intersections or unprotected turns is provided. The source code will be released as part of our trajectory planning library under `https://github.com/uulm-mrm/tpl`.

Keywords: Automated Driving, Trajectory Planning, Dynamic Programming

1 Introduction

Since the beginning of the automated driving development planning approaches were required, which could efficiently evaluate many distinct trajectory options in a large search space. A particular category are graph-based planning methods, which utilize a structured graph of trajectory segments. In these methods the action and state domain is necessarily discretized, and the naive approach then evaluates all possible actions in each time step. However, this increases the computational effort exponentially, where after only a limited number of time steps prohibitively many actions and states would have to be evaluated. Thus, an improved construction of the graph or an improved search strategy is required.

The A*-algorithm [1] improves the search, by not evaluating every possibility, but relying on a heuristic function to guide the exploration. However, depending on the use-case, a good heuristic can be complicated to construct. Lattice-based planning methods therefore utilize exhaustive search, i.e. dynamic programming (DP), and improve the construction of the graph instead. This is often realized by carefully selecting the possible actions, such that only a limited, closed set of states can be visited. While this reduces the computational burden, the limited resolution of the graph can become a hindrance, if states need to be reached, which are not close to any graph node.

A naive solution is to instead construct the graph in such a way that a more dense sampling of the state space is achieved. This in turn increases again the computational cost, but not to an exponential amount, since the set of states in the lattice is still limited.

*All authors are with the Institute of Measurement-, Control-, and Microtechnology, Ulm University, Albert-Einstein-Allee 41, 89081 Ulm, Germany. E-Mail: {firstname}.{lastname}@uni-ulm.de

Furthermore, a possible remedy for the increased computational effort is readily provided by graphic processing units (GPUs). Earlier lattice-planning approaches from the last decade, already explored the potential of parallelization [2], [3]. But while the capabilities of GPUs have rapidly increased since then, the development of parallel, lattice-based planning methods has seemingly stalled. We therefore investigate these dynamic programming approaches again under consideration of the improved capabilities of currently available hardware.

The remainder of this paper is structured as follows: After summarizing related literature in section 2 we will briefly outline necessary fundamentals in section 3, investigate the requirements and feasibility of parallel planning approaches in section 4, and outline an exemplary, real-time capable planning method section 5. Finally, we will evaluate the obtained algorithm on driving problems, like unprotected turns, roundabouts and general intersections scenarios in section 6.

2 Related Work

In the literature, dynamic programming and related approaches have already been explored in some variants and applications. In [2] a path lattice is first constructed by computing pose transitions in a Frenet frame via a shooting method with point mass dynamics. By assigning velocity profiles with constant acceleration to the paths a spatio-temporal lattice is obtained, pruned, and searched exhaustively for a cost-minimizing trajectory. The evaluation on highway scenarios (e.g. overtaking) showed the effectiveness of the approach. Additionally, a GPU was used to accelerate the search, which lead to considerable speedups on hardware available at that time (2011). A weakness of [2] is the greedy pruning strategy, which may prematurely remove viable trajectory candidates. In contrast, the approach presented in 5 does not require a pruning step.

A later approach by Heinrich et al. [3] follows a strategy very similar to [2]. First, paths are generated by sampling in a Frenet frame and then velocity profiles are planned along feasible paths using a dynamic program. In contrast to [2] fifth-order polynomials are used as longitudinal transitions to increase the degree of continuity of the trajectory.

Where [2], [3] build the lattice during execution, in [4] the state lattice is first constructed offline. Hermite interpolation of quintic polynomials is applied to generate motion primitives between different states of a Frenet frame. After pruning infeasible transitions, the lattice is exhaustively searched to obtain the trajectory. Since the used lattice is even sparser compared to [2] no GPU acceleration was investigated. On a CPU the search problem could already be solved in $\approx 20\,\mathrm{ms}$. As limiting factors of the approach the offline construction and relative sparsity of the lattice may be noted.

For off-road planning, recent work by Botros et al. [5] also utilizes lattice planning methods. In this work, special consideration was put into the geometric continuity of the lattice, where G3 curves were used as transitions. By application of an A^* search a viable path was then obtained and simplified. Finally, a velocity profile was planned along this path, which constitutes a path velocity decomposition approach.

Another work by Meng et al. [6] uses lattice planning in combination with numerical smoothing. Initially a comparatively sparse lattice is used to plan a rough motion profile with a low degree of continuity. Then the motion profile is refined with quadratic programming under consideration of the dynamics and additional constraints. In [6] this

process is done in a laterally and longitudinally decoupled fashion, which can lead to suboptimal results in some cases.

A very similar and earlier work by Fan et al. [7] instead utilizes a procedure similar to the expectation maximization algorithm: Lateral and longitudinal components are optimized separately, but repeatedly for multiple iterations. Each optimization step consists of a dynamic program, which constructs a rough initial motion, which is then numerically smoothed. While both [6] and [7] use exhaustive search for the dynamic programming steps, the states spaces are intentionally kept small by decoupling and exclusion of higher order derivatives. Thus, GPU acceleration was not required nor investigated.

Another creative approach was examined in a recent publication by Sormoli et al. [8] and earlier by Sulkowski et al. [9]. In these publications the driving problem is modeled via fluid dynamics, where the Lattice Boltzmann Method (LBM) is used for flow propagation. Since the LB-Method is computationally expensive, [8] outlines how parallelization can be achieved via GPU hardware. While the approach is conceptually distinct from a dynamic program, the final algorithm and implementation is surprisingly similar to our approach and is thus referenced here for completeness.

