2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)
October 25-29, 2020, Las Vegas, NV, USA (Virtual)

Feedback Enhanced Motion Planning for Autonomous Vehicles

Ke Sun, Brent Schlotfeldt, Stephen Chaves, Paul Martin, Gulshan Mandhyan, and Vijay Kumar

***Abstract*—In this work, we address the motion planning problem for autonomous vehicles through a new lattice planning approach, called Feedback Enhanced Lattice Planner (FELP). Existing lattice planners have two major limitations, namely the high dimensionality of the lattice and the lack of modeling of agent vehicle behaviors. We propose to apply the Intelligent Driver Model (IDM) [1] as a speed feedback policy to address both of these limitations. IDM both enables the responsive behavior of the agents, and uniquely determines the acceleration and speed profile of the ego vehicle on a given path. Therefore, only a spatial lattice is needed, while discretization of higher order dimensions is no longer required. Additionally, we propose a directed-graph map representation to support the implementation and execution of lattice planners. The map can reflect local geometric structure, embed the traffic rules adhering to the road, and is efficient to construct and update. We show that FELP is more efficient compared to other existing lattice planners through runtime complexity analysis, and we propose two variants of FELP to further reduce the complexity to polynomial time. We demonstrate the improvement by comparing FELP with an existing spatiotemporal lattice planner using simulations of a merging scenario and continuous highway traffic. We also study the performance of FELP under different traffic densities.**

I. Introduction

Motion planning for self-driving vehicles remains challenging, not only because dynamically feasible trajectories for the ego vehicle should be generated in a structured environment, but also because the behavior of agent vehicles has to be predicted to avoid collisions. A common approach to address the motion planning problem is based on decomposing it into two sub-problems, namely behavior planning and trajectory planning [2]. The underlying heuristics is that solving the two sub-problems is easier than solving the original motion planning problem directly. Behavior planners [3]–[6] generally serve two purposes: 1) generating high-level commands, such as lane keeping or changing; 2) predicting the intentions of the agent vehicles. Trajectory planners [7]–[9] then solve a smooth trajectory often through convex optimization, utilizing the output from the behavior planner.

While decomposition approaches achieve promising results, two issues remain hard to resolve in this framework. First, as pointed out by McNaughton [10], behavior planning often relies on a flawed model of the underlying trajectory planning. This mismatch may lead to unstable or, worse, infeasible trajectory optimization. Sadat [11] discusses the second limitation of the decomposition framework: behavior and trajectory planners optimize different objective functions. As a result, changes in the objective function of behavior planning may have a negative impact on the final trajectory, which then requires re-tuning or re-designing the objective function for the trajectory planner. Lattice planners [12] could be a potential solution to resolve the issues. Instead of decomposing the motion planning problem into behavior and trajectory planning, lattice planners address the problem directly by discretizing the spatiotemporal space and finding the optimal trajectory through graph search.

State lattice approaches [13] are originally designed for rover-like vehicles operating in unstructured environments. Ziegler [14] extends the concept to a spatiotemporal lattice by introducing time as the extra dimension. Although it is efficient to search for an optimal solution within the spatiotemporal lattice, constructing the lattice is time-consuming. This limitation makes the method impractical for continuous autonomous driving tasks, where repetitive lattice construction is required as the local environment around the ego vehicle changes.

McNaughton [10] leverages the fact that the path and velocity of a trajectory can be decoupled [15]. In [10], position is discretized explicitly. Clothoid paths [16] are used to connect positions close to each other. For each path, trajectories are obtained by applying different constant accelerations. Hence, instead of explicit discretization, velocity and time are induced by the starting state and the applied acceleration. Generating velocity profiles with constant acceleration echos another regime of creating a state lattice [17], where the state lattice is no longer obtained by explicitly discretizing the state space, but rather induced by motion primitives obtained by discretizing the control space. The performance of the algorithm in [10] depends heavily on the set of constant accelerations. An overly coarse set results in jerky trajectories, while a fine set significantly increases running time.

Recently, Ajanovic [18] proposes another way of constructing the spatiotemporal lattice in the same spirit as [10]. In addition to the differences in the constraints and types of paths, the velocity dimension is discretized explicitly in [18], leaving position and time to be induced from the initial state and the applied constant acceleration. Since the algorithm structure remains the same, the method in [18] has the same computation complexity as [10] and has the same limitation caused by a finite set of constant accelerations.

Limitations of lattice planners, such as those in [10], [14], [18], are notable. As discussed above, lattice planners require discretization, either explicitly or implicitly, of the

Ke Sun, Brent Schlotfeldt, and Vijay Kumar are with GRASP Lab, University of Pennsylvania, Philadelphia, PA 19104, USA, {sunke, brentsc, kumar}@seas.upenn.edu. Stephen Chaves, Paul Martin, and Gulshan Mandhyan are with Qualcomm Technologies Inc., Philadelphia, PA 19146, USA, {schaves, pdmartin, gmandhya}@qti.qualcomm.com. We gratefully acknowledge the support of Qualcomm Research who sponsored this work.

spatiotemporal space which is often of high dimensionality. The high dimensionality makes the lattice expensive to be repeatedly constructed to adapt to changing environments. Another limitation of existing lattice planners is the lack of modeling of agent vehicles' responsive behavior. In [10], [14], [18], agent vehicles are assumed to maintain constant velocity. Compared with decomposition frameworks, behavior planning provides such modeling, where advanced prediction of agent vehicles' behavior [3]–[5] is possible at the cost of abstracting the controls to only a few high-level maneuvers. However, it is not clear how to apply these methods to lattice approaches.

Additionally, the efficiency of the planning algorithms relies on the representation of the local environment, *i.e.* the map. The map should satisfy three requirements: 1) reflect the geometric structure of the local environment; 2) encode the traffic rules adhering to the road (e.g. an exit-only lane); and 3) be constructed, accessed and updated efficiently. Widely used map formats include OpenStreetMap [19], OpenDrive [20], Lanelet [21] and Lanelet2 [22], which are designed for large scale environments. Although these can be used for motion planning in theory, repeatedly retrieving information from such map representations can be time-consuming. Few papers in the motion planning literature discuss the map representation. [10] briefly reports that the local environment is represented with an occupancy grid map. However, occupancy grid maps [23, Ch.9] are not designed for structured environments, satisfying none of the above requirements.

Contributions: In this work, we propose a Feedback Enhance Lattice Planner (FELP), addressing the motion planning problem of self-driving vehicles in highway scenarios. Our major contributions are summarized as follows:

First, we use an Intelligent Driver Model (IDM) [1, Ch.11] as the feedback policy to control the speed for both the ego and agent vehicles. Given a speed feedback policy for the ego, the velocity at the end of a trajectory and the time used to execute the trajectory are determined uniquely knowing the path and the starting state. Thus, discretization of acceleration, velocity, and time dimension is no longer required. The lattice remains in the 2-D spatial space. Meanwhile, IDM provides an efficient way to model the responsive behaviors for agent vehicles, which integrates seamlessly with lattice planning approaches.

