Historical Improvement Optimal Motion Planning with Model Predictive Trajectory Optimization for On-road Autonomous Vehicle

Duong Le[1], Zhichao Liu[1,2], IEEE student member , Jingfu Jin[1], Kai Zhang*[1], and Bin Zhang*[2], IEEE senior member

***Abstract*— This paper presents an efficient, robust, comfortable, and real-time motion planning framework for on-road autonomous vehicles. This proposed framework aims to enhance the performance of motion planning in complex environments such as driving in the urban area. It uses a path velocity decomposition method to separate the motion planning problem into path planning and velocity planning. The novelty lies in the use of Historical data in the SL coordinate in the framework of a tree version of Rapidly-exploring Random Graph (RRT*) technique in path planner, called HSL-RRT*, which grows the path tree efficiently by the data from previous planning cycle. The velocity planner uses a Nonlinear Model Predictive Controller (NMPC) to generate optimal velocity along the path generated from the path planner, taking account of vehicle constraints and comfort. Analytic and simulation results are presented to validate the approach, with a special focus on the robustness and efficiency of the algorithm operating in complex scenarios.**

I. Introduction

Self-driving technology has been developing rapidly over the last few years. Such technology offers great promising and potential society benefits. A vital component of self-driving technology is the ability to plan a safe, feasible, comfortable, and agile motion that enables vehicles to operate safely and efficiently in traffic.

Motion planning for self-driving vehicles, however, confronts significant challenges. Vehicle motion profiles are governed by its underlying dynamics, which must meet the physical constraints (e.g. minimum turning radius, bounding velocity, acceleration and steering angle, etc.). Besides that, comfort is taken into account for vehicle motion while ensuring the safety and following the traffic rule (lane keeping, velocity keeping, etc.). Moreover, motion planner for self-driving vehicles need to have quick response to meet the requirement of real-time applications.

A. Related Works

Three categories of motion planning approaches for autonomous vehicles have been widely studied in both academics and industry: numerical optimization approaches, sampling-based approaches, and path-velocity decomposition approaches.

Numerical optimization approaches formulate the trajectory generation problem as an optimization problem over the state space to find the optimal trajectory. The methods of gradient or conjugate gradient descending [1], [2], sequential quadratic programming [3], dynamic programming [4], and nonlinear methods [5] were applied to find an optimal solution of a well-defined cost function. These approaches have shown efficiency. However, the computation time may significantly increase if the environment is complex, e.g., the presence of multiple obstacles.

Sampling-based approaches take direct samples in the state space to find a set of feasible trajectories and then select the best trajectory based on a cost function. Deterministic sampling-based approach, such as lattice search [6], [7], constructs a discrete search space to find the optimal trajectory. The main disadvantage of lattice search is that it could not guarantee the completeness. Random sampling-based approaches, such as Rapidly-exploring Random Tree [8] and others [9]–[13], have successfully shown the ability to plan a feasible trajectory. The disadvantages are that they often rely on nearest neighbors, distance metrics, probability distributions, and other factors to guide the search, as surveyed in [14], [15]. Furthermore, the resulted trajectory is usually jerky and needs further smoothing. A variant of RRT, called RRT* [16], can guarantee the optimality. However, the optimality depends on the processing time, which is not available for an online planner.

Path velocity decomposition approaches decompose motion planning problem into path planning and velocity planning [7], [17]. The decomposition, thus, breaks down the motion planning problem into two sub-problems with lower dimensionality for the efficiency. The path planning problem generates a kinematically feasible path for the vehicle to follow. After a path is generated, the velocity planning generates a speed profile along the path.