3 Fundamentals

The fundamental principle of dynamic programming is captured by the Bellman equation [10], which we repeat here briefly: Consider states $\boldsymbol{x}_t \in \mathcal{X}$ from a discrete state space $\mathcal{X}$ and actions $\boldsymbol{u}_t \in \mathcal{U}$ from a likewise discrete action space $\mathcal{U}$ at discrete time points $t \in \{0, ..., T\}$ up to a planning horizon T. States in different time steps are related with a dynamics function $\boldsymbol{x}_{t+1} = \boldsymbol{f}(\boldsymbol{x}_t, \boldsymbol{u}_t)$ and rated by a cost function $c(\boldsymbol{x}_{t+1}, \boldsymbol{u}_{t+1}, t) \in \mathbb{R}_0^+$. Further, the value function $v(\boldsymbol{x}_t)$ relates costs from future time steps with the current time step by the recursive definition

$$v(\boldsymbol{x}_t) = \min_{\boldsymbol{u}_t} \left[c(\boldsymbol{x}_t, \boldsymbol{u}_t, t) + v(\boldsymbol{f}(\boldsymbol{x}_t, \boldsymbol{u}_t)) \right] , \tag{1}$$

where we select terminal costs $c(\boldsymbol{x}_T, \boldsymbol{u}_T, T) = v(\boldsymbol{x}_T)$ as boundary condition of the recursion. Since $\boldsymbol{x}_t, \boldsymbol{u}_t$ are discrete, it is possible to computationally evaluate the value function by iterating over all possible actions in each state. Note that the formulation presented above is already a specialized case, where under the assumption of causality, states at t can only be visited from states in the time step $t - 1$. This means that, topologically, states are organized in a directed acyclic graph (DAG), which makes it possible to compute the value function in a single pass of backward induction [10]. For a specific starting state $\boldsymbol{x}_0$ one can then evaluate Equation 1 forward in time to obtain an ideal sequence of actions, where the respective states are obtained by application of the dynamics function. As many works have demonstrated that the exact computation of the value function for all possible states is computationally expensive, $v(\boldsymbol{x}_t)$ is often replaced by an estimate $\tilde{\boldsymbol{v}}(\boldsymbol{x}_t)$ in the minimization. Thus, we also apply an (arguably minor) approximation here, which will be outlined in the following sections.

4 Requirements and Computational Feasibility

First, this section analyzes the necessary state-space requirements for trajectory planning in automated driving. The computational complexity arising for a dynamic programming algorithm using the required state-space is examined subsequently.

4.1 Requirements

Since on-road driving is considered in this work, the state-space is naturally aligned along the targeted road and given in Frenet coordinates. Practically, this means that the state-space must contain at least the longitudinal position on the path, the lateral offset to the left and right, and the velocity along the road. For each of these quantities different requirements need to be considered.

For the longitudinal dimension it must be ensured that for higher velocities the state space covers a sufficient distance ahead of the vehicle. As, realistically, the perception range of many vehicles is rather limited, extending the horizon further would result in diminishing returns, as the vehicle may not be able to react anyway. Therefore, covering the longitudinal dimension up to a fixed perception range from 0 m to 200 m is considered necessary. Note that, this limit may be dynamically reduced further in regions with a lower maximum velocity. Even though, the longitudinal dimension thus covers a comparatively large distance, its discretization has more lenient requirements. The selected discretization step effectively decides, which intermediate discrete values are included in the range 0 m $-$ 200 m. This becomes most relevant when stopping, as the vehicle can only assume the discretized longitudinal states. However, since stopping early before an obstacle is less problematic, the spatial steps can be as high as multiple meters in practice.

For the lateral dimension across the road, its precise size may be dependent on the road shape itself or rather on the availability of lanes. If there are no further lanes, there is no need to extend the state space beyond the current lane. Therefore, the critically necessary lateral space varies. For simplicity, ± 5 m in the left and right direction are considered necessary in this analysis, which includes one additional left and right lane. For the lateral discretization, much finer steps are required compared to the longitudinal dimensions. In practice, this becomes especially relevant when planning through gaps.

In the velocity dimension, the vehicle needs to be able to assume a wide range of speeds and target certain specific speed limits. In accordance with UN regulation No. 157 [11] autonomous vehicles are allowed operation up to a maximum speed limit of 130 kph or 36 m/s. Since stopping must always be an option, the velocity state dimension must naturally cover the entire velocity range from 0 m/s to 36 m/s. Additionally, arbitrary speed limits may be imposed, where the strictest tolerance for speed limit violation is usually about 5 kph or 1.3 m/s. Therefore, the discretization must be chosen fine enough, such the speed limits can be respected within the legal tolerances.

While the longitudinal, lateral and velocity dimension cover the primary states of driving along a road, changes in these quantities cannot happen instantaneously. Instead, the motions are dependent on previous states and their respective derivatives. To prevent dynamically infeasible, discontinuous state transitions, higher order derivatives must also be considered in the state space. Mathematically the continuity of a trajectory can be defined through membership of continuity classes C^n. A function $g \in C^n$ is at least n-

times continuously differentiable. This means that for a trajectory $\boldsymbol{\tau}$, where the action is a function of a derivative quantity of the state $\boldsymbol{u}_t = g(\boldsymbol{x}_t^{(n)})$, the trajectory is at most in C^{n-2}. To capture realistic vehicle dynamics both the lateral and longitudinal dimension needs to be at least in C^2, as accelerations would be discontinuous otherwise. Practically, this means that the state space needs to include additionally the lateral velocity, lateral acceleration and longitudinal acceleration. In summary, a six-dimensional state space is thus required to capture the dynamics accurately.