Second, we propose a directed-graph map representation. The proposed representation satisfies all of the previously discussed map requirements. In addition to representing the static environment, vehicles can also be registered onto the map, simplifying the process of collision checking and identifying the relative positions of vehicles.

Finally, we show that the runtime complexity of FELP is significantly reduced compared to the lattice planners in [10], [18]. While the complexity still grows exponentially with the spatial planning horizon, we propose two variants of FELP that bring down the complexity to polynomial time.

II. Problem Formulation

We start the problem formulation by considering the dynamics of a single vehicle. As in [16], a vehicle is modeled as a unicycle,

$$\dot{\boldsymbol{x}}(t) = \Big(\dot{x}(t), \dot{y}(t), \dot{\theta}(t), \dot{v}(t)\Big)^\top = f(\boldsymbol{x}(t), \boldsymbol{u}(t)) = \\ (v(t)\cos\theta(t), v(t)\sin\theta(t), v(t)\kappa(t), a(t))^\top. \tag{1}$$

The state, $\boldsymbol{x} = (x, y, \theta, v)^\top \in \mathcal{X}$, consists of 2-D position (x, y), orientation θ, and speed v. The control $\boldsymbol{u} = (\kappa, a)^\top \in \mathcal{U}$ is composed of curvature κ and acceleration a. More precisely, a is the norm of tangential acceleration.

By stacking the dynamics of a single vehicle in Eq. (1), the dynamics of the local traffic can be constructed as,

$$\dot{\boldsymbol{x}}_s(t) = \begin{pmatrix} \dot{\boldsymbol{x}}_e(t) \\ \dot{\boldsymbol{x}}_0(t) \\ \vdots \\ \dot{\boldsymbol{x}}_{M-1}(t) \end{pmatrix} = \begin{pmatrix} f(\boldsymbol{x}_e(t), \boldsymbol{u}_e(t)) \\ f(\boldsymbol{x}_0(t), \boldsymbol{u}_0(t)) \\ \vdots \\ f(\boldsymbol{x}_{M-1}(t), \boldsymbol{u}_{M-1}(t)) \end{pmatrix}, \tag{2}$$

where $\boldsymbol{x}_e$ is the ego state, $\boldsymbol{x}_i, i = 0, 1, \ldots, M-1$ are the states of agent vehicles. Together, $\boldsymbol{x}_s = \left(\boldsymbol{x}_e^\top, \boldsymbol{x}_0^\top, \ldots, \boldsymbol{x}_{M-1}^\top\right)^\top \in \mathcal{X}^{M+1}$ is the traffic state.

With the assumption that agent vehicles are lane followers, curvature for the agent vehicle i can be generated based on the lane geometry, denoted as $\kappa(\boldsymbol{x}_i) : \mathcal{X} \mapsto \mathbb{R}$. By modulating vehicle speed with IDM, acceleration for the agent vehicle i can be generated by a feedback function $\xi(\boldsymbol{x}_i, \boldsymbol{y}_i, \boldsymbol{\lambda}_i) : \mathcal{X} \times \mathcal{X} \times \Lambda \mapsto \mathbb{R}$. In $\xi(\cdot)$, $\boldsymbol{y}_i$ is the state of the leading vehicle of agent vehicle i. $\boldsymbol{\lambda}_i \in \Lambda$ consists of the IDM hyper-parameters for agent vehicle i. In this work, we assume that $\boldsymbol{\lambda}_i$ are known for all agents. Therefore, $\xi(\boldsymbol{x}_i, \boldsymbol{y}_i, \boldsymbol{\lambda}_i)$ can be simplified to $\xi(\boldsymbol{x}_i, \boldsymbol{y}_i) : \mathcal{X} \times \mathcal{X} \mapsto \mathbb{R}$, embedding the IDM hyper-parameters as constants in $\xi(\cdot)$. Combining $\kappa(\cdot)$ and $\xi(\cdot)$ gives the feedback policies for agent vehicles,

$$\pi_i(\boldsymbol{x}_s) = (\kappa(\boldsymbol{x}_i)\ \xi(\boldsymbol{x}_i, \boldsymbol{y}_i))^\top. \tag{3}$$

The traffic dynamics in Eq. (2) is updated as a result of the introduced feedback policies in Eq. (3),

$$\dot{\boldsymbol{x}}_s(t) = \begin{pmatrix} \dot{\boldsymbol{x}}_e(t) \\ \dot{\boldsymbol{x}}_0(t) \\ \vdots \\ \dot{\boldsymbol{x}}_{M-1}(t) \end{pmatrix} = \begin{pmatrix} f(\boldsymbol{x}_e(t), \boldsymbol{u}_e(t)) \\ f(\boldsymbol{x}_0(t), \pi_0(\boldsymbol{x}_s)) \\ \vdots \\ f(\boldsymbol{x}_{M-1}(t), \pi_{M-1}(\boldsymbol{x}_s)) \end{pmatrix}, \tag{4}$$

which is of the form $\dot{\boldsymbol{x}}_s(t) = f_s(\boldsymbol{x}_s(t), \boldsymbol{u}_e(t))$. In practice, the behavior of agents may not exactly follow the assumed model in Eq. (3). In this work, FELP is applied in the framework of Receding Horizon Control (RHC), which re-plans at a fixed frequency to utilize the latest traffic state.

With the dynamics of the traffic system, the motion planning problem for autonomous driving can be formulated as the following optimal control problem,

$$\min_{\boldsymbol{u}_e(t),\, t_f} \int_{t_0}^{t_f} c_g(\boldsymbol{x}_s(\tau), \boldsymbol{u}_e(\tau))d\tau + c_t(\boldsymbol{x}_s(t_f)) \\ \text{subject to} \quad (1)\ \dot{\boldsymbol{x}}_s(t) = f_s(\boldsymbol{x}_s(t), \boldsymbol{u}_e(t)) \\ (2)\ \text{collision avoidance} \\ (3)\ \text{traffic rules}. \tag{5}$$

In Eq. (5), t_f is the free final time, $c_g : \mathcal{X}^{M+1} \times \mathcal{U} \mapsto \mathbb{R}$ is the running cost, and $c_t : \mathcal{X}^{M+1} \mapsto \mathbb{R}$ is the terminal cost. Note that IDM ensures collision-free driving assuming no acceleration constraints. However, in our implementation, the vehicle acceleration is bounded by physical limits. Therefore, an additional collision avoidance constraint is still required.

