2018 IEEE Intelligent Vehicles Symposium (IV)
Changshu, Suzhou, China, June 26-30, 2018

Combining Lattice-Based Planning and Path Optimization in Autonomous Heavy Duty Vehicle Applications

Rui Oliveira[1,2], Marcello Cirillo[2], Jonas Mårtensson[1] and Bo Wahlberg[1]

Abstract— **Lattice-based motion planners are an established method to generate feasible motions for car-like vehicles. However, the solution paths can only reach a discretized approximation of the intended goal pose. Moreover, they can be optimal only with respect to the actions available to the planner, which can result in paths with excessive steering. These drawbacks have a negative impact when used in real systems. In this paper we address both drawbacks by integrating a steering method into a state-of-the-art lattice-based motion planner. Unlike previous approaches, in which path optimization happens in an *a posteriori* step after the planner has found a solution, we propose an interleaved execution of path planning and path optimization. The proposed approach can run in real-time and is implemented in a full-size autonomous truck, and we show experimentally that it is able to greatly improve the quality of the solutions provided by a lattice planner.**

I. Introduction

Autonomous driving has become a major focus of research in the past decade. Competitions such as the Darpa Urban Challenge [1] have sparked an interest in the field, and, in recent years, academia and industry alike have started directing their research into further developing these systems [2]–[5].

We focus on motion planning for non-holonomic vehicles in unstructured environments. A direct, practical application of the methods hereby presented lies in automating the transportation of heavy material in industrial environments, such as construction sites or open pit mines. Although motion planning has been the focus of extensive research, and many solutions have been proposed to deal with non-holonomic vehicles [6], a definitive solution does not exist yet.

Steering methods are a class of path planners that calculate feasible motions between vehicle states under the assumption of obstacle-free environments. Computationally efficient steering methods are available only for a few simple systems [7]–[9]. In recent work [10] proposes a steering method that computes vehicle motions while taking into account several steering actuator constraints. Even though the vast majority of applications involve the presence of obstacles, steering methods still prove to be useful, as they are often used as essential components of more sophisticated planners that can consider obstacles [9], [11]–[13].

Sampling-based methods have proven to be very effective for non-holonomic motion planning in the presence of obstacles [6], as they efficiently explore only a discrete subset of the state space. Among them, lattice-based motion planners have achieved very good results by combining the strengths of sampling-based methods with powerful AI search algorithms [14]–[16]. Non-holonomic constraints are respected by pre-computing motion primitives which trap the motions onto a regular lattice. The state space is then explored using graph search algorithms.

Although effective, lattice-based motion planners rely on state space discretization, which means that the goal state of the vehicle can only be chosen from the allowed discrete state space. As a consequence, the precision with which solution paths can approach the intended goal state depends directly on the granularity of the lattice.

Another problem arises from the limited number of motion primitives available in the lattice. The solutions found are optimal with respect to the discretized states and the motions available to the planner on which the lattice is built. This can result in solution paths that require excessive steering action.

In this paper we address both problems described above, and mitigate them by seamlessly combining a state-of-the-art lattice-based motion planner [17] with an efficient steering method [10]. Path optimization, *i.e.* improving certain properties of a path, making use of steering methods can be formulated as a discrete optimization problem, which becomes computationally too expensive to solve with brute force methods. Instead we propose a Greedy Optimization Algorithm which is shown to present results with a quality comparable to those of brute force methods, however doing so in a fraction of the time. As opposed to previous approaches [18], [19], the path optimization is not used solely as an *a posteriori* step to a given planner solution. Instead, it is interleaved with lattice exploration, so that the graph is updated in between planning cycles with optimized new edges. This results in a novel path planning method, that effectively generates smoother paths that are able to bring the system into an exact, non discretized state, while at the same time mitigating the use of sub-optimal motion primitives.

The main contributions of this paper are:

- A new formulation and algorithm for the problem of path optimization, which makes use of steering methods and heuristics to achieve real-time performance;
- A novel approach to path optimization in lattice based planners, which alternates between path planning and path optimization in an interleaved way;
- Simulations with a heavy duty truck showing the benefits and applicability of the proposed algorithms to autonomous vehicles.

*This work was partially supported by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation.

[1]Department of Automatic Control, School of Electrical Engineering and Computer Science, KTH Royal Institute of Technology, Stockholm, Sweden rfoli@kth.se, jonas1@kth.se, bo@kth.se

[2]Scania, Autonomous Transport Solutions, Södertälje, Sweden marcello.cirillo@scania.com

Section II describes the planning framework and introduces the problems to be solved. The path optimization with steering methods problem is formulated in III. The improved planning algorithm is presented in Section IV. Section V presents the results while Section VI concludes the work.

II. Planning framework

In this section we explain the planning framework on which we build upon, and present the challenges that will motivate our contribution.

A. State Lattice Planning

Planning in a state lattice consists of a graph search, in which the graph corresponds to a specially discretized state space. Associated to the discretized space, a set of elementary motions is chosen so that it allows neighbouring discrete states to be connected over feasible motions [20], [21].

We first define the world space $\mathcal{W}$, as the set of possible states of the system. $\mathcal{W}$ can have an obstacle region $\mathcal{O}$, which denotes the set of all states in $\mathcal{W}$ that lie in one or more obstacles. We then define the obstacle-free space as the set of states that do not lie in any obstacle as $\mathcal{W}_{free} = \mathcal{W} \setminus \mathcal{O}$.