4.2 Runtime Complexity and Feasibility

In the previous section, the necessary dimensions of the state space and the respective required extends are established. Naively, this results in a six-dimensional state space $\mathcal{X} = \mathcal{S} \times \dot{\mathcal{S}} \times \ddot{\mathcal{S}} \times \mathcal{L} \times \dot{\mathcal{L}} \times \ddot{\mathcal{L}}$. Thereby, $\mathcal{S}, \mathcal{L}$ are the set of possible longitudinal and lateral states, and $\dot{\mathcal{S}}, \ddot{\mathcal{S}}, \dot{\mathcal{L}}, \ddot{\mathcal{L}}$ the sets of their respective derivatives. Likewise, $\mathcal{U} = \dddot{\mathcal{S}} \times \dddot{\mathcal{L}}$ is the action set composed of the action spaces $\dddot{\mathcal{S}}$, $\dddot{\mathcal{L}}$ of the next higher-order derivatives. By examining the definition of Equation 1 it becomes apparent, that the computational complexity of the dynamic programming algorithm is given by

$$O(T \cdot N) = O(T \cdot |\mathcal{X}| \cdot |\mathcal{U}|) = O(T \cdot |\mathcal{S}| \cdot |\dot{\mathcal{S}}| \cdot |\ddot{\mathcal{S}}| \cdot |\mathcal{L}| \cdot |\dot{\mathcal{L}}| \cdot |\ddot{\mathcal{L}}| \cdot |\dddot{\mathcal{S}}| \cdot |\dddot{\mathcal{L}}|)\,. \tag{2}$$

This means that the required value-function evaluations scale linearly with the size of the planning horizon and the size of the state and action space. Also, the size of each of the state or action dimensions contributes multiplicatively. Given the requirements from the previous subsection 4.1, a numeric estimate for N in a possible real world application can be obtained. Since the longitudinal dimension S can be discretized rather coarsely, we estimate a regular discretization of 2 m, which yields $|\mathcal{S}| = 101$ (including 0) for the previously established 200 m perception range. For the lateral dimension a finer regular discretization of 0.5 m is assumed, yielding $|\mathcal{L}| = 21$. For the longitudinal velocity the discretization step is bound by the minimal legal speed limit tolerance of 1.3 m/s. If arbitrary speed limits are assumed, this necessitates a finer discretization of 1 m/s resulting in $|\mathcal{S}| = 37$ for the entire admissible velocity range. Further, optimistic state dimension sizes of $|\dot{L}| = 11, |\ddot{L}| = 7, |\ddot{S}| = 7$ and action dimension sizes $|\dddot{L}| = 7, |\dddot{S}| = 7$ are assumed. In summary, this yields $\approx 2.07 \cdot 10^9$ value-function computations per time step, which is still infeasible for real-time operation on recent hardware. Note that, each value-function evaluation involves non-trivial computation. This includes, for example, the computation of a cost or dynamics function and the verification of constraints, such as collision checks.

Obviously, a substantial reduction of the computational effort is thus necessary. Due to the multiplicative effect of each additional dimension, the most effective alternative is a reduction of the dimensionality of the state space. This is common in the literature, as outlined in section 2, where additional smoothing steps are employed to recover a dynamically feasible trajectory [6]. If, for example, only the dimensions $\mathcal{L}, \mathcal{S}, \dot{\mathcal{S}}$, with the respective actions $\dot{\mathcal{L}}, \ddot{\mathcal{S}}$, $|\dot{\mathcal{L}}| = 7, |\ddot{\mathcal{S}}| = 7$ are considered only $\approx 3.84 \cdot 10^6$ evaluation are required per-time step. Another option, explored in this work, is the focus on only the longitudinal components. This approach requires that a path is selected in advance by a separate path planning step. However, it also avoids additional smoothing steps, which can be computationally expensive [6]. Using the previously established cardinality

estimates, a total of $\approx$ 550000 value function evaluations would be required per time step. If one assumes a trajectory of 10 time steps, approximately $5.5 \cdot 10^6$ evaluations need to be computed in total. Note that earlier publications, such as [2] (publication date 2011), assumed computation on this order of magnitude infeasible for real-time operation. However, on recent (2025) hardware real-time execution is possible, given an efficient implementation, as outlined in the following sections 5 and 6.

5 Planning Method

As justified in the previous section, the presented algorithm utilizes a path velocity decomposition approach. For on-road driving, vehicular motions follow relatively simple paths along a reference line, while most of the complexity stems from the longitudinal movement. Thus, the velocity planning is handled with dynamic programming, while a path along the reference line is obtained via polynomial sampling. Since the computational effort is significant, all steps are implemented in parallel via CUDA on a GPU.

5.1 Environment

In this work, access to an environment model is assumed, which provides accurate information of the objects around the vehicle. In a first step, the information from the environment model is now converted into grid-based data structures, which can be used more efficiently in parallel processing. To this end, all object predictions from the environment model are rendered onto Cartesian grid maps for each considered time step. Since the trajectory planning will follow a reference path, an environment representation in a Frenet frame is most practical. Thus, the Cartesian grid maps are then converted into Frenet grid-maps along the reference line. Next, for each of the grid maps a directional distance map is computed along the direction of the reference line. This distance map tremendously simplifies the subsequent collision checks during the planning phase. It also prevents discretization artifacts, such as the "tunneling" through obstacles, which can happen when planning higher velocities at a coarse temporal resolution. For clarity, all processing steps are also visualized in Figure 1.

5.2 Path Planning

Based on the computed distance maps, a path for the vehicle in a Frenet frame around the given reference line is now planned. To this end, the popular polynomial sampling method using fith-order polynomials akin to Werling et al. [12] is employed. However, compared to [12] the polynomials are parameterized over space instead of time. Since the future velocities of the vehicle are unknown at the path planning stage, a time-based parameterization is not practical. Among all polynomial paths the jerk-minimal path, which also allows the most progress along the reference line, is selected. On this path the dynamic programming algorithm is now applied for velocity planning.