A. Motion Primitives of the Ego Vehicle

In this work, the set of motion primitives are feedback control policies instead of open-loop policies as in [17]. Similar to Eq. (3), the j^{th} motion primitive for the ego is of the form,

$$\pi_{e,j}(\boldsymbol{x}_s) = (\kappa_{e,j}(\boldsymbol{x}_e)\ \xi(\boldsymbol{x}_e, \boldsymbol{y}_e)). \tag{6}$$

Some differences between Eq. (6) and Eq. (3) should be noted. First, $\kappa_{e,j}(\cdot)$ is not determined by the lane geometry, but rather a specific path to be followed by the ego. Second, as the ego may change lanes, $\boldsymbol{y}_e$, the ego's leading vehicle, should be determined with special care. In this work, we define the leading vehicle of the ego as the agent which is on the same lane as the front of the ego. For example, if the ego is changing lanes, it switches its leader once its front bumper passes the lane boundary.

A few observations can help simplify the representation of the motion primitives in Eq. (6). First, function $\xi(\cdot)$ is the same across all motion primitives. This can be potentially removed from the representation of a motion primitive and embedded in the traffic dynamics in Eq. (4). Second, the path for each motion primitive, determining $\kappa_{e,j}(\cdot)$, can be constructed knowing the boundary conditions (position, orientation, and curvature at the two end points). In this work, we use the optimization method from [16] to construct such paths. Considering an ego-centric frame where the starting point of a path is fixed at the origin, the path can be determined by only providing the end point $\boldsymbol{p} = (x, y, \theta, \kappa)^\top \in \mathbb{R}^4$. Combining the two observations, the set of motion primitives $\mathcal{M}$ is simply a collection of end points of paths, *i.e.* $\mathcal{M} = \{\boldsymbol{p}_j \in \mathbb{R}^4, j = 0, 1, \dots\}$ with each $\boldsymbol{p}_j$ representing $\boldsymbol{\pi}_{e,j}$ in Eq. (6).

Given the representation of $\mathcal{M}$, only a spatial lattice, instead of spatiotemporal lattice, is required. More specifically, discretization over acceleration, velocity, or time is no longer needed. Acceleration and velocity of the ego and the time to complete a given path are determined by the starting traffic state and the ego speed feedback policy $\xi(\cdot)$. Note that although $\boldsymbol{p}_j \in \mathcal{M}$ is in $\mathbb{R}^4$, only the longitudinal and lateral dimension of the road needs to be discretized. As in [10], orientation and curvature at $\boldsymbol{p}_j$ are determined by the road geometry in order to ensure the path is conformal to the road.

B. Directed-graph Map Representation

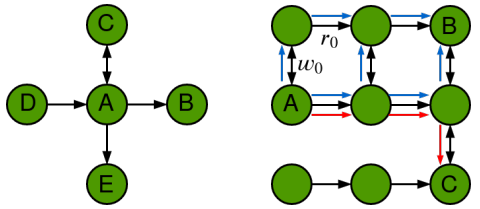
The proposed directed-graph map, $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, consists of vertices, $\mathcal{V}$, and directed edges, $\mathcal{E}$. Vertices are waypoints, consisting of position, orientation, and curvature, regularly sampled along the lane centers. Directed edges model the connectivity between vertices. Each vertex in $\mathcal{V}$ can at most connect to its four neighbors. A direct edge from vertex A to B means that a vehicle can "hop" from a waypoint at A to B without violating any traffic rules, *i.e.* only kinematics need to be considered. Fig. 1a shows a mini directed-graph map example demonstrating possible connections between vertices. Fig. 2 shows an example of the directed-graph map on a highway environment of a larger scale.

The map construction requires interfacing with the map file (e.g. with OpenDrive format) of the environment at a larger scale and route information. Only four queries need to be made

Algorithm 1: Directed-graph Map Construction

Input: $\boldsymbol{p}_0$: the starting waypoint.
r_m: longitudinal range of the map.
r_0: longitudinal resolution of the map.
Output: $\mathcal{G} = (\mathcal{V}, \mathcal{E})$

```
range(p) ← 0
E ← ∅, V ← {p_0}, Q ← {p_0}
while Q is not empty do
    p ← pop(Q)
    // Extend the map forward.
    F ← frontWaypoints(p, r_0)
    p_f ← onRoute(p, F)
    if p_f exists then
        if p_f ∉ V and range(p) + r_0 ≤ r_m then
            range(p_f) ← range(p) + r_0
            V ← V ∪ {p_f}, Q ← Q ∪ {p_f}
        end
        if p_f ∈ V then E ← E ∪ {(p, p_f)}
    end
    /* Extend the map to the left and right with similar
       steps using interfaces leftWaypoint and
       rightWaypoint.                                  */
    ⋮
end
```



(a) (b)

Fig. 1. (a) shows a mini directed-graph map example demonstrating the connections of a vertex, A, with its four neighbor vertices. A, B, and D are on the same lane. $D \to A \to B$ is the direction of the traffic. Therefore, the connections between A, B, and D are one-way. C and E are on adjacent lanes of A. A two-way connection between A and C implies that vehicles can change lanes both from A to C and C to A. In contrast, lane changing is only allowed from A to E. (b) is an illustration for the function $d(\cdot)$ and $\phi(\cdot)$. In (b), w_0 and r_0 are the metric length of the lateral and longitudinal edges. The length of shortest path from both A to B and A to C, is $w_0 + 2r_0$, *i.e.* $d(A, B) = d(A, C) = w_0 + 2r_0$. However, $\phi(A, B) = 3$, *i.e* there are three paths (shown in blue) from A to B with the shortest length, while $\phi(A, C) = 1$ (shown in red).

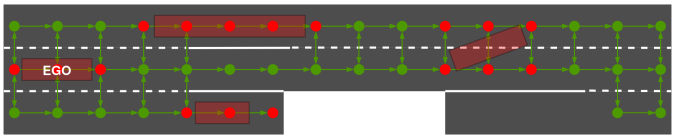


Fig. 2. A directed-graph map overlaid on a highway environment. The right lane (lower) is discontinuous because of the off-ramp and on-ramp. In the middle section of the roads, left (upper) and center (middle) lanes are separated with solid lines preventing lane changes. Therefore, waypoints on adjacent lanes are not connected. No vertex is constructed at the initial section of the on-ramp on the right lane since this section is inaccessible to a vehicle already on the highway. Vehicles, shown as transparent red rectangles, are registered at the corresponding vertices, shown as red dots. If a vehicle is in the process of changing lanes, it occupies vertices on both lanes.

repeatedly: given a waypoint $\boldsymbol{p}$ at a lane center, 1) what are the accessible waypoints if a vehicle moves forward x meters from $\boldsymbol{p}$; 2) what is the waypoint to the left of $\boldsymbol{p}$; 3) what is the waypoint to the right of $\boldsymbol{p}$; 4) is $\boldsymbol{p}$ on the pre-defined route. The first three queries are related to the environment and the last is to the route. Both are assumed to be known a priori. Alg. 1 shows the construction of the directed-graph map.