Other path planning approaches include graph search methods [18]–[20], and incremental search path planning [8]. However, the path planned from these approaches is jerky and not optimal. Lattice search [21], on the other hand, heavily depends on the grid size, and has lower flexibility in planning. To get a smooth path, a numerical optimization process is often used [7], [17]. However, the optimization is sensitive to the number of constraints and cannot guarantee the processing time. For velocity planner,

[1,2]Zhichao Liu is with the Department of Electrical Engineering, University of South Carolina, Columbia, SC, and was with American Haval Motor Technology, MI, USA

[1] Duong Le and Jingfu Jin is with American Haval Motor Technology, MI, USA

[1] Kai Zhang was with American Haval Motor Technology, MI, USA. He is now with Argo AI, PA, USA carlzhangk@gmail.com

[2] Bin Zhang is with the department of Electrical Engineering, University of South Carolina, Columbia, SC zhangbin@cec.sc.edu

978-1-7281-4878-6/19/$31.00 ©2019 IEEE

it is often considered as an optimization problem that takes account of various criteria (obstacle avoidance, comfort, fuel consumption, behavior preference, etc.) [7]. The limitation of this approach is that it often does not consider vehicle dynamic and constraints, which may leads to infeasible trajectories. Some efforts [17] use Model Predictive Controller to overcome this limitation.

B. Contributions

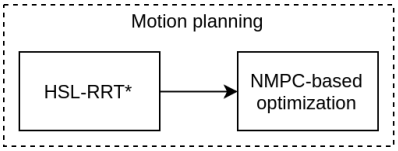


Fig. 1. Overview of our framework

This paper presents an online motion planner framework using the path-velocity decomposition approach to effectively plan trajectories for on-road autonomous vehicles. Fig.1 shows the framework architecture overview. In this framework, an RRT* method in SL coordinate using Historical planning data, denoted as HSL-RRT* is proposed as the path planner. It can effectively find an optimal path by using historical planning data. After the optimal path is obtained, a velocity planner is applied to generate a trajectory for an autonomous vehicle. A Nonlinear Model Predictive Controller (NMPC) is then used to generate an optimal velocity profile to follow the generated optimal path. The NMPC allows us to take into account of vehicle model along with various motion criteria (path tracking, obstacle avoidance, the uncertainty related to the dynamic obstacle predictions, comfort, safety, etc.). Experiments and comparisons demonstrate that our proposed framework is able to handle complex scenarios. Our approach has the following contributions:

- A novel motion planning framework provides effectiveness, robustness, flexibility and feasibility to handle complex scenarios.
- The SL coordinate dissolves the complex geometry of the road. Combined with a sampling-based approach like RRT*, the exploration in SL coordinate is much faster than the one in Cartesian coordinate.
- Using data from the previous cycle combined with RRT* technique, HSL-RRT* is able to compute an optimal path within a limited processing time.
- Comparing with lattice search, which depends on grid resolution to find a solution, our path planning approach is more robust in term of completeness and optimality.
- A NMPC is used to satisfy the requirements of flexibility, feasibility, and comfort.

II. Preliminaries

A. Architecture Overview

Data collected from the vehicle sensors are fed to perception and localization modules, which provides necessary information about vehicle surrounding environments such as the vehicle's current pose, routing information, and obstacle prediction. Also, a High Definition (HD) map is available to provide data of drivable region and a reference line of the driving lane.

As the focus of this paper is on the motion planning, it is assumed that the all the data fed to motion planning module are reliable. With these, this paper proposes a motion planner framework based on path-velocity decomposition approach to generate a safe and smooth trajectory for the vehicle control module.

B. Vehicle Model

Consider a vehicle in a two-dimensional workspace $\mathrm{W} \subset R^2$ with a specific orientation $\theta \in T$ w.r.t. the world coordinate frame. Then the configuration of the vehicle is represented as $q = [x, y, \theta]^T \in \mathrm{SE}(2)$. The vehicle cannot slip in a lateral direction due to the nonholonomic constraints $\dot{x}\sin(\theta) - \dot{y}\cos(\theta) = 0$. Hence, the vehicle's motion can be described by the following nonholonomic kinematic equations

$$\dot{q}(t) = \begin{bmatrix} \dot{x}(t) \\ \dot{y}(t) \\ \dot{\theta}(t) \end{bmatrix} = \begin{bmatrix} \cos(\theta(t)) & 0 \\ \sin(\theta(t)) & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v(t) \\ w(t) \end{bmatrix} \tag{1}$$

where $v(t)$ and $w(t)$ represent the linear and angular velocities of the vehicle, respectively.