Additionally we define the graph to be searched as $\mathcal{G} = \langle V, E \rangle$. Each vertex $v \in V$ represents a discretized state $s \in \mathcal{W}$. In our case, a state s is defined by a 4-tuple $s = (x, y, \theta, \kappa)$, in which x and y represent the position of the state in a 2D environment, θ its heading, and κ its curvature. Each edge $e \in E$ encodes a feasible motion connecting two states. There is an associated nonnegative cost $l(e)$ to each edge e, corresponding to the cost that our system incurs in, when performing it. The set of possible edges E is defined as to respect the kinematic constraints of the vehicle for which we intend to plan [6] and to enforce curvature continuity, *i.e.*, we require the steering wheel rate to be bounded.

A planning problem is defined by a starting state s_{init}, a goal state s_{goal}, and the obstacle-free space $\mathcal{W}_{free}$. A valid solution is a sequence of feasible motions, that are collision-free, connecting s_{init} to s_{goal}. The state space can be explored using graph search algorithms that seek to find a sequence of edges connecting the vertex corresponding to s_{init} to the vertex corresponding to s_{goal}. Ideally returning the sequence of edges $\mathcal{E}_S = \{e_1, \ldots, e_N\}$ that does so with the minimum cumulative cost $J(\mathcal{E}_S) = \sum_{i=1}^{N} l(e_i)$. From $\mathcal{E}_S$, a path Π can be created. A path Π corresponds to a sequence of states, that abide by the feasible motions of the vehicle, and can be used to take the vehicle from s_{init} to s_{goal}. One such graph search algorithm is described below.

B. Lattice Time-Bounded A*

Lattice Time-Bounded A* (LTBA*) [17] is a search algorithm which has been designed to work for real-time problems. The algorithm runs an A* search in a time-sliced fashion, and it is both real-time and incremental.

LTBA* introduces a mechanism to ensure that planned paths take into account systems that can be moving with considerable speed. At each planning cycle, the position and speed of the system, together with the previous solution path, are used to predict the future progression of the system. From this, a committed state s_{com} is computed, that corresponds to the state that lies on the solution path before which, the system, at its current speed, is not capable to come to a stop. Once s_{com} has been selected, the algorithm prunes away the lattice so that all edges starting from the vertices laying along the current (partial) solution and representing the states from s_{init} to s_{goal} are eliminated. Also, these vertices can no longer be used for further exploring the lattice. In practice, this means that at every cycle the algorithm commits to an initial sub-portion of the provided solution by advancing s_{com} along it. As new information arrives to the planner (*e.g.*, from sensor updates), the current solution can be adapted to the new conditions, but only from the vertex representing s_{com} onwards. Further details about LTBA* are outside the scope of this paper, but it is necessary to mention that the algorithm does support emergency braking and replanning in case of obstacles detected on the path before s_{com}.

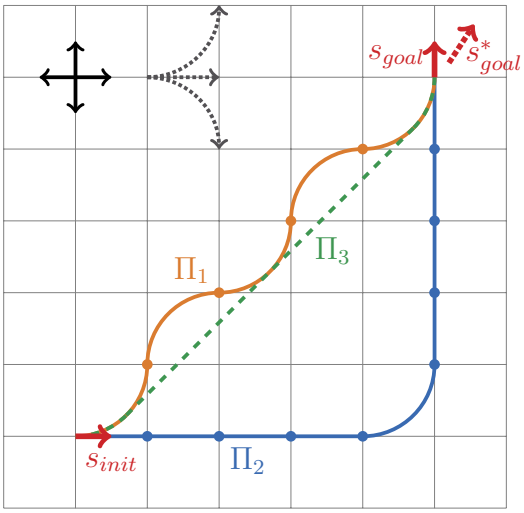

Fig. 1: A toy example of a simplistic lattice with a positional discretization of one meter and four possible headings $\theta = \{0, \pi/2, \pi, -\pi/2\}$. Three elementary motions (dashed lines at the top left) are allowed, move forward (length 1) and turn left/right (length $\pi/2$). A start state s_{init} and a goal state s_{goal} are shown as filled arrows. The undiscretized goal state s^*_{goal} is shown as a dashed arrow. Two possible solution paths, Π_1 and Π_2, lying in the lattice are shown. A third path Π_3 that does not follow the lattice is also shown.

C. Challenges

1) Excessive Steering: One problem with path solutions returned from lattice planners is that they might contain oscillatory behaviour. This is a consequence of both the state discretization and the set of elementary motions available. A lattice planner will return an optimal solution, however under the constraints of being trapped onto the lattice. Such solution is not necessarily optimal if we consider the problem in continuous space with the same optimality criteria.

As an illustrative example consider a simplified lattice as the one shown in Figure 1. We want to find the best path

taking the vehicle from the start state (red arrow s_{init}) to the goal state (red arrow s_{goal}). The best path is defined as the one with the lowest cost $J(\mathcal{E}_S)$. A typical choice for the cost of an edge $l(e_i)$ consists in the length of the path that it represents. Figure 1 shows two possible solution paths, Π_1 and Π_2 that connect the start and goal states. Path Π_1 (orange) has a length $l = 5\pi/2 \approx 7.9$ and path Π_2 (blue) has a length $l = 8 + \pi/2 \approx 9.6$. Since path Π_1 is the shortest, it is returned as the best solution. Even though it is shorter, it is not desirable to select path Π_1 as the solution, as it has an oscillatory behaviour.

It is possible to change the edge costs to prioritize the choice of paths with straight sections, by making the edge costs a function of the length of the path and the amount of steering required. However, designing such a cost function, *i.e.*, finding the right tradeoff between length and steering amount of an edge, can be a challenging and time consuming process and with high subjectivity.

We also show a path Π_3 (green) that is arguably better (shorter and with no oscillatory behaviour). However, Π_3 cannot be a solution, since it does not respect the limitations of the lattice, and therefore cannot be found by the planner.