The directed-graph map can be updated incrementally, extended, shortened, or shifted (a combination of extension and shortening). This can be achieved by maintaining the set of entrance and exit vertices. Entrance vertices are the ones with no vertex connecting to it from the back. Exit vertices are the ones with no vertex connected to its front. The procedure of extending the map is similar to Alg. 1. The difference is that $\mathcal{Q}$, in Alg. 1, is initialized with the set of exit vertices, instead of just the starting waypoint $\boldsymbol{p}_0$. Shortening the map is slightly different. Vertices, with the associated edges, have to be removed starting from the set of entrance vertices. The removal process stops until the desired range is met.

The directed-graph map can also be used to register the position and orientation of vehicles. Based on the state and the length of a vehicle, corresponding vertices in the directed graph are occupied. Fig. 2 shows a directed-graph map with registered vehicles. Using the registered map, relative positions of the vehicles can be easily extracted. For example, one can follow the vertices on the same lane as the ego to identify its leading/following vehicle or its left and right leaders/followers using the vertices on adjacent lanes.

In order to specify the constraints (2) and (3) in Eq. (5), we define a few functions operating on the directed-graph map. For all $\boldsymbol{p} \in \mathcal{V}$, define $s(\boldsymbol{p})$ and $l(\boldsymbol{p})$ as the coordinates of $\boldsymbol{p}$ in the Frenet frame of the road curve. In the case that $\boldsymbol{p} \notin \mathcal{V}$, define $s(\boldsymbol{p}) = s(\boldsymbol{q})$ and $l(\boldsymbol{p}) = l(\boldsymbol{q})$ where $\boldsymbol{q}$ is the projection of $\boldsymbol{p}$ on the graph, *i.e.* $\boldsymbol{q} \in \mathcal{V}$ and $\|\boldsymbol{q} - \boldsymbol{p}\|_2 \leq \|\boldsymbol{q}' - \boldsymbol{p}\|_2$ for all $\boldsymbol{q}' \in \mathcal{V}$. Define $d(\boldsymbol{p}, \boldsymbol{q})$ as the length of the shortest path connecting $\boldsymbol{p}$ and $\boldsymbol{q}$, where path, in this case, is a sequence of edges in $\mathcal{E}$. Since there might be more than one shortest path, $\phi(\boldsymbol{p}, \boldsymbol{q})$ is defined as the number of shortest paths between $\boldsymbol{p}$ and $\boldsymbol{q}$. If either $\boldsymbol{p}$ or $\boldsymbol{q}$ is not in $\mathcal{V}$, d and ϕ take their projections on the graph. Fig. 1b provides examples to illustrate the definitions of $d(\cdot)$ and $\phi(\cdot)$. Finally, we define the function $\mathcal{O}(\boldsymbol{x}(t))$, mapping the state of a vehicle to the set of vertices occupied by the vehicle. Note that the vehicle length is also required to determine the occupied vertices, but not included in $\boldsymbol{x}(t)$. However, we keep the notation $\mathcal{O}(\boldsymbol{x}(t))$ for brevity.

C. Consolidated Problem Formulation

With definitions for the ego motion primitives and the directed-graph map, we are able to consolidate the problem formulation in Eq. (5). First, we reformulate Eq. (5) without the constraints of collision avoidance and traffic rules,

$$\begin{aligned} \min_{\boldsymbol{p}_k,\, k=1,2,\ldots} \; & \sum_k c_s(\boldsymbol{p}_k) + c_t(\boldsymbol{x}_s(t_f)) \\ \text{subject to} \quad & \boldsymbol{p}_0 = (x_e(t_0), y_e(t_0), \theta_e(t_0), \kappa_e(t_0))^\top \\ & \boldsymbol{p}_k \in \mathcal{V},\ s(\boldsymbol{p}_k) \leq s_m \\ & s(\boldsymbol{p}_k) - s(\boldsymbol{p}_{k-1}) = r_0 \cdot n_0 \\ & |l(\boldsymbol{p}_k) - l(\boldsymbol{p}_{k-1})| \leq w_0 \end{aligned} \tag{7}$$

Instead of directly optimizing $\boldsymbol{u}_e(t)$, the optimization is over the motion primitives of the ego, represented as the end points, $\boldsymbol{p}_k$'s, of paths. Because optimization variables have changed, the objective function must be updated. Integrating the running cost c_g gives the stage cost c_s,

$$c_s(\boldsymbol{p}) = \int_{t_1}^{t_2} c_g(\boldsymbol{x}_s(\tau), \boldsymbol{\pi}_{e,\boldsymbol{p}}(\boldsymbol{x}_s(\tau)))d\tau,$$

where t_1 and t_2 are the start and end time for executing the motion primitive represented by $\boldsymbol{p}$. The final time t_f is no longer a variable to be explicitly optimized. Instead, t_f is implicitly determined by the traffic dynamics and the selected sequence of motion primitives. The first constraint in Eq. (7) fixes $\boldsymbol{p}_0$ at the initial state of the ego, which is not necessarily at a vertex in the directed-graph map. $\boldsymbol{p}_k \in \mathcal{V}$ ensures the end points of paths are vertices in the map. $s(\boldsymbol{p}_k) \leq s_m$ sets the spatial planning horizon to s_m. The remaining two constraints instruct the selection of motion primitives. Paths of motion primitives are of constant arclength $r_0 \cdot n_0$ in the longitudinal direction of the road. The constant arclength of paths is defined using r_0, the longitudinal resolution of the map, and n_0, the length of path in terms of edges in the longitudinal direction. In addition, each motion primitive should perform at most one lane change. We note that the last two constraints in Eq. (7) only provide one possible way of selecting motion primitives. Depending on the computation budget, different methods in selecting motion primitives can be applied. The same problem formulation should fit seamlessly. Fig. 3a shows a feasible solution to the problem in Eq. (7).