C. SL coordinate

From sensor data and HD map, a reference line is generated. The reference line also contains information about traffic regulation and obstacles. From the reference line, an SL coordinate is constructed by using the Frenet coordinate. S represents the distance along the tangential direction of the lane, or the longitudinal distance. Correspondingly, L represents the distance perpendicular to the s-direction, or the lateral distance. The ego vehicle and its surrounding environment are projected to an SL coordinate constructed by the reference line.

III. Path Planner

In this section, we present a path planner in SL coordinate called HSL-RRT*. Our approach generates a curvature-continuous path subject to the directives of the behavior planner and conforms to user preferences. HSL-RRT* algorithm is provided in Algorithm 1.

Algorithm 1 Pseudocode of path planner

```
1: W ← FrameProjectToSLFrame()
2: T ← InitializeTree(n_init)
3: for n_h ∈ H do
4:     ExtendPathTree(n_h)
5: while TerminalCondition() = false do
6:     n_sample ← {null, null, null, HUGE_VALUE}
7:     n_sample.cfg ← SamplePathNode(W)
8:     T ← ExtendPathTree(n_sample)
9: path ← GetPath(T)
10: H ← AddPathHistory(path)
11: return path
```

HSL-RRT* starts by projecting ego vehicle, obstacles in the SL coordinate to construct a work-space $\mathcal{W}$ (Algorithm 1:1). A path tree $\mathcal{T}$ is initialized with root node n_{init} at ego vehicle's initial configuration $\{s_{init}, l_{init}, dl_{init}, ddl_{init}, dddl_{init}\}$. The previous planning data $\mathcal{H}$ is used to extend the tree $\mathcal{T}$ which serves as a guidance for tree $\mathcal{T}$ to grow (Algorithm 1:3-4). The detail of using previous planning data will be discussed in the following section. The algorithm continues with an iterative process. In each iteration, a random node n_{sample} is sampled in $\mathcal{W}$. A tree expansion process (Algorithm 1:5-8) will grow the tree $\mathcal{T}$ by connecting n_{sample} to $\mathcal{T}$. After each iteration, the tree $\mathcal{T}$ gradually improves and grows along the S direction in the SL coordinate. This process is repeated until a terminal condition is met. Depending on the planner objective, a path is derived from the path tree (Algorithm 1:9). The solution path is saved as $\mathcal{H}$ for the next planning cycle (Algorithm 1:10) and is fed to speed planner to generate a smooth trajectory for the controller module to control vehicle's maneuver. Fig.2 shows an interplay of path planner in two consecutive planning cycles. In the following, details about the main component of path planning will be provided.