2) Goal State Discretization: As previously stated, the search is performed between two states, s_{init} and s_{goal}. Since the states are limited to the discretized states of the lattice, it becomes impossible to find solution paths to states outside of the lattice discretization.

A common approach when one wants to compute a path to a state s^*_{goal}, that lies outside of the lattice, is to simply compute a path to the nearest discretized lattice state. One such example is shown in Figure 1, where the desired undiscretized goal state s^*_{goal} is shown as the dashed arrow. Since it does not coincide with any of the discretized states, it is approximated by the closest one, corresponding to s_{goal}. The resulting solution path will then not take the vehicle into the desired state s^*_{goal}, but instead to a nearby state s_{goal}.

3) Reducing Lattice Coarseness: The previous drawbacks can be mitigated by reducing the coarseness of the discretization, including additional dimensions to the state space (*e.g.*, the curvature κ), or increasing the set of available feasible motions. However these strategies will not fully remove the problems, just alleviate them, and they will severely increase the state space the algorithm needs to explore, and consequently its running time.

In the following sections we propose a solution for these problems that can be implemented while respecting the real-time requirements of our system.

III. Path Optimization

Here we introduce an algorithm to optimize paths provided by the lattice planner, in terms of minimizing their length.

A. Problem Formulation

Given an initial and goal states $(s_{init}, s_{goal}) \in V \times V$, the lattice planner described in Section II provides a sequence of vertices $\mathcal{V}_S = \{s_{init}, s_1, \ldots, s_M, s_{goal}\} \subseteq V$ and a sequence of edges $\mathcal{E}_S = \{e_{init,1}, e_{1,2}, \ldots, e_{M,goal}\} \subseteq E$ that connects s_{init} to s_{goal}. An edge $e_{i,j}$ corresponds to a motion connecting s_i and s_j. Let $J(\mathcal{E}') : \mathcal{E}' \subseteq E \rightarrow \mathbb{R}^+_0$ denote the function that maps a sequence of edges $\mathcal{E}'$ to the sum of the cost of each edge. Here we assume that the cost of each edge is the length of the edge, and as such $J(\mathcal{E}')$ becomes the length of the concatenated path formed by them.

We define $\mathcal{V}'_S$ to be an arbitrary subsequence of $\mathcal{V}_S$, and $\mathcal{E}'_S$ to be a sequence of new edges resulting from connecting the vertices in $\mathcal{V}'_S$ over paths generated by the steering method described in [10]. Given two vertices s'_i and s'_{i+1} the steering method is able to generate a path, *i.e.*, an edge $e'_{i,i+1}$, that connects the vertices with a feasible motion. This path has the property of respecting the curvature change limitations that are usually associated with car-like vehicles.

$\mathcal{W}_{free}$ corresponds to the obstacle-free space in which edges must be in order to be collision-free. Furthermore s^*_{goal} corresponds to the undiscretized goal state. The problem that we want to solve can be formalised as follows:

Problem 1: Given the states s_{init}, s^*_{goal} and a solution to the path planning problem $(\mathcal{V}_S, \mathcal{E}_S)$ find a sequence of states $\mathcal{V}'_S = \{s'_1, s'_2, \ldots, s'_N\}$ and a sequence of paths $\mathcal{E}'_S$ connecting the states $\mathcal{V}'_S$, such that:

$$\begin{aligned} \underset{\mathcal{V}'_S \subseteq \mathcal{V}_S}{\text{minimize}} \quad & J(\mathcal{E}'_S) \\ \text{subject to} \quad & e'_{i,i+1} = SteeringMethod(s'_i, s'_{i+1}) \\ & s'_1 = s_{init} \\ & s'_N = s^*_{goal} \\ & \mathcal{E}'_S \in \mathcal{W}_{free} \end{aligned}$$

That is, the objective is to find a subset $\mathcal{V}'_S$ of $\mathcal{V}_S$ that when connected through the paths $\mathcal{E}'_S$, minimizes the length of the edges $J(\mathcal{E}'_S)$. An edge $e'_{i,i+1}$ which is part of the sequence $\mathcal{E}'_S$ corresponds to the path generated by the steering method [10] between vertices s'_i and s'_{i+1}. The sequence $\mathcal{E}'_S$ must also be collision-free, *i.e.*, it must be in $\mathcal{W}_{free}$.

The cost function to minimize is selected to be the length of the path $J(\mathcal{E}'_S)$. One possible alternative would be to minimize the steering/oscillations of the path, as that is the main drawback of lattice planner path solutions. However it is noticed in practice that optimizing with respect to the length of the path, also results in decreasing the oscillations.

It is worth noting that, even though solutions to Problem 1 are collision-free, it might be of interest to have obstacle clearance as an optimization objective. This would result in paths that maximize the safety margin to obstacles.

Figure 2 shows an example of a possible problem instance as described in Problem 1. A path that is provided by the lattice planner (filled line), is composed of several intermediate states (circles), $s_1, s_2, \ldots, s_7$, that are connected over edges $e_{1,2}, e_{2,3}, \ldots, e_{6,7}$. After the optimization, a possible solution path (dashed line) is one that connects states s_1 and s_7 directly. In this particular case, the optimized path results from a steering method between s_1 and s_7, denoted as $e'_{1,7}$.