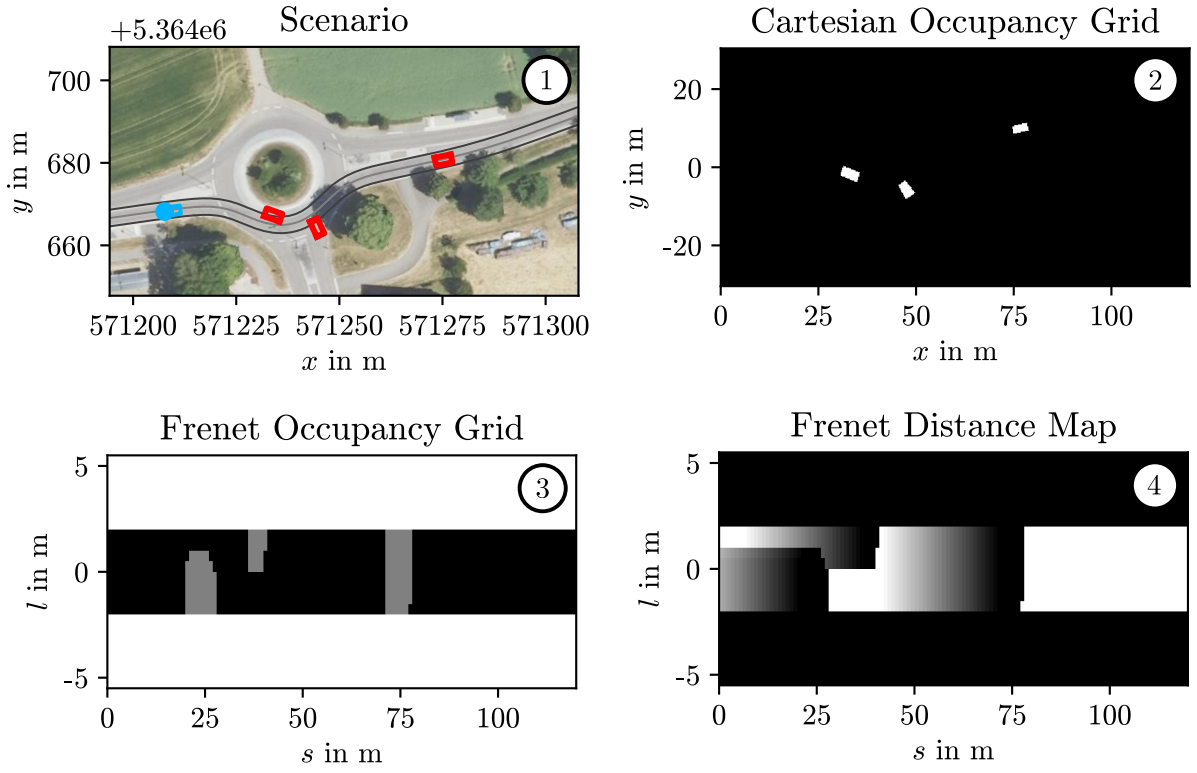

Figure 1: Overview of the different steps of the environmental pre-processing. The first figure shows an exemplary scenario in UTM coordinates. The ego vehicle is depicted in blue, other vehicles in red, and the reference line in gray. The second figure shows the intermediate Cartesian occupancy map, generated from the detection of the environment. Occupied space is depicted as white and free space as black. The third image shows the transformed occupancy map in Frenet coordinates. The space outside the reference corridor is depicted in white, free space in black, and occupied space in gray. The fourth image shows the distance map generated from the occupancy map in Frenet coordinates. The brightness visualizes the distance to the next occupied cell in positive longitudinal direction along the reference line, where black corresponds to non-traversable space. This figure uses aerial images from the DOP20 dataset, https://www.lgl-bw.de/Produkte/Luftbildprodukte/DOP20/, © LGL, www.lgl-bw.de, under the license dl-de/by-2-0, www.govdata.de/dl-de/by-2-0.

5.3 Velocity Planning

For the velocity planning, the state space of the considered dynamic program is selected as $\mathcal{X} = \mathcal{S} \times \mathcal{V} \times \mathcal{A}$, where $\mathcal{V}$ and $\mathcal{A}$ are sets of discrete velocity and acceleration values respectively. A single state $\boldsymbol{x}_t \in \mathcal{X}$ for a time step t thus becomes $\boldsymbol{x}_t = (s_t, v_t, a_t)^T$. Thereby s_t denotes the station on the previously selected polynomial path, v_t the velocity, and a_t the acceleration along the path. Likewise, the action space $\mathcal{U}$ becomes $\mathcal{U} = \mathcal{J}$, with $\boldsymbol{u}_t = j_t$, which corresponds to the longitudinal jerk. Transitions between states, are

then modeled, as movements of constant jerk, with the dynamics $\boldsymbol{f}(\boldsymbol{x}_t, u_t)$ given by

$$\boldsymbol{x}_{t+1} = f(\boldsymbol{x}_t, u_t) = \begin{pmatrix} s_t + v_t\Delta t + \frac{1}{2}a_t\Delta t^2 + \frac{1}{6}j_t\Delta t^3 \\ v_t + a_t\Delta t + \frac{1}{2}j_t\Delta t^2 \\ a_t + j_t\Delta t \end{pmatrix}, \tag{3}$$

where Δt is the temporal difference between two consecutive time steps. Each state and action is then rated by a cost function $c(\boldsymbol{x}_t, u_t)$ defined as

$$c(\boldsymbol{x}_t, u_t) = w_v(v_{\text{trg}} - v_t) + w_{\text{d,safe}} \max(0, s_{t+1} - s_t - d_{\text{safe}}) \tag{4}$$

$$+ w_a a_t^2 + w_j j_t^2 + w_{\text{snap}}((j_{t+1} - j_t)/\Delta t)^2 . \tag{5}$$

Thereby, v_{trg} denotes a target velocity, and d_{safe} a minimal velocity-dependent, safe distance to the next obstacle. Additionally, weighting coefficients $w_{(.)}$ are used to tune the cost function to individual preferences. In addition, hard collision avoidance and velocity constraints are enforced with

$$h(\boldsymbol{x}_t, u_t) = \max(0, v_t - v_{\text{lg}(s_t)}) + \max(0, (s_{t+1} - s_t) - d_{\text{lead}}) , \tag{6}$$

where v_{lg} denotes the legal speed limit, and d_{lead} is the distance to the leading vehicle. If $h > 0$ for any evaluated states, the respective state is marked as invalid and disregarded in further value-function evaluations.