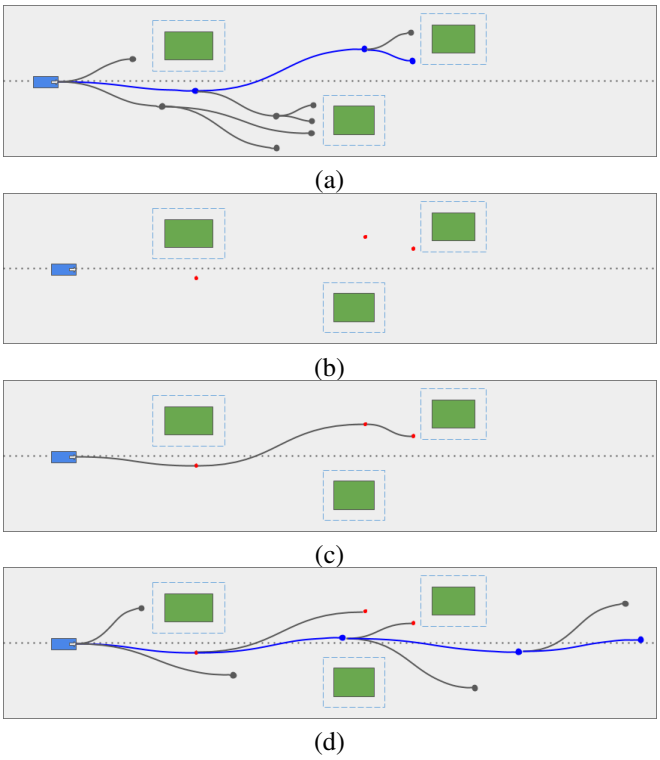


Fig. 2. HSL-RRT* finds a path to navigate complex environment in 2 consecutive planning cycles. In this figure, blue box is the vehicle, green box are obstacles, the blue dots define the unsafe area around obstacles, black dots are the sampling nodes, the black curves show how two sampling nodes are connected. (a) path tree of the first planning cycle, where blue line is the solution of the first planning cycle, (b) red dots are the solution nodes from previous planning cycle which carried to second planning cycle (c) add previous planning solution to the tree of the second planning cycle, (d) path tree of the second planning cycle, where blue line is the solution of the second planning cycle.

A. SL projection (Algorithm 1:1)

Objects, in Cartesian space, are described with location and heading angle (x, y, θ), with addition for the ego vehicle, the curvature and the derivative of curvature (k, dk). Then, objects are projected to SL coordinates $\{s, l, dl, ddl, dddl\}$, which present the state, lateral and lateral derivatives. The SL projection is based on a $Cartesian - Frenet$ frame transformation [6]. Both static and dynamic obstacles are projected to SL coordinate. Static obstacles are projected directly to SL coordinate since the positions of static obstacles are time invariant. On the other hand, dynamic obstacles are described with an obstacle moving trajectory. The trajectory of each dynamic obstacle is discretized into several trajectory points with time, and then these points are projected to SL coordinate. Considering planning trajectory from the previous cycle, collision of the ego vehicle with trajectory points of the dynamic obstacle at each time interval can be estimated.

As shown in Fig.3, increment sampling-based approaches expand poorly in Cartesian coordinate in case of curvy road. The exploration often gets stuck since the lacking of exploration guidance. As a result, the planner fails or generates a poor solution, which can be solved by using a decomposition as a guidance for the tree exploration [13]. However, these often require extra processing time. By utilizing the SL coordinate, the tree exploration can ignore the geometry of the road, which provides a boost for the exploration.

B. Add previous planning data (Algorithm 1:3-4)

After the initial path tree $\mathcal{T}$, data from the previous planning cycle is used to grow the path tree. For autonomous vehicle path planning, the change in the environment of each planning cycle is relatively small. By utilizing previous planning data, the same environment will not be re-explored again. Besides, the planner result will not change much for each planning cycle. However, instead of using the whole previous planning tree data, only the solution from the last planning cycle is carried to the next cycle. Each node n_h from the previous solution $\mathcal{H}$ is added to the path tree sequentially using the same process of tree expansion Algorithm 2. The reason is, although the change in the environment of each planning cycle is relatively small, each node in the last planning tree still needs to be checked for the new planning cycle to ensure a collision-free path. By using RRT* technique in our approach, only the result from the last planning cycle is sufficient. One remarkable feature of RRT* is that [16] the solution quality only gets improved after each iteration once a solution is found. The solution from the last planning cycle can be considered as a solution for the new cycle. Since the last planning solution as a guide for RRT* grows a new tree, after each iteration, RRT* will improve the last solution and expand from the last solution, which results in a new extended path with better quality. This approach also ensures that the solution path does not differ much from the previous one, which reduces the processing time for validation of the history.

C. Tree expansion (Algorithm 1:5-8)

A motion tree $\mathcal{T}$, represented as a directed acyclic graph, is used to conduct a search for a path in $\mathcal{W}$ that satisfies the planner's goal. The path tree $\mathcal{T}$ is expanded by adding a new node n. Each node n is described as $(cfg, parent, curve, cost)$ with $cfg \leftarrow \{s, l, dl, ddl, dddl\}$ is the node configuration in SL coordinate, $parent$ is the node parent, $curve$ is the path from $parent$ to n, and $cost$ is the cost of curve. The main expansion process algorithm is provided in Algorithm 2.