The optimal solution for Problem 1 can be found using a brute-force method, *i.e.*, by evaluating all possible and valid solutions existing in the search space. However this type of approach is computationally expensive, as the number

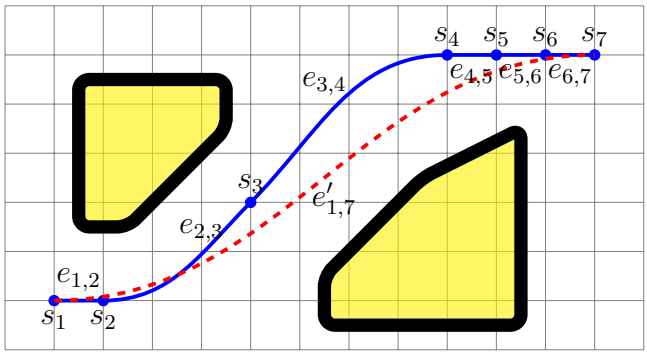

Fig. 2: An example solution path as the filled line (blue). The path is composed of motion primitives connected over intermediate nodes (filled circles). A possible path resulting from a steering method connecting the initial and final nodes is shown as the dashed line (red). $e_{1,2}, e_{2,3}, \ldots, e_{6,7}$ represent the edges connecting the intermediate nodes, as provided by the lattice planner. $e'_{1,7}$ shows a path generated by the steering method, that connects states s_1 and s_7. The polygons are obstacles that the path solution must avoid.

of possible combinations has dimension 2^n, where n is the number of elements in the set $\mathcal{V}_S$. In fact, for a sequence of states $\mathcal{E}_S$ with n states, a brute force approach will require at least $O(2^n)$ steering method queries and its respective collision checks. This makes it unsuitable for our application, in which computations must be kept to a minimum, in order to maintain real-time capabilities.

B. Greedy Optimization

We propose an algorithm to determine the sets $\mathcal{V}'_S$ and $\mathcal{E}'_S$ that tries to find an acceptable solution to Problem 1 within a time frame that complies with our real-time requirements. Thus we require the algorithm to be time-bounded. Also, we will have the opportunity to replan in future planning cycles, so it is beneficial to prioritize optimization in the path edges closer to the current position of the vehicle, which will be executed first. Edges that are farther away can be optimized in later planning cycles. Taking into account these aspects we formulate Algorithm 1.

Algorithm 1 takes as input the sequence of states $\mathcal{V}_S$ corresponding to the lattice planner solution, and the obstacle-free space $\mathcal{W}_{free}$. The algorithm will run for a limited time ΔT_{limit} (line 4), after ΔT_{limit} is elapsed the while loop is exited. Alternatively it can finish before, if it considers the algorithm complete (line 18). In a first step the algorithm tries to generate a feasible path between the first state s_1 to all the states in $\mathcal{V}_S$ that follow it. The path is generated making use of the steering method (line 7) and for it to be valid it needs to be collision-free (line 8). The farthest state from s_1 that it was possible to connect to, is then selected as part of the solution, and both the state and the edge are added at the end of sequences $\mathcal{V}'_S$ and $\mathcal{E}'_S$ (lines 13-14). The process then restarts, but now it tries to connect from the farthest node, to all of the following nodes. In the worst case, each state tries to connect with all following states, resulting in a

Algorithm 1: Greedy Optimization Algorithm

Input: $\mathcal{V}_S = (s_1, \ldots, s_N)$, $\mathcal{E}_S = (e_{1,2}, \ldots, e_{N-1,N})$, $\mathcal{W}_{free}$, ΔT_{limit}
Output: $\mathcal{V}'_S$, $\mathcal{E}'_S$

```
1  V'_S ← (s_1);
2  E'_S ← ( );
3  i ← 1;
4  while ΔT_limit do
5      success ← false;
6      foreach s_j ∈ V_S \ (s_1, ..., s_{i-1}, s_i) do
7          e_{i,j} ← SteeringMethod(s_i, s_j);
8          if e_{i,j} ∈ W_free then
9              e_{i,k} ← e_{i,j};
10             k ← j;
11             success = true;
12     if success then
13         V'_S ← V'_S + s_k;
14         E'_S ← E'_S + e_{i,k};
15         i ← k;
16     else
17         if V_S \ (s_1, ..., s_{i-1}, s_i) = {} then
18             break;
19         else
20             V'_S ← V'_S + (s_i, s_{i+1}, ..., s_N);
21             E'_S ← E'_S + (e_{i,i+1}, ..., s_{N-1,N});
```

complexity of $O(n^2)$ in the number of steering attempts.

Remark 1: It can happen that $\mathcal{V}'_S$ does not include s_N, this means that the algorithm could not compute a feasible path to s_N, either due to infeasibility or shortage of computation time. If this happens, a path that ends up in s_N can still be obtained, by concatenating the original states and edges contained in the original lattice solution $\mathcal{V}_S, \mathcal{E}_S$ that come after the last state in $\mathcal{V}'_S$ (lines 20-21).

By designing the algorithm in this way, we can incrementally improve the original solution throughout consecutive planning cycles. And we do so prioritizing the nearest states of the path, and complying with time limitations. Furthermore, in the worst case, when the algorithm is not able to improve anything, the resulting path will simply be the original lattice planner solution.

C. Reducing the Computational Complexity

Problem 1 can have a very large search space ($2^{\mathcal{V}_S}$ possible solutions). Reducing the search space will reduce the problem complexity. The search space can be reduced by decreasing the number of elements in $\mathcal{V}_S$ over which we optimize. This forms a new of sequence of elements $\mathcal{V}^*_S$ which will correspond to the new decision variables of our optimization problem. However, this will remove a number of possible valid solutions to Problem 1, and possibly remove the optimal or near-optimal solutions from the search space, which is not desirable. Taking into account this trade-off between complexity and the possibility of removing

desired solutions, Algorithm 2 is proposed, which reduces the original search space $2^{\mathcal{V}_S}$, while trying to keep the expressiveness of the original problem in the reduced search space $2^{\mathcal{V}_S^*}$.