Using the described dynamics, cost, and constraint functions, the dynamic programming algorithm is now applied to find a viable trajectory in the state space. To this end, the value function needs to be evaluated with backward-induction from the last to the first time step. However, since the dynamics function is continuous, a state $\boldsymbol{x}_{t+1} = f(\boldsymbol{x}_t, u_t)$ obtained via the dynamics, is likely not on any of the discrete states in $\mathcal{X}$. This poses a problem, as the value-function in the next time step $t+1$ is only defined at states $\boldsymbol{x}_t \in \mathcal{X}$. Since evaluating the value function exactly for all intermediate state is infeasible, an alternative is needed. However, it can be noted, that the value-function for any continuous next state $\boldsymbol{x}_{t+1}$ is likely to be similar to its discrete neighbors in $\mathcal{X}$. Therefore, a value-function estimate $\tilde{v}(\boldsymbol{x}_{t+1})$ can be obtained by considering these neighboring discrete states. As outlined in [13], this estimate can be obtained via trilinear interpolation of the eight, discrete states surrounding any next state $\boldsymbol{x}_{t+1}$. Apart from this value-function approximation, the dynamic programming algorithm proceeds as outlined in section 3.

6 Evaluation

To evaluate the proposed method, three simulated scenarios are examined in the following. First qualitative results are presented in subsection 6.4, where the trade-off between computation time and solution quality is discussed in subsection 6.5.

6.1 Simulation Environment

All scenarios were executed with the simulation environment of the trajectory planning library available at `https://github.com/uulm-mrm/tpl`. Thereby, the ego vehicle movement is computed with a kinematic bicycle model, while participating vehicles are simulated using the intelligent driver model (IDM) [14]. Additionally, the maximum absolute

Table 1: Parameters of the planning method used for all scenarios

Parameter(s)	Value(s)
$s_{\min}$, $s_{\max}$, $v_{\min}$, $v_{\max}$, $a_{\min}$, $a_{\max}$	0 m, 200 m, 0 m/s, 36 m/s, $-1.5\,\mathrm{m/s^2}$, $1.5\,\mathrm{m/s^2}$
$j_{\min}$, $j_{\max}$	$-1.5\,\mathrm{m/s^3}$, $1.5\,\mathrm{m/s^3}$
$\lvert\mathcal{S}\rvert$, $\lvert\mathcal{V}\rvert$, $\lvert\mathcal{A}\rvert$, $\lvert\mathcal{J}\rvert$	201, 37, 9, 7
T, Δt	9 s, 1.0 s
v_{trg}, v_{lg}	100.0, 13.88
w_v, $w_{\mathrm{d,safe}}$, w_a, w_j, w_{snap}	0.5, 10.0, 1.0, 1.0, 0.5

acceleration actions of the IDM are clipped at $\pm 3\,\mathrm{m/s}$. This intentionally removes the collision-free property of the IDM. Thus, reckless behavior of the ego-vehicle can lead to collisions, which are then detected by the simulator. The simulated vehicles are provided to an environment model as raw detections. This means that the tracking and prediction algorithms of the environment model are also executed during the scenario, which improves the simulation fidelity. For the reference line, either precise map data captured manually using RTK-GPS[1] or map data extracted from aerial images was utilized.

6.2 Parameters

The parameters, used for the construction of the state/action space, and the cost function parameters are summarized in Table 1. Unless specified otherwise, these parameters are unchanged for all scenarios. The state space limits and discretization steps were chosen as explained in section 4. The cost function parameters were determined experimentally, with a focus on obtaining smooth driving behavior.

6.3 Implementation Details

The proposed method is implemented in CUDA utilizing an Nvidia RTX 3090 GPU, where the value-function evaluation is parallelized over all states in a specific time step t. Since $v(\boldsymbol{x}_t)$ is dependent on $v(\boldsymbol{x}_{t+1})$ the time steps must still be processed sequentially. To efficiently implement the approximation $\tilde{v}$, the results of each value-function evaluation are stored in a three-dimensional *CUDA texture*. Textures have the advantage that interpolated access is supported directly by specialized hardware of the GPU. Practically, this means that even though eight memory locations need to be accessed and combined, the interpolated texture access is not significantly slower compared to discrete memory access. In practice, this reduces the total runtime of the dynamic program significantly, which enables a real-time capable implementation and evaluation.

During all scenarios the proposed planning method is executed with a replanning rate of 1 Hz. The start of the trajectory is reinitialized from the previously planned trajectory, shifted by the time since the last planning step. As the environment may change drastically even between replanning cycles, the current trajectory is also continuously validated between replanning steps. This validation only recomputes the cost and constraint functions for the current trajectory and is, therefore, very computationally inexpensive. If the

[1]Real-Time Kinematics Global Positioning System

trajectory becomes invalid, a replanning step is triggered immediately. This way short reaction times are guaranteed, while the load on the GPU is reduced. Additionally, if even the replanning fails to compute a trajectory without constraint violation, an emergency mode is activated. This mode locks the current steering angle and applies a constant acceleration of $-6\,\mathrm{m/s}$ until standstill.

To ensure tracking of the computed trajectory by the simulated vehicle, a PI-controller is employed for longitudinal control. Thereby, the trajectory acceleration is used as a feedforward term, and the station and velocity error along the trajectory as tracking error. For the lateral control, a geometric Stanley controller [15] with an additional feedforward term based on the trajectory curvature is utilized.