In each iteration, a random node n_{sample} is generated by sampling a random cfg in $\mathcal{W}$. The tree expansion starts by selecting a set of neighbors N_{neighs} which are close to n_{sample}. A node $n_{neigh} \in N_{neighs}$ is selected to be the parent of n_{sample} if it has the best cost connection to n_{sample} and the curve connection from n_{neigh} to n_{sample} is valid. Then n_{sample} is added to path tree $\mathcal{T}$ as a child of n_{neigh} (Algorithm 1:a). If no valid parent is found, the tree expansion starts over sampling. After adding n_{sample} to $\mathcal{T}$, a rewiring process checks if n_{sample} could be the parent of any $n_{neigh} \in N_{neighs}$ (Algorithm 1:b). The tree expansion process is repeated until the terminal condition is met. Details of selecting parent and rewiring process is provided in the following.

The processes of parent selection and tree rewiring are the two most promising features of RRT* and contribute to an asymptotic optimal property of RRT*. Even with the best parent selection and tree rewiring, the path tree $\mathcal{T}$ gradually improves after each iteration [22]. Main components of tree expansion are presented in the following.

Algorithm 2 Pseudocode for tree extension

EXTENDPATHTREE(n_{sample})
1: $N_{\text{neighbors}} \leftarrow$ GETNEIGHBORS($\mathcal{T}, n_{\text{sample}}$)
2: $n_{\text{sample}} \leftarrow$ CONNECTTOPARENT($N_{\text{neighbors}}, n_{\text{sample}}$)
3: **if** $n_{\text{sample}}.parent$ is $null$ **then**
4: **return**
5: $\mathcal{T} \leftarrow$ INSERTPATHNODE($n_{\text{sample}}, \mathcal{T}$)
6: $\mathcal{T} \leftarrow$ REWIRE($\mathcal{T}, N_{\text{neighbors}}, n_{\text{parent}}, n_{\text{sample}}$)

(a)CONNECTTOPARENT($N_{\text{neighbors}}, n_{\text{sample}}$)

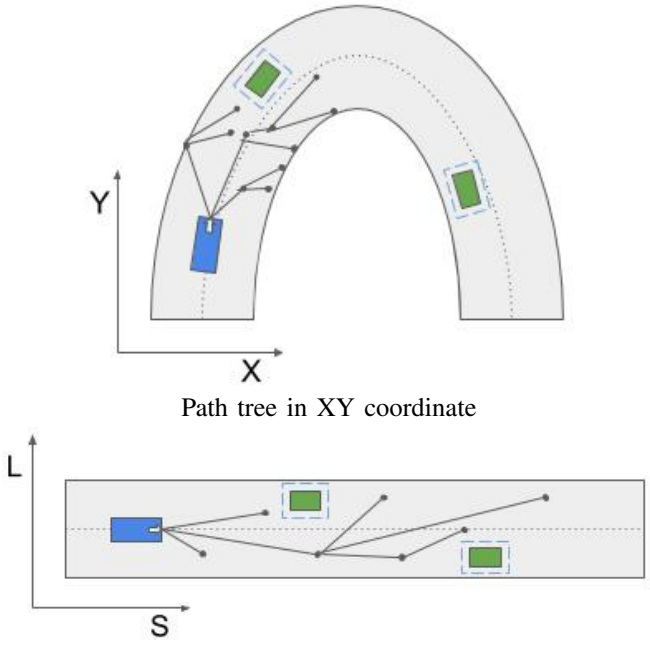
1: **for** $n_{\text{neighbors}} \in N_{\text{neighbors}}$ **do**
2: $path\_curve \leftarrow$ GENERATECURVE($n_{\text{neighbors}}, n_{\text{sample}}$)
3: **if** PATHVALID($path\_curve$) is $false$ **then**
4: *continue*
5: $cost = n_{\text{neighbors}}.cost +$ PATHCOST($path\_curve$)
6: **if** $cost < n_{\text{sample}}.cost$ **then**
7: $n_{\text{sample}} \leftarrow \{n_{\text{sample}}.cfg, n_{\text{neighbors}}, path\_curve, cost\}$
8: **return** n_{sample}