Algorithm 2: State Set Reduction

Input: $\mathcal{V}_S = \{s_1, \ldots, s_N\}$, l_{min}
Output: $\mathcal{V}_S^*$
1 $\mathcal{V}_S^* \leftarrow s_1$;
2 $k \leftarrow 1$;
3 **for** $i = 2$ **to** $N - 1$ **do**
4 **if** $Distance(s_k, s_i) > l_{min}$ **then**
5 $\mathcal{V}_S^* \leftarrow \mathcal{V}_S^* \cup s_i$;
6 $k \leftarrow i$;
7 $\mathcal{V}_S^* \leftarrow s_N$;

Function $Distance(s_k, s_i)$ (line 4) is defined as the euclidean distance in the (x, y) position between states s_k and s_i, $||(x_k - x_i, y_k - y_i)||_2$. In essence Algorithm 2 will go over the original set $\mathcal{V}_S$ and remove states that are near each other, *i.e.*, at a distance smaller than $l_{min} \in \mathbb{R}_0^+$ (line 4).

Algorithm 2 is designed taking into account the following practical observation:

Observation 1: Assume Π_1 to be a path generated by the steering method [10] between states s_a and s_b, under the assumption of an obstacle-free environment. Assume a path Π_2 between states s_a and s_c generated in the same conditions. The smaller the value of $Distance(s_b, s_c)$, the more similar (*i.e.* smaller Hausdorff distance) the paths Π_1 and Π_2 are.

Using the intuition provided by Observation 1, Algorithm 2 removes from $\mathcal{V}_S$ states that do not alter the search space $2^{\mathcal{V}_S}$ drastically. If we assume a state s_i to be removed from $\mathcal{V}_S$ by Algorithm 2, we know that there exists another state s_k nearby s_i, that will allow the reduced search space $2^{\mathcal{V}_S^*}$ to contain solution paths which present a high similarity (in the Hausdorff metric) to those that were removed from the original search space $2^{\mathcal{V}_S}$ when removing s_i. There are however exceptions, as certain states in $\mathcal{V}_S$ might be crucial to guarantee a collision-free path, even though they might be very close to other states in $\mathcal{V}_S$.

IV. Interleaving Graph Search and Path Optimization

In this section we propose a novel way to interleave path optimization, as the one described in III, with the LTBA* described in II. We explain why this is not directly implementable, and propose a modification to LTBA* in order to achieve interleaved planning and optimization.

A. Problem Statement

At the end of each planning cycle, the lattice planner returns a solution path as a sequence of states $\mathcal{V}_S$ and edges $\mathcal{E}_S$. This solution is then fed into the optimization algorithm described in III, resulting in a new sequence of states $\mathcal{V}'_S$ and edges $\mathcal{E}'_S$. The current structure of the search graph will then be altered, so that the original states $\mathcal{V}_S$ and edges $\mathcal{E}_S$, are replaced by the optimized states $\mathcal{V}'_S$ and edges $\mathcal{E}'_S$.

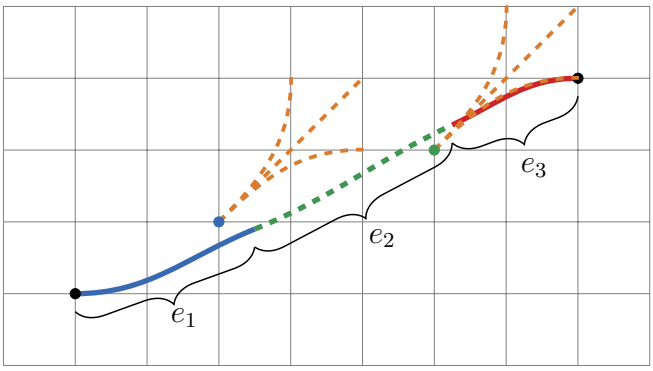

Fig. 3: Splitting an optimized path in order to create escape vertices. The sub-segments e_1, e_2, e_3 of the optimized path become coupled to the nearest escape vertices. A set of edges resulting from the graph search expanding the escape vertices is shown as dashed orange branches.

As an illustrative example, we assume that a given optimization procedure resulted in a $\mathcal{V}'_S$ which only contains the first and last element of $\mathcal{V}_S$. This means that the optimization solution is simply a steering method path between the start state s_1, and end state s_N of $\mathcal{V}_S$. $\mathcal{E}'_S$ then corresponds to a single edge connecting the two states in $\mathcal{V}'_S$. Additionally, we assume that states s_1 and s_N are located far apart, meaning that the edge connecting them will have a large length. According to the mechanisms of LTBA* explained in subsection II-B, the committed state s_{com} will be at least s_N or a state following it. Since planned paths from following planning cycles must include s_{com}, this means that the vehicle is forced to commit to the whole path, and not only to a portion of it. This results in reduced capability for replanning and maneuvering around newly sensed obstacles. To solve this issue, it is necessary to ensure that when changing the structure of the search graph, we do not use edges with a large length.

B. Escape Vertices

In order to solve the problem described above, we introduce a mechanism to split an arbitrary path into several short edges. We introduce the concept of escape vertices, which are vertices that are created in order to split a long optimized path into multiple edges. The mechanism of creation of escape vertices is presented in Algorithm 3.