6.4 Qualitative Results

To obtain a qualitative impression of the behavior of the planning approach, the described setup was applied on three simulated scenarios. The results for each scenario are depicted in Figure 2 and Figure 3. Broadly, all scenarios can be classified as intersection scenarios, where the ego path conflicts with the path of other traffic participants. Three different road topologies were considered, namely, a roundabout, an unsignalized intersection, and an unprotected left turn. For the roundabout scenario the planner initially ($t = 0\,\mathrm{s}$) approaches the roundabout with moderate velocity and decelerates smoothly until $t \approx 4.5\,\mathrm{s}$ to let a vehicle coming from the left pass. Then it accelerates again until $t \approx 7\,\mathrm{s}$ to enter the roundabout and reduces its velocity to follow the slower vehicles in the roundabout. Note that, the planning algorithm determined that stopping was not required in this case, which enabled efficient merging. In the unsignalized intersection scenario, multiple vehicles from the left and the right need to be prioritized. Since the intersection seems to be blocked for a longer duration, the planner decides to decelerate slowly until $t \approx 12\,\mathrm{s}$. Only after all vehicles have passed, the planner accelerates again leaving the intersection. In the unprotected left turn scenario, the planner starts before the sizeable intersection at $t = 0\,\mathrm{s}$. However, since the lane conflicting with the left turn is further ahead, the planner readily enters the intersection. Due to prioritized traffic coming from the north, it then reduces the vehicle velocity until near standstill at $t \approx 12\,\mathrm{s}$. Only after the vehicles have cleared the intersection, it applies sustained acceleration to pass the intersection quickly.

Note that, during all scenarios, the other traffic participants *did not* behave exactly as predicted by the environment model. While the environment model, applies a constant velocity prediction, the simulator moves the vehicles using the IDM, thus causing varying vehicle accelerations. Due to the fact that the planner considers a relatively large planning horizon of $T = 9\,\mathrm{s}$, changing predictions further in the future can be accounted for with a small impact on the current acceleration. Additionally, the constant revalidation and replanning of the trajectories detects and prevents critical situations early. It must be stressed that the replanning is only triggered on constraint violations, while the cost function additionally pushes the ego vehicle away from obstacles. Intuitively this means, that each replanned trajectory includes a "cost buffer zone", which can be reduced by changing predictions, without immediately invalidating the trajectory.

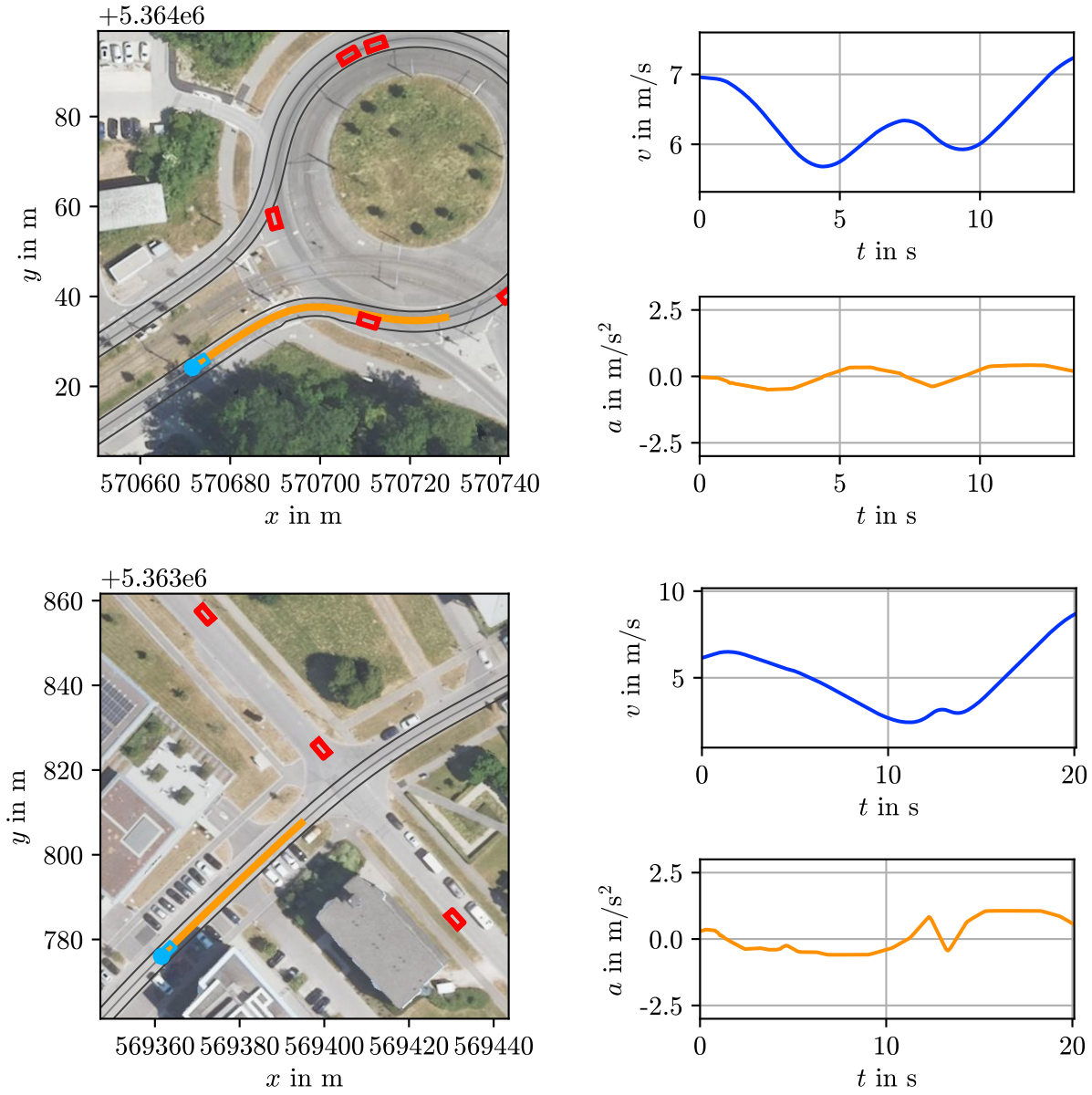

Figure 2: Shows the longitudinal behavior of the proposed method in a simulated roundabout and an unsignalized intersection scenario. The left column depicts a top-down overview in UTM coordinates at $t = 5\,\mathrm{s}$ and $t = 2\,\mathrm{s}$ respectively. The ego-vehicle is depicted in blue, other vehicles in red, the planned trajectory in orange, the route borders in dark gray and the route center in light gray. The right column shows the plot of the ego velocity (blue) and acceleration (orange) during the scenarios. This figure uses aerial images from the DOP20 dataset, https://www.lgl-bw.de/Produkte/Luftbildprodukte/DOP20/, © LGL, www.lgl-bw.de, under the license dl-de/by-2-0, www.govdata.de/dl-de/by-2-0.