Thanks to the functions defined on the directed-graph map, the constraints of collision avoidance and traffic rules adhering to the road structure can be formulated. Collision avoidance means vehicles should not overlap,

$$\begin{aligned} \mathcal{O}(\boldsymbol{x}_e(t)) \cap \mathcal{O}(\boldsymbol{x}_i(t)) = \emptyset \\ t \in [t_0, t_f], \quad i \in \{0, 1, \ldots, M-1\}. \end{aligned} \tag{8}$$

In Eq. (8), collision checking among agent vehicles is ignored. Speeds of agent vehicles are modulated with IDM. As long

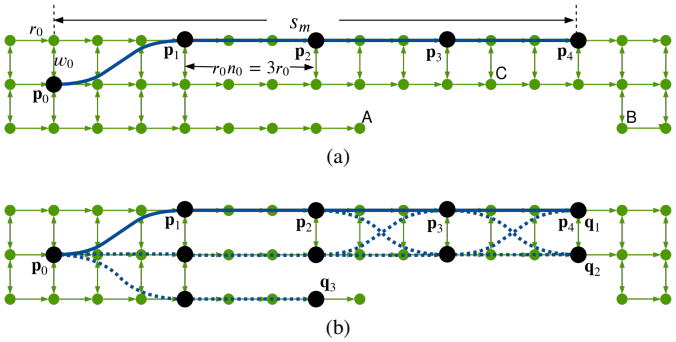


Fig. 3. (a) shows a feasible solution (blue paths) to the problem in Eq. (7) using the directed-graph map in Fig. 2. As in Fig. 1b, w_0 and r_0 are the metric length of the lateral and longitudinal edges of the map. In this example, the planning horizon $s_m = 12r_0$. Longitudinal arclength of paths is $3r_0$, *i.e.* n_0 in Eq. (7) is 3. (b) shows the paths (blue) constructed in the search process of Alg. 2. $\boldsymbol{q}_0$, $\boldsymbol{q}_1$, and $\boldsymbol{q}_2$ are terminal waypoints. The traffic states at these waypoints constitute the terminal set, $\mathcal{T}$, in Alg. 2. The solid path sequence, with end points marked from $\boldsymbol{p}_0$ to $\boldsymbol{p}_4$, is the possible optimal solution.

as initial states are reasonable, collision among agent vehicles rarely happens as we experience in the experiments. Additionally, the constraint of traffic rules adhering to the road are,

$$\phi(\boldsymbol{p}_{k-1}, \boldsymbol{p}_k) = \frac{|l(\boldsymbol{p}_k) - l(\boldsymbol{p}_{k-1})|}{w_0} \cdot n_0 + 1 \tag{9}$$

$$d(\boldsymbol{p}_{k-1}, \boldsymbol{p}_k) \leq r_0 \cdot n_0 + w_0. \tag{10}$$

Eq. (9) prevents illegal lane changes (*e.g.* C to B in Fig. 3a). Eq. (10) forbids paths connecting waypoints on the same lane of a discontinuous road (*e.g.* A to B in Fig. 3a).

To complete the problem formulation, we comment on the cost functions implemented in our work. Running cost c_g includes the acceleration and headway of the ego reflecting both comfort and safety. Since IDM is used to model agent vehicles, it is implicitly assumed that agents would always yield to the ego. To avoid inconsiderate behavior, braking of agents are also included in the running cost. Therefore, aggressive maneuvers of the ego are discouraged. Terminal cost depends on the difference between the terminal and desired speed and travelled distance. The cost of the travelled distance helps the ego avoid exit-only lanes. For example, in Fig. 3b, the path option ending at $\boldsymbol{q}_3$ is likely to be avoided because of the short travelling distance.

III. Algorithm

In this section, we introduce the solution to the problem in Sec. II, and compare the runtime complexity of FELP against [10], [14], [18]. Since the complexity of FELP grows exponentially with the spatial horizon, we propose two variants with polynomial runtime complexity.

Alg. 2 shows the solution algorithm to the problem in Sec. II. Starting from each waypoint $\boldsymbol{p}$ in $\mathcal{Q}$, three possible options are evaluated including lane keep, left and right lane change. For each option, the evaluation starts by locating the end point $\boldsymbol{p}'$ of the path segment. A dynamically feasible path segment, $\boldsymbol{\sigma}$, is then constructed connecting $\boldsymbol{p}$ and $\boldsymbol{p}'$ using the optimization algorithm in [16]. Once the path segment

Algorithm 2: Searching for Optimal Motion Primitives

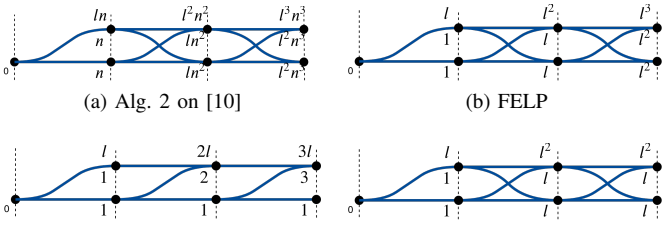
Input: $\boldsymbol{x}_s(t_0)$: the initial traffic state.
$\mathcal{G}$: the directed-graph map.
Output: $\boldsymbol{p}_k$'s: the optimal motion primitives.
$\mathcal{S} \leftarrow \{(\boldsymbol{x}_s(t_0), \boldsymbol{p}_0, 0)\}$, $\mathcal{Q} \leftarrow \{(\boldsymbol{x}_s(t_0), \boldsymbol{p}_0, 0)\}$
while $\mathcal{Q}$ *is not empty* **do**
  $\boldsymbol{x}_s, \boldsymbol{p}, c \leftarrow$ `pop` $(\mathcal{Q})$
  // Lane keep option.
  $\boldsymbol{p}' \leftarrow$ `frontEndPoint` $(\boldsymbol{p}, \mathcal{G})$
  if `satisfyConstraints` $(\boldsymbol{p}, \boldsymbol{p}', \mathcal{G})$ **then**
    $\mathcal{S}, \mathcal{Q} \leftarrow$ `extendGraph` $(\boldsymbol{x}_s, \boldsymbol{p}, c, \boldsymbol{p}', \mathcal{S}, \mathcal{Q})$
  end
  // Examine left and right lane change options.
  ⋮
end
$\mathcal{T} \leftarrow$ `terminals` $(\mathcal{S})$
$\boldsymbol{x}_s, \boldsymbol{p}, c \leftarrow$ `optimalTerminal` $(\mathcal{T})$
return `backtrace` $(\boldsymbol{x}_s, \boldsymbol{p}, c)$

Function $\mathcal{S}, \mathcal{Q} \leftarrow$ `extendGraph` $(\boldsymbol{x}_s, \boldsymbol{p}, c, \boldsymbol{p}', \mathcal{S}, \mathcal{Q})$
  $\boldsymbol{\sigma} \leftarrow$ `dynamicalPath` $(\boldsymbol{p}, \boldsymbol{p}')$
  $\boldsymbol{x}_s'', \boldsymbol{p}'', c_g \leftarrow$ `simulate` $(\boldsymbol{x}_s, \boldsymbol{\sigma})$
  if *collision* **then return** $\mathcal{S}, \mathcal{Q}$
  if $\boldsymbol{p}''$ *is* $\boldsymbol{p}'$ **then**
    $\mathcal{Q} \leftarrow \mathcal{Q} \cup \{(\boldsymbol{x}_s'', \boldsymbol{p}'', c + c_g)\}$
  end
  $\mathcal{S} \leftarrow \mathcal{S} \cup \{(\boldsymbol{x}_s'', \boldsymbol{p}'', c + c_g)\}$
  return $\mathcal{S}, \mathcal{Q}$
end

is available, traffic dynamics can be forward simulated from $\boldsymbol{x}_s$ at $\boldsymbol{p}$ to $\boldsymbol{x}_s''$ at $\boldsymbol{p}''$. Note that it is not necessary that $\boldsymbol{p}''$ overlaps with $\boldsymbol{p}'$, since the ego may not reach $\boldsymbol{p}'$ within finite duration (*e.g.* the traffic congestion). If so, $\boldsymbol{x}_s''$, together with $\boldsymbol{p}''$, will not be added to $\mathcal{Q}$ for further extension, but is treated as a terminal state. Once the spatial horizon is reached, the optimal terminal traffic state is identified, backtracing from which produces the optimal sequence of motion primitives. An example of the constructed paths is shown in Fig. 3b.