Algorithm 3 tries to split an edge e' resulting from the optimization in Section III, into a sequence of shorter edges $\mathcal{E}_{esc}$. First it computes the length of the path encoded in e' (line 1), and then it will iterate over the path states (line 5) at regular distances with increments of d_Δ (line 3). For a given path state s_{path}, it finds a nearby lattice vertex s_{esc}, that has not yet been expanded, *i.e.*, it has no child vertices (line 6). In case it has found s_{esc} (line 6), it then proceeds to add it to $\mathcal{V}_{esc}$ (line 10). It also creates an e_{esc}, corresponding to the

Algorithm 3: Finding Escape Vertices

Input: e', d_{min}, d_Δ
Output: $\mathcal{V}_{esc}, \mathcal{E}_{esc}$

```
1  l_e' ← Length(e');
2  d_last ← 0;
3  for d = 0, d_Δ, 2d_Δ, ..., l_e' do
4      if d − d_last > d_min then
5          s_path ← StateAtPathLength(e', d);
6          s_esc ← UnexpandedVerticeNearby(s_path);
7          if s_esc ≠ null then
8              e ← PathSection(e', d_last, d);
9              d_last ← d;
10             V_esc ← V_esc ∪ s_esc;
11             E_esc ← E_esc ∪ e;
```

section of the original path, evaluated from distance d_{last} to d (line 8), and adds it to $\mathcal{E}_{esc}$ (line 11). By updating (line 9) and checking (line 4) the last distance d_{last} at which the path was split, it avoids creating escaping edges that are too short, *i.e.*, with a length smaller than d_{min}.

An arbitrary path might not overlap with any of the discretized lattice vertices, as is the case of the path shown in Figure 3. Thus it is likely that Algorithm 3 will create edges that do not connect discretized lattice vertices with a feasible motion, *i.e.*, one that respects the system model. If the graph search then expands these vertices, the resulting edges will be discontinuous, as illustrated by the orange branches in Figure 3. This violates the assumption of the lattice search, and can result in solution paths that are discontinuous. To deal with this we must introduce a few modifications to the original LTBA*.

The search graph $\mathcal{G}$ is updated by replacing the original lattice planner solution states $\mathcal{V}_S$ and edges $\mathcal{E}_S$ by the optimized states $\mathcal{V}'_S$ and edges $\mathcal{E}'_S$. However the new states $\mathcal{V}'_S$ are flagged as being escape vertices. When a vertex is flagged as an escape vertex, it is known that the edge associated to it does not respect the lattice continuity. As such whenever a potential path solution is about to be returned at the end of the planning cycle, it is checked for path continuity. The path continuity can be checked by evaluating its vertices, and in case they are escape vertices, checking if their connecting edges are discontinuous. In the case that they are, the whole path is invalidated, and an alternative path solution is requested. This is an existing feature of the LTBA* algorithm, which was first introduced in [17] in order to deal with the appearance of new obstacles.

V. Results

In this section we show the results of the methods proposed in the previous sections. The results are obtained using the computing unit in a prototype autonomous heavy duty vehicle for industrial applications, shown in Figure 4.

Fig. 4: An autonomous heavy duty vehicle for which the work presented is developed and tested on.

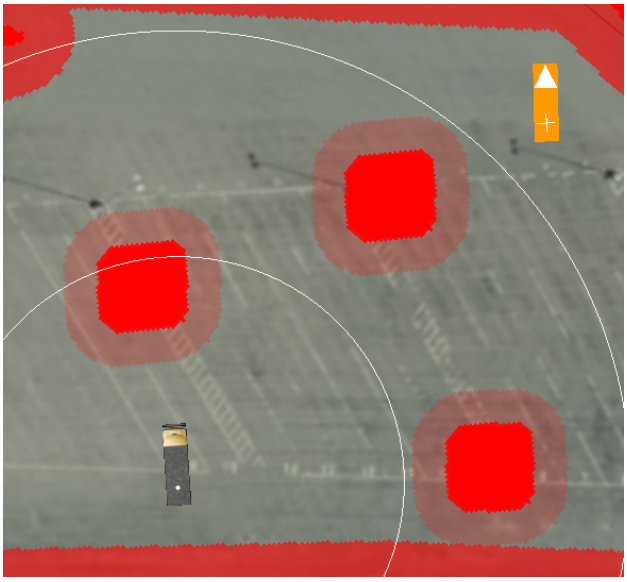

Fig. 5: An example of a planning problem. From the current state (vehicle location), find a path to a goal state (highlighted in orange), while avoiding obstacles (regions in red).

A. Problem Setup

We generate 5000 randomized test scenarios corresponding to typical planning problems. A planning problem is characterized by the initial vehicle state, a goal state at which the vehicle should arrive, and a set of obstacles corresponding to areas in which the vehicle cannot drive. One possible random problem instance is shown in Figure 5. To generate a random planning problem, the initial and goal states are obtained by randomly sampling a uniform distributions to sample x and y coordinates and a heading θ. The coordinates are sampled from a distribution with a width of 100 meters, and the headings from a distribution $[0, 2\pi]$. Three random square obstacles are also generated, from the same uniform distribution. The lattice planner and path optimization are required to run on a continuous loop, at a frequency of 2 Hz.

B. Path Optimization

We compare the proposed Greedy Optimization Algorithm (*Greedy*) against a brute force method (*Brute*) as solvers for Problem 1. For each test scenario we get the lattice planner solution and corresponding states $\mathcal{V}_S$, and then solve Problem 1 using both methods. We measure the execution time Δt and solution cost $J(\mathcal{E}'_S)$ of both methods.