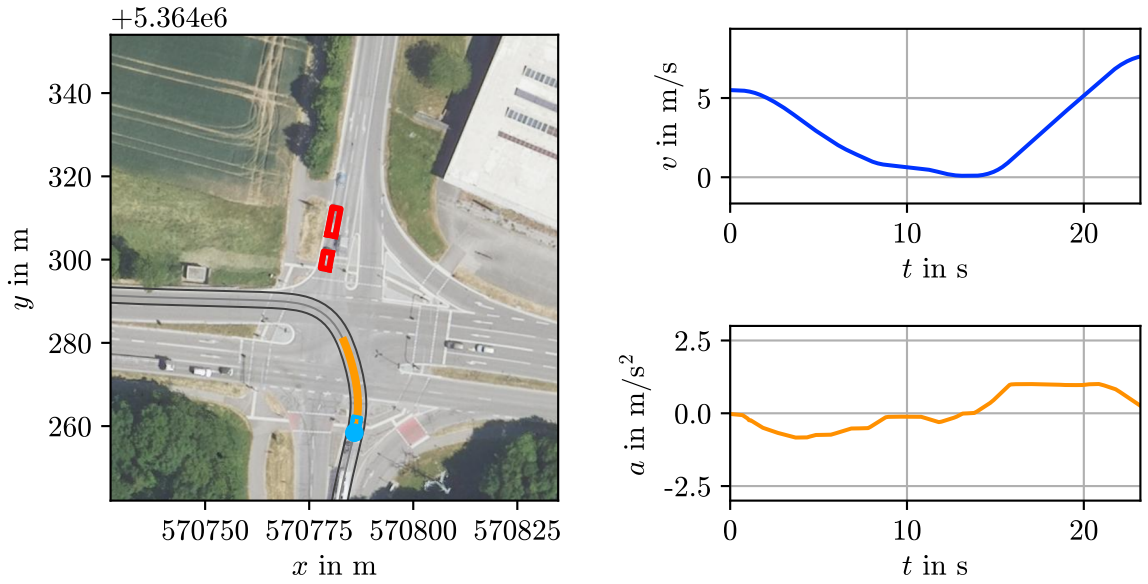

Figure 3: Shows the longitudinal behavior of the proposed method, in a simulated, unprotected left turn scenario, with notation identical to Figure 2. The overview on the left shows the scenario at $t = 2\,\mathrm{s}$. This figure uses aerial images from the DOP20 dataset, https://www.lgl-bw.de/Produkte/Luftbildprodukte/DOP20/, © LGL, www.lgl-bw.de, under the license dl-de/by-2-0, www.govdata.de/dl-de/by-2-0.

6.5 Runtime and Solution Quality

To evaluate the impact of the state space size $|\mathcal{X}|$ on the runtime and solution quality, the roundabout scenario was simulated repeatedly, while varying the number of discretized longitudinal positions $|\mathcal{S}|$ and velocities $|\mathcal{V}|$. Thereby, only the number of discretization steps was changed, while the limits of the state space were kept constant. As cost of the planned behavior, we consider the sum of the absolute longitudinal jerk $J = \sum_{t=0}^{D} |j_t|$, captured during the duration D of the scenario. Additionally, the cost is marked as invalid (denoted "−−"), if a collision occurs, the vehicle leaves the route, or fails to finish the scenario within 60 s. Note that, the cost is intentionally not normalized over the scenario duration. This way, solutions with low jerk and fast progress accumulate the least cost.

The results of this evaluation are depicted in the Figure 4. Most noteably, the left figure shows, that below a certain amount of discretization steps the proposed method cannot find a valid solution. This holds for small $|\mathcal{S}|$ or $|\mathcal{V}|$ and also for moderate $|\mathcal{S}|$ and $|\mathcal{V}|$. Among all combinations with valid results, the accumulated costs are relatively similar. Still, higher values for $|\mathcal{S}|, |\mathcal{V}|$ yield comparatively lower costs. This is expected as more values in the state space allow a more accurate estimation of the value function. In fact, the lowest costs are achived for the second-largest configuration $|\mathcal{S}| = 157, |\mathcal{V}| = 37$.