In the following, we compare the runtime complexity of FELP with [10], [14], and [18]. The runtime complexity is quantified with the number of evaluated trajectories. Variables that determine the complexity include the discretization density of the 2-D position, 2-D velocity, 2-D acceleration, and time. n and l denote the density of the discretization along the longitudinal and lateral directions of a road, where l is often less than n. To avoid cluttering of expressions, we use n to also denote the discretization density of other dimension.

In [14], there are ln^6 nodes in the lattice (In order to avoid confusion with the vertices in the directed-graph map in Sec. II-B, we use "node" to refer to the vertices in the lattice). Assuming the trajectories going out from a node only connect to the nodes one unit away in the longitudinal dimension, the out-degree of a node is ln^5. Therefore, the total number of



(a) Alg. 2 on [10] (b) FELP

(c) C-FELP (d) R-FELP

Fig. 4. (a), (b), (c), and (d) shows evaluated trajectories for Alg. 2 applied to the problem in [10], Sec. II, and its two variants in Sec. III-A and Sec. III-B respectively. Each figure shows the paths (blue) created for a road with l lanes, although only two lanes are drawn. The vertical dash lines marks the discretization of the longitudinal dimension. Next to each node (**black**) marks the number of trajectory segments ending at the node, the sum of which for the nodes on the same column is marked at the top. The total number of evaluated trajectories can be obtained by summing the values on the top row of each figure.

trajectories to be evaluated is $ln^6 \cdot ln^5 = O\big(l^2 n^{11}\big)$.

Both [10] and [18] use hybrid A* [24] to reduce runtime complexity, which prunes the induced position, velocity, or time using a pre-defined grid. Hybrid A*, as an approximation to graph search algorithms, may fail to find the optimal solution. Therefore, we consider the complexity of using authentic graph search algorithms, such as Alg. 2, for the problems in [10] and [18]. As discussed in Sec. I, both [10] and [18] share the same runtime complexity. We focus only on [10] because of its similarity with this work. Fig. 4a shows the growth of the number of evaluated trajectories as the spatial horizon extends. The total number of trajectories is $\sum_{k=1}^{n} l^k n^k = O(l^n n^n)$. Note that since n is often less than 10, the running time of [10] may not be more than [14].

Fig. 4b shows the growth of the trajectories of FELP. The total number of trajectories is $\sum_{k+1}^{n} l^k = O(l^n)$. The significant reduction of the runtime complexity is due to the usage of IDM as the speed feedback policy in the ego motion primitives. Unlike [10], one no longer has to discretize acceleration, evaluate different constant accelerations over the same path, producing more trajectories. Instead, in FELP, the trajectory of the ego is uniquely determined by its initial state and a path. Constructing paths requires a spatial lattice only. Although the runtime complexity is significantly reduced compared to [10], it still grows exponentially with spatial horizon, n. In the following, we propose two methods to further reduce the runtime to polynomial.

A. Introducing Additional Constraints (C-FELP)

One possible way to reduce the runtime complexity of a graph search is introducing additional constraints, thereby eliminating a significant number of options. In C-FELP, we introduce a new restriction for the ego where at most one lane change is allowed over the entire planning horizon. Considering the spatial planning horizon for a local trajectory planner is on the order of one hundred meter, the new restriction is sensible. Effectively, this is equivalent to replacing the constraint $|l(\boldsymbol{p}_k) - l(\boldsymbol{p}_{k-1})| \leq w_0$ in Eq. (7) with $|l(\boldsymbol{p}_k) - l(\boldsymbol{p}_0)| \leq w_0$.

Algorithm 3: Modified extendGraph

Function $\mathcal{S}, \mathcal{Q} \leftarrow$ `extendGraph` $(\boldsymbol{x}_s, \boldsymbol{p}, c, \boldsymbol{p}', \mathcal{S}, \mathcal{Q})$
- $\boldsymbol{\sigma} \leftarrow$ `dynamicalPath` $(\boldsymbol{p}, \boldsymbol{p}')$
- $\boldsymbol{x}_s'', \boldsymbol{p}'', c_g \leftarrow$ `simulate` $(\boldsymbol{x}_s, \boldsymbol{\sigma})$
- **if** *collision* **then return** $\mathcal{S}, \mathcal{Q}$
- **if** $\boldsymbol{p}''$ *is not* $\boldsymbol{p}'$ **then**
  - $\mathcal{S} \leftarrow \mathcal{S} \cup \{(\boldsymbol{x}_s'', \boldsymbol{p}'', c + c_g)\}$
  - **return** $\mathcal{S}, \mathcal{Q}$
- **end**
- **if** $\boldsymbol{p}'' \in \mathcal{S}$ **then**
  - $\boldsymbol{x}_s''', \boldsymbol{p}'', c''' \leftarrow$ `snapshotAt` $(\mathcal{S}, \boldsymbol{p}'')$
  - **if** $c_g + c \geq c'''$ **then return** $\mathcal{S}, \mathcal{Q}$
  - $\mathcal{S} \leftarrow$ `removeSnapshotAt` $(\mathcal{S}, \boldsymbol{p}'')$
  - $\mathcal{Q} \leftarrow$ `removeSnapshotAt` $(\mathcal{Q}, \boldsymbol{p}'')$
- **end**
- $\mathcal{S} \leftarrow \mathcal{S} \cup \{(\boldsymbol{x}_s'', \boldsymbol{p}'', c + c_g)\}$
- $\mathcal{Q} \leftarrow \mathcal{Q} \cup \{(\boldsymbol{x}_s'', \boldsymbol{p}'', c + c_g)\}$

end

All path end points are either on the same lane or adjacent lanes to $\boldsymbol{p}_0$. Fig. 4c shows the growth of trajectories with the new constraint. The total number of constructed trajectories is $\sum_{k=1}^{n} kl = O(n^2 l)$.