Table I shows the comparison of both methods for 5000 randomized environments, in which the original lattice planner solution has an average length $J(\mathcal{E}_S) \approx 80$ meters. The minimum distance l_{min} is set to 10 meters. We define $\delta_J(\mathcal{E}_S, \mathcal{E}'_S)$ as the normalized difference between the lattice solution length $J(\mathcal{E}_S)$, and the optimized solution length $J(\mathcal{E}'_S)$, *i.e.*, $\delta_J(\mathcal{E}_S, \mathcal{E}'_S) = (J(\mathcal{E}'_S) - J(\mathcal{E}_S))/J(\mathcal{E}_S)$. The larger $\delta_J(\mathcal{E}_S, \mathcal{E}'_S)$ the greater is the improvement in terms of path length reduction. The execution time Δt corresponds to the amount of time required for the algorithm to run.

TABLE I: Comparison of *Brute* and *Greedy* results for 5000 random planning problem instances. The theoretical complexity O of steering method calls with respect to the number of states n is also presented.

Method	*Brute*	*Greedy*
mean $\delta_J(\mathcal{E}_S, \mathcal{E}'_S)$ [%]	-4.0886	-3.9749
std $\delta_J(\mathcal{E}_S, \mathcal{E}'_S)$ [%]	3.1508	3.1574
Measure	***Brute***	***Greedy***
mean $\Delta t[s]$	0.1835	0.0058
std $\Delta t[s]$	0.3375	0.0034
Complexity	$O(2^n)$	$O(n^2)$

Table I shows that *Brute* achieves on average shorter paths than *Greedy*. Even though the *Greedy* results are worse than the *Brute* ones, they are still fairly similar. Comparing the computational times Δt of both algorithms shows a clear advantage for *Greedy*. It can be seen that *Greedy* requires much less time (two orders of magnitude smaller) than *Brute*.

We conclude that *Greedy* finds solutions with a cost $J(\mathcal{E}'_S)$ comparable to that of *Brute*, however it does so in a fraction of the computational time required by *Brute*.

C. Interleaved Planning and Optimization

In this experiment we run the LTBA* and the Path Optimization, using the Greedy Optimization Algorithm, in an interleaved fashion, as explained in IV. For each test scenario, we compare the solution path obtained from the original LTBA* implementation [17] and the solution path obtained from our proposed interleaved planner.

TABLE II: Average results from 5000 planning instances.

Metric	Original Path	Optimized Path
Path Length [m]	76.54	73.51
Straight Length [m]	29.03	53.65
Number Steering Changes	12.42	4.19

Table II shows the aggregated results of these experiments. The mean length of the original solution paths is around 77 meters. When using the optimization procedure we get a decreased mean length of 74 meters. We then see that the paths get, on average, slightly shorter after the optimization.

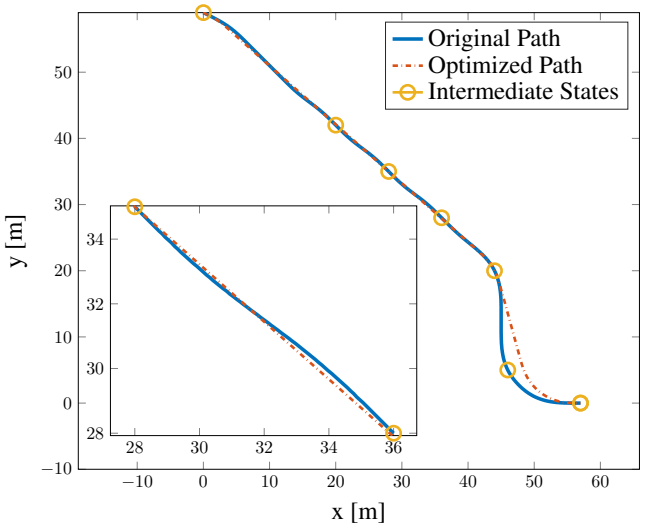

Fig. 6: An example of a planned path (filled line) and the resulting optimized path (dashed line). The circles correspond to intermediate states. A view of a zoomed section highlights the excessive steering of the original planned path.

Besides path length, we also evaluate other metrics associated to path quality, comfort and human-like characteristics.

We introduce the straight length metric, which is a measure of the length of the path corresponding to straight driving, *i.e.* with zero curvature. We see that the original paths have on average 29 meters of straight section. The optimized paths are able to greatly increase this number to 54. This then corresponds to paths that will have more sections of straight driving, which is desirable, since it relates to more comfort, and resembles human-like driving.

A third metric that is measured is the number of steering changes. This metric corresponds to the number of times that a vehicle will have to change between left, right or null steering, when following a path. The original paths present an average of 12 steering changes, while the optimized paths greatly reduce this number to 4 steering changes. Such a drastic reduction shows that the paths have a reduced number of turning maneuvers that the vehicle performs.

D. Goal State Reachability

We also study if the interleaved planner paths are able to deal with the problem of goal state discretization mentioned in II-C.2. We measured that 85% of the interleaved planner paths arrived at the exact goal state s^*_{goal}, instead of at the discretized goal state s_{goal} lying on the lattice, as it happens in the original LTBA*. The other 15% end up the in discretized lattice approximation s_{goal}, just like in the original LTBA*. Ideally the paths would always arrive at the exact goal state s^*_{goal}, however due to the greedy nature of the optimization, which prioritizes optimizing nearer edges of the solution, it is the case that sometimes, the path is optimized without successfully connecting to s^*_{goal}.

E. Illustrative Example

Figure 6 shows a selected problem instance. The paths seem quite similar, but there is a drastic reduction of the number of steering changes, from 20 in the original path (filled line), to 6 in the optimized path (dashed line). Even though the original path seems quite straight, it has several small oscillations, resulting from the drawbacks of the lattice detailed in II-C.1. Our proposed interleaved planner is particularly good at improving the path quality in this aspect.