Additionally, the runtime of the planning algorithm for varying dimension sizes is depicted on the right side of Figure 4. As expected, evaluating more states generally leads to higher average runtimes. However, due to the efficient, parallel GPU implementation the

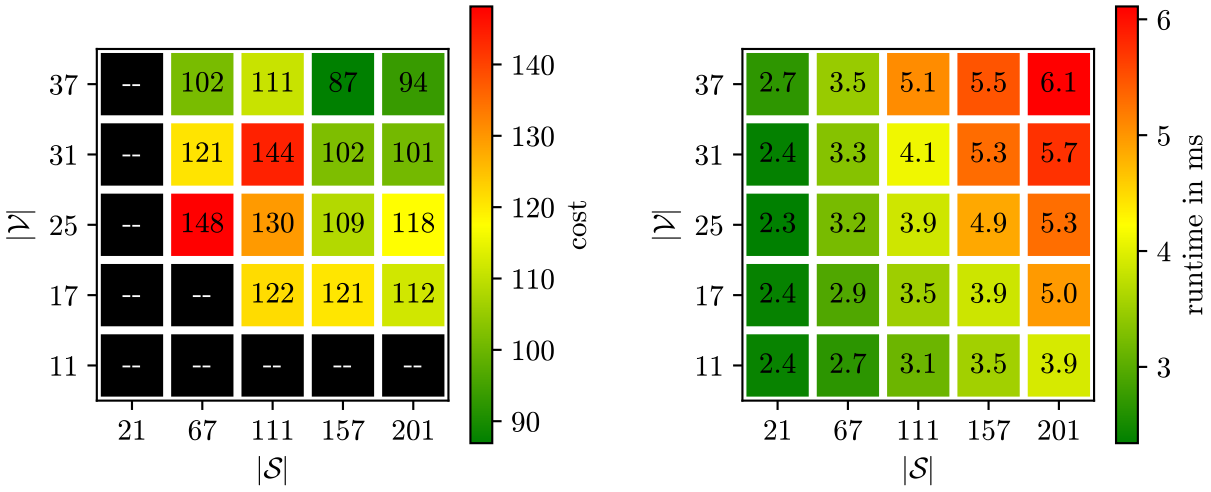

Figure 4: The left plot shows the accumulated cost during the repeated execution of the roundabout scenario from Figure 2, while varying the amount of discrete values in the station and velocity dimension. For each entry the costs have been average over 5 scenario runs with different intial states of the participating vehicles. Fields, where at least one of the scenario runs did produce an invalid result, are denoted as "−−" and marked in black. In the right plot, the corresponding average runtime per planning cycle over all scenario runs of the combined path and velocity planning are displayed.

absolute runtimes are comparatively small in all configurations. While the state space size increases by a factor of ≈ 32, from $|\mathcal{S}| = 21, |\mathcal{V}| = 11$ to $|\mathcal{S}| = 201, |\mathcal{V}| = 37$, the runtime only increases by a factor of ≈ 2.5. Thus, a real-time implementation can be considered feasible, even when utilizing embedded hardware during real-world deployment.

7 Conclusion

In this paper, we examined the dynamic programming concept for trajectory planning of autonomous vehicles. The analysis of the expected application requirements and runtime complexity concludes that a specialized implementation of the algorithm is feasible on current GPU hardware. If increased runtime is permissible, even more accurate and higher quality results may be obtained offline. On a variety of non-trivial scenarios like e.g., roundabout traversal, unsignalized intersections, and unprotected left turns, promising results can be demonstrated in simulation, thus motivating further research. Therefore, it can be concluded that the dynamic programming algorithm should be regarded as a viable tool for online planning and offline validation applications.

References

[1] P. Hart, N. Nilsson, and B. Raphael, "A Formal Basis for the Heuristic Determination of Minimum Cost Paths," *IEEE Transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968.

[2] M. McNaughton, C. Urmson, J. M. Dolan, and J.-W. Lee, "Motion planning for autonomous driving with a conformal spatiotemporal lattice," in *2011 IEEE Int. Conf. on Robotics and Automation*, pp. 4889–4895, May 2011. ISSN: 1050-4729.

[3] S. Heinrich, A. Zoufahl, and R. Rojas, "Real-time trajectory optimization under motion uncertainty using a GPU," in *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 3572–3577, Sept. 2015.

[4] J. Ziegler and C. 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*, pp. 1879–1884, Oct. 2009. ISSN: 2153-0866.

[5] A. Botros and S. L. Smith, "Spatio-Temporal Lattice Planning Using Optimal Motion Primitives," *IEEE Tran. on Intel. Transp. Sys.*, vol. 24, pp. 11950–11962, Nov. 2023.

[6] Y. Meng, Y. Wu, Q. Gu, and L. Liu, "A Decoupled Trajectory Planning Framework Based on the Integration of Lattice Searching and Convex Optimization," *IEEE Access*, vol. 7, pp. 130530–130551, 2019.

[7] H. Fan, F. Zhu, C. Liu, L. Zhang, L. Zhuang, D. Li, W. Zhu, J. Hu, H. Li, and Q. Kong, "Baidu Apollo EM Motion Planner," July 2018. arXiv:1807.08048 [cs].

[8] M. A. Sormoli, K. Koufos, M. Dianati, and R. Woodman, "Towards A General-Purpose Motion Planning for Autonomous Vehicles Using Fluid Dynamics," June 2024. arXiv:2406.05708.

[9] T. Sulkowski, P. Bugiel, and J. Izydorczyk, "Dynamic Trajectory Planning for Autonomous Driving Based on Fluid Simulation," in *2019 24th Int. Conf. on Methods and Models in Automation and Robotics (MMAR)*, pp. 265–268, Aug. 2019.

[10] R. Bellman, R. Bellman, and R. Corporation, *Dynamic Programming*. Rand Corporation research study, Princeton University Press, 1957.

[11] United Nations Economic and Social Council, "Proposal for the 01 series of amendments to un regulation no. 157," 2022.

[12] M. Werling, J. Ziegler, S. Kammel, and S. Thrun, "Optimal trajectory generation for dynamic street scenarios in a frenét frame," *Proceedings - IEEE Int. Conf. on Robotics and Automation*, pp. 987–993, 2010. ISBN: 9781424450381.

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

[14] M. Treiber, A. Hennecke, and D. Helbing, "Congested traffic states in empirical observations and microscopic simulations," *Physical review E*, vol. 62, no. 2, p. 1805, 2000.

[15] G. M. Hoffmann, C. J. Tomlin, M. Montemerlo, and S. Thrun, "Autonomous automobile trajectory tracking for off-road driving: Controller design, experimental validation and racing," in *2007 Am. Control Conference*, pp. 2296–2301, IEEE, 2007.