B. Removing Traffic States (R-FELP)

The underlying reason of the $O(l^n)$ runtime complexity of FELP is the fast growing of traffic states at the nodes. Different traffic states at a node are created if the ego can reach the node through different combination of paths. A direct way to address this is to ensure that only one traffic state is maintained at each node. We propose to use the cost-to-come of trajectories to determine which traffic state should be maintained at the nodes. Alg. 3 is an update of the function `extendGraph` reflecting this idea. Fig. 4d shows that the runtime complexity of the modified algorithm is $\sum_{k=1}^{n} l^2 = O(nl^2)$. It should be noted that, although the complexity is reduced compared to $O(l^n)$, the principle of optimality is violated. To see this, the optimal sequence of motion primitives may have higher cost-to-come compared to another sequence that reaches the same intermediate waypoint. As a result, the optimal sequence is removed at an early stage of the algorithm, and thus the returned sequence of motion primitives is non-optimal, in general.

IV. Experiment

In the experiments, We compare FELP and its variants with the spatiotemporal lattice planner in [10] through a merging scenario and highway traffic simulations. In addition, we show the performance of FELP with different traffic densities. For ease of reference, we call the method in [10] STLP, which is the most similar to FELP in terms of the planning methodology. In our implementation of STLP, six accelerations are provided, $\{-8, -4, -2, -1, 0, 1\} m/s^2$. To ensure reasonable runtime, hybrid-A* is applied. Longitudinal speed is discretized to three intervals, while the temporal dimension is

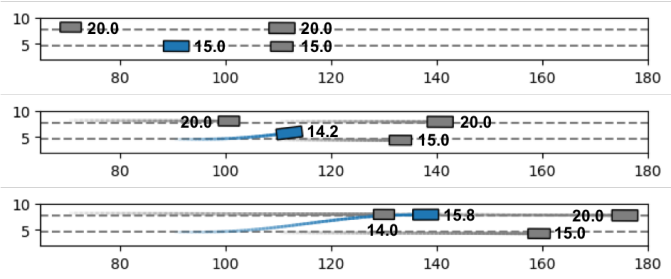


Fig. 5. The merging scenario where the ego (blue) merges into the left lane. The three snapshots shows the traffic at different time, $0.0s$, $1.5s$, $3.25s$ respectively. Next to each vehicle marks the speed (m/s) of the vehicles at the corresponding time.

left undiscretized. All simulations are built with CARLA [25]. The planning algorithms are implemented to run with a single CPU core. The reported timing of the following experiments are obtained with Intel Core i9-9920X running at 3.5GHz. Implementations of the algorithms in this work can be found at https://github.com/KumarRobotics/conformal_lattice_planner.

A. Merging Scenario

In this scenario, the ego is moving at $15m/s$, $20m$ behind a leading vehicle with the same speed, trying to merge to the left. Two vehicles on the left lane are faster moving at $20m/s$, $20m$ ahead and behind the ego respectively. The scenario simulates a typical on-ramp merging case. Since agent vehicles are assumed to move with constant velocity in STLP, lane changing will cause a collision in prediction, and thus is not allowed at this moment. As a result, the ego may fail to merge and has to take the exit. However, lane changing is possible with FELP as agents are modeled with IDM. Fig. 5 shows traffic snapshots with the ego planned with FELP.

As discussed in Sec. II, the assumption that agents would always yield may lead to inconsiderate behavior of the ego. Therefore, the cost induced by agents' braking is used to prevent aggressive actions from the ego. In this example, one can tune the cost function in case more conservative behavior is desired. However, tuning the cost function does not help in producing a lane-change option in STLP.

B. Highway Traffic

The highway traffic simulation is set on the map Town04, a default map in CARLA. To simulate the limited perception range of the ego, traffic is maintained in the neighborhood of the ego vehicle, specifically, $100m$ ahead and $50m$ behind the ego. Within the $150m$ range, 8 agent vehicles are maintained. In the case that an agent vehicle moves out of the range, a new vehicle is added close to either the end or the front of the perception range, simulating the detection of new vehicles. As in Sec. II, all agent vehicles are lane followers with the speed modulated by IDM. The desired speed of the agent vehicles are set to around $20m/s$, close to the desired speed of the ego. In order to improve the similarity with real-world scenarios, behavior of agents are randomized through two ways. First, the IDM hyper-parameters of the agents deviate from nominal values. Second, the online desired speed of agent vehicles is perturbed with slow varying Gaussian noise.

Table I reports the performance of different methods with a one-hour highway traffic simulation. Average values for the metrics are almost the same for all methods, thus omitted. Instead, the percentile data are shown to reflect the extreme statistics. A few observations can be made. Comparing jerk and acceleration, FELP and its variants are less aggressive, implying an improvement in the comfort. This is because IDM produces continuous acceleration if no abrupt change is observed from the traffic. The proposed methods also significantly improve the running time agreeing with the analysis in Sec. III. According to the induced brake, FELP and its variants may expect harder braking from the followers on the target lanes when the ego changes lanes. As discussed in Sec. IV-A, the induced brake can be reduced by tuning the objective function. Comparing C-FELP, R-FELP, and FELP, the performance is almost the same per the metrics, although C-FELP and R-FELP generates sub-optimal trajectories in terms of the objective function. However, C-FELP and R-FELP do further reduce the planning time, which makes them more suitable for online usage.

C. Effect of the Traffic Density

The performance of FELP is also compared with different traffic densities. The setup of the experiments is similar to that in Sec. IV-B. The difference is the number of agent vehicles is configured to 4, 8, and 12 respectively to simulate various traffic densities. Table II reports the experimental results. As the number of agent vehicles increases, both the comfort (reflected by jerk and acceleration) and the safety (reflected by headway and induced brake) of the trajectories are affected, which agrees with the normal driving experience. It is also worth noting that the planning time of FELP increases linearly with the traffic density, making FELP suitable for applications in dense traffic scenarios.

V. Conclusion

In this work, we propose a new lattice planning approach, FELP. FELP applies IDM as the speed feedback policy to modulate the speed of the ego and predict the behavior of agents. IDM enables the responsive behavior of agent vehicles. We show that such modeling can prevent over-conservative behavior of the ego in a merging scenario simulation. Combining IDM with paths, we are able to construct ego motion primitives as feedback policies. With the velocity and acceleration determined by the motion primitives, only spatial dimensions need to be discretized. The reduction in the lattice dimensionality significantly improves the efficiency of FELP. We show this by comparing runtime complexity and actual online planning time of FELP with other lattice planners.

Two directions of extending the current work are promising. First, FELP remains a deterministic planning approach, relying on receding horizon control to cope with the mismatch between the prediction and actual scenarios. Stochasticity can be introduced in modeling the motion of agent vehicles. Planning

TABLE I
COMPARISON OF DIFFERENT METHODS WITH ONE-HOUR HIGHWAY TRAFFIC SIMULATIONS

	jerk[1](m/s^3)	acceleration[1](m/s^2)	speed[1](m/s)	headway[1, 2] (s)	induced brake[3](m/s^2)	planning time (ms)
STLP	$-10.00/10.00$	$-4.00/1.00$	$16.70/20.00$	$1.15/4.70$	2.00	1631
FELP	$-0.43/0.51$	$-0.52/0.54$	$15.45/20.00$	$1.23/4.75$	3.04	276
C-FELP	$-0.36/0.36$	$-0.41/0.38$	$16.04/20.00$	$1.29/4.80$	3.04	153
R-FELP	$-0.35/0.38$	$-0.41/0.34$	$15.95/20.00$	$1.26/4.75$	1.79	232

[1] The data represents 1% and 99% percentile.
[2] Headway is computed whenever there is a leading vehicle for the ego. Otherwise, headway is not defined.
[3] Induced brake refers to brake of the follower on the target lane when the ego changes lanes. The data represent the 1% percentile.