VI. Conclusions

We have proposed a new planning scheme that combines lattice-based motion planning with path optimization using a steering method. This scheme is successful in reducing excessive steering from planned paths, and in allowing the solutions paths to end up in arbitrary goal states.

We formulated the path optimization problem as a discrete optimization, and propose an algorithm that is able to find a solution with comparable quality to brute force methods, in a fraction of the time. Furthermore the optimization is done in an interleaved fashion with the planning cycles, instead of as a final planning step, as is common in other approaches.

Future work involves the refinement of the algorithms used to find a solution to the discrete optimization and a deeper study of them, with respect to complexity and time guarantees. The mechanisms that allow interleaving the graph search with path optimization, should be studied further, to understand how they influence the efficiency and completeness properties of the graph search.

References

[1] M. Buehler, K. Iagnemma, and S. Singh, *The DARPA urban challenge: autonomous vehicles in city traffic*. Springer, 2009, vol. 56.

[2] C. Pek, P. Zahn, and M. Althoff, "Verifying the safety of lane change maneuvers of self-driving vehicles based on formalized traffic rules," in *2017 IEEE Intelligent Vehicles Symposium (IV)*, June 2017, pp. 1477–1483.

[3] C. Chen, M. Rickert, and A. Knoll, "Motion planning under perception and control uncertainties with space exploration guided heuristic search," in *2017 IEEE Intelligent Vehicles Symposium (IV)*, June 2017, pp. 712–718.

[4] P. E. Ross, "Robot, you can drive my car," *IEEE Spectrum*, vol. 51, no. 6, pp. 60–90, 2014.

[5] P. F. Lima, M. Nilsson, M. Trincavelli, J. Mårtensson, and B. Wahlberg, "Spatial model predictive control for smooth and accurate steering of an autonomous truck," *IEEE Transactions on Intelligent Vehicles*, vol. 2, no. 4, pp. 238–250, Dec 2017.

[6] S. M. LaValle, *Planning algorithms*. Cambridge University Press, 2006.

[7] J. Reeds and L. Shepp, "Optimal paths for a car that goes both forwards and backwards," *Pacific journal of mathematics*, vol. 145, no. 2, pp. 367–393, 1990.

[8] T. Fraichard and A. Scheuer, "From Reeds and Shepp's to continuous-curvature paths," *IEEE Transactions on Robotics*, vol. 20, no. 6, pp. 1025–1035, 2004.

[9] H. Banzhaf, L. Palmieri, D. Nienhser, T. Schamm, S. Knoop, and J. M. Zllner, "Hybrid curvature steer: A novel extend function for sampling-based nonholonomic motion planning in tight environments," in *2017 IEEE 20th International Conference on Intelligent Transportation Systems (ITSC)*, Oct 2017.

[10] R. Oliveira, P. F. Lima, M. Cirillo, J. Mårtensson, and B. Wahlberg, "Trajectory generation using sharpness continuous dubins-like paths with applications in control of heavy duty vehicles," *CoRR*, vol. abs/1801.08995, 2018, accepted for publication at 2018 European Control Conference (ECC). [Online]. Available: http://arxiv.org/abs/1801.08995

[11] Y. Kuwata, J. Teo, G. Fiore, S. Karaman, E. Frazzoli, and J. P. How, "Real-time motion planning with applications to autonomous urban driving," *IEEE Transactions on Control Systems Technology*, vol. 17, no. 5, pp. 1105–1118, 2009.

[12] S. Karaman, M. R. Walter, A. Perez, E. Frazzoli, and S. Teller, "Anytime motion planning using the RRT," in *Robotics and Automation (ICRA), 2011 IEEE International Conference on*. IEEE, 2011, pp. 1478–1483.

[13] D. Dolgov, S. Thrun, M. Montemerlo, and J. Diebel, "Path planning for autonomous vehicles in unknown semi-structured environments," *The International Journal of Robotics Research*, vol. 29, no. 5, pp. 485–501, 2010.

[14] M. Pivtoraiko, R. A. Knepper, and A. Kelly, "Differentially constrained mobile robot motion planning in state lattices," *Journal of Field Robotics*, vol. 26, no. 3, pp. 308–333, 2009.

[15] S. Koenig and M. Likhachev, "D*lite," in *Eighteenth National Conference on Artificial Intelligence*. American Association for Artificial Intelligence, 2002, pp. 476–483.

[16] Y. Björnsson, V. Bulitko, and N. R. Sturtevant, "TBA*: Time-bounded A*," in *Proc. of the 21st Int. Joint Conf. on Artificial Intelligence (IJCAI)*, 2009, pp. 431–436.

[17] M. Cirillo, "From videogames to autonomous trucks: A new algorithm for lattice-based motion planning," in *2017 IEEE Intelligent Vehicles Symposium (IV)*, June 2017, pp. 148–153.

[18] H. Andreasson, J. Saarinen, M. Cirillo, T. Stoyanov, and A. J. Lilienthal, "Fast, continuous state path smoothing to improve navigation accuracy," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, May 2015, pp. 662–669.

[19] D. Dolgov, S. Thrun, M. Montemerlo, and J. Diebel, "Practical search techniques in path planning for autonomous driving," 01 2008.

[20] M. Pivtoraiko, R. A. Knepper, and A. Kelly, "Differentially constrained mobile robot motion planning in state lattices," *Journal of Field Robotics*, vol. 26, no. 3, pp. 308–333, 2009.

[21] M. Cirillo, T. Uras, and S. Koenig, "A lattice-based approach to multi-robot motion planning for non-holonomic vehicles," in *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Sept 2014, pp. 232–239.