TABLE II
PERFORMANCE OF FELP IN ONE-HOUR HIGHWAY TRAFFIC SIMULATIONS WITH DIFFERENT TRAFFIC DENSITY[1]

agent #	jerk (m/s^3)	acceleration (m/s^2)	speed (m/s)	headway (s)	induced brake (m/s^2)	planning time (ms)
4	$-2.87 \times 10^{-3}/0.0$	$0.00/8.28 \times 10^{-3}$	$19.92/20.00$	$1.80/4.80$	1.35	239
8	$-0.43/0.51$	$-0.52/0.54$	$15.45/20.00$	$1.23/4.75$	3.04	276
12	$-0.50/0.61$	$-0.60/0.54$	$15.18/20.00$	$1.21/4.78$	2.62	317

[1] Data is in the same format as Table I.

with the new model promises the capability of considering various possible future scenarios. Another future direction is extending the current work to urban environments. Compared with highways, urban driving is more challenging because of more complicated traffic rules and more types of dynamical obstacles in addition to agent vehicles.

REFERENCES

[1] M. Treiber and A. Kesting, "Traffic flow dynamics," *Traffic Flow Dynamics: Data, Models and Simulation, Springer-Verlag Berlin Heidelberg*, 2013.

[2] B. Paden, M. Čáp, S. Z. Yong, D. Yershov, and E. Frazzoli, "A survey of motion planning and control techniques for self-driving urban vehicles," *IEEE Transactions on intelligent vehicles*, vol. 1, no. 1, pp. 33–55, 2016.

[3] C. Hubmann, M. Becker, D. Althoff, D. Lenz, and C. Stiller, "Decision making for autonomous driving considering interaction and uncertain prediction of surrounding vehicles," in *2017 IEEE Intelligent Vehicles Symposium*. IEEE, 2017, pp. 1671–1678.

[4] E. Galceran, A. G. Cunningham, R. M. Eustice, and E. Olson, "Multipolicy decision-making for autonomous driving via changepoint-based behavior prediction: Theory and experiment," *Autonomous Robots*, vol. 41, no. 6, pp. 1367–1382, 2017.

[5] Z. N. Sunberg, C. J. Ho, and M. J. Kochenderfer, "The value of inferring the internal state of traffic participants for autonomous freeway driving," in *2017 American Control Conference*. IEEE, 2017, pp. 3004–3010.

[6] C. Hubmann, J. Schulz, G. Xu, D. Althoff, and C. Stiller, "A belief state planner for interactive merge maneuvers in congested traffic," in *2018 21st International Conference on Intelligent Transportation Systems*. IEEE, 2018, pp. 1617–1624.

[7] J. Ziegler, P. Bender, T. Dang, and C. Stiller, "Trajectory planning for berthaa local, continuous method," in *2014 IEEE intelligent vehicles symposium proceedings*. IEEE, 2014, pp. 450–457.

[8] 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," *arXiv preprint arXiv:1807.08048*, 2018.

[9] J. Chen, W. Zhan, and M. Tomizuka, "Autonomous driving motion planning with constrained iterative lqr," *IEEE Transactions on Intelligent Vehicles*, vol. 4, no. 2, pp. 244–254, 2019.

[10] M. McNaughton, C. Urmson, J. M. Dolan, and J.-W. Lee, "Motion planning for autonomous driving with a conformal spatiotemporal lattice," in *2011 IEEE International Conference on Robotics and Automation*. IEEE, 2011, pp. 4889–4895.

[11] A. Sadat, M. Ren, A. Pokrovsky, Y.-C. Lin, E. Yumer, and R. Urtasun, "Jointly learnable behavior and trajectory planning for self-driving vehicles," *arXiv preprint arXiv:1910.04586*, 2019.

[12] L. Claussmann, M. Revilloud, D. Gruyer, and S. Glaser, "A review of motion planning for highway autonomous driving," *IEEE Transactions on Intelligent Transportation Systems*, 2019.

[13] 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.

[14] 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*. IEEE, 2009, pp. 1879–1884.

[15] K. Kant and S. W. Zucker, "Toward efficient trajectory planning: The path-velocity decomposition," *The international journal of robotics research*, vol. 5, no. 3, pp. 72–89, 1986.

[16] A. Kelly and B. Nagy, "Reactive nonholonomic trajectory generation via parametric optimal control," *The International Journal of Robotics Research*, vol. 22, no. 7-8, pp. 583–601, 2003.

[17] M. Rufli and R. Siegwart, "On the design of deformable input-/state-lattice graphs," in *2010 IEEE International Conference on Robotics and Automation*. IEEE, 2010, pp. 3071–3077.

[18] Z. Ajanovic, B. Lacevic, B. Shyrokau, M. Stolz, and M. Horn, "Search-based optimal motion planning for automated driving," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2018, pp. 4523–4530.

[19] M. Haklay and P. Weber, "Openstreetmap: User-generated street maps," *IEEE Pervasive Computing*, vol. 7, no. 4, pp. 12–18, 2008.

[20] M. Dupuis, E. Hekele, A. Biehn, *et al.* Opendrive. [Online]. Available: http://www.opendrive.org

[21] P. Bender, J. Ziegler, and C. Stiller, "Lanelets: Efficient map representation for autonomous driving," in *2014 IEEE Intelligent Vehicles Symposium Proceedings*. IEEE, 2014, pp. 420–425.

[22] F. Poggenhans, J.-H. Pauls, J. Janosovits, S. Orf, M. Naumann, F. Kuhnt, and M. Mayr, "Lanelet2: A high-definition map framework for the future of automated driving," in *2018 21st International Conference on Intelligent Transportation Systems*. IEEE, 2018, pp. 1672–1679.

[23] S. Thrun, W. Burgard, and D. Fox, *Probabilistic robotics*. MIT press Cambridge, 2000, vol. 1.

[24] D. Dolgov, S. Thrun, M. Montemerlo, and J. Diebel, "Practical search techniques in path planning for autonomous driving," *Ann Arbor*, vol. 1001, no. 48105, pp. 18–80, 2008.

[25] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, "CARLA: An open urban driving simulator," in *Proceedings of the 1st Annual Conference on Robot Learning*, 2017, pp. 1–16.