(b)REWIRE($\mathcal{T}, N_{\text{neighbors}}, n_{\text{sample}}$)

1: **for** $n_{\text{neigh}} \in N_{\text{neighbors}} \setminus n_{\text{sample}}.parent$ **do**
2: $path\_curve \leftarrow$ GENERATECURVE($n_{\text{sample}}, n_{\text{neighbors}}$)
3: **if** PATHVALID($path\_curve$) is $false$ **then**
4: *continue*
5: $new\_cost = n_{\text{sample}}.cost +$ PATHCOST($path\_curve$)
6: **if** $n_{\text{neigh}}.cost > new\_cost$ **then**
7: $n_{\text{neigh}}.parent \leftarrow n_{\text{sample}}$
8: $n_{\text{neigh}}.curve \leftarrow path\_curve$
9: $n_{\text{neigh}}.cost \leftarrow new\_cost$

1) Terminal condition (Algorithm 1:5): As mentioned above, HSL-RRT* approach helps to improve the solution quality over each iteration. However, for an online planner, the processing time is always limited. Different termination conditions can be implemented such as the process runs in a limited number of iterations, or the quality of the solution does not get further improvement within a number of iterations. Note that by limiting the processing time of RRT* with a terminal condition, the optimality does not guarantee in each planning cycle. However, by using planning data from the last planning cycle in Algorithm 1:3-4, the solution is always an improvement to the previous planning cycle solution and eventually become an optimal one.

2) Node sampling (Algorithm 1:6-7): A random node n_{sample} is generated by sampling a random configuration $cfg \leftarrow \{s, l, dl, ddl, dddl\}$ in the SL coordinate, where s is sampled with a uniform distribution along the s-direction, l is sampled with a normal distribution with a mean of 0 and variance of half the lane width, and $dl, ddl, dddl$ only matter for the initial position of ego vehicle, so they are all set to 0 for the sampled nodes. This sampling strategy helps the path tree $\mathcal{T}$ grow close to the reference line, resulting in the trajectory of the center lane following.



Path tree in XY coordinate

Path tree in SL coordinate

Fig. 3. Path tree using sampling-based method in Cartesian coordinate vs in SL coordinate

3) Neighbor selection (Algorithm 2:1): Neighbor selection can be based on the distance to n_{sample} or number of nearest neighbor of n_{sample}.

4) Curve generation: A curve is used to connect two nodes by quintic polynomial edges smoothly. A curve is valid if it satisfies several criteria such as no collision, on lane, and geometry shape (to ensure a feasible path). Curve cost function is a combination of curve length, smoothness, obstacle avoidance, and lane cost functions and is given as:

$$\mathcal{C}_{\text{total}}(curve) = w_1\mathcal{C}_{\text{length}}(curve) + w_2\mathcal{C}_{\text{smooth}}(curve) + w_3\mathcal{C}_{\text{obs}}(curve) + w_4\mathcal{C}_{\text{lane}}(curve) \quad (2)$$

where $\mathcal{C}_{\text{length}}$ is the length cost of the curve, $\mathcal{C}_{\text{smooth}}$ is the smooth cost of the curve (the change in lateral direction of the curve), $\mathcal{C}_{\text{obs}}$ is the obstacle cost of the curve, related to how close the curve is to obstacles, $\mathcal{C}_{\text{lane}}$ is the cost of curve following and stay to the center of lane, and w_1, w_2, w_3, w_4 are weighting factors.

5) Path generation: After a terminal condition is met, a path generator provides a solution. The path generation first selects a node n_{goal} in path tree $\mathcal{T}$ as the goal node. The criteria of goal node n_{goal} is that it satisfies the path planner goal such as traveling at least a distance s in s-direction and having the best cost. After a goal node n_{goal} is selected, a set of node solution is obtained by travel back from the goal node n_{goal} to the initial node n_{init}. Set of node solution is saved as history $\mathcal{H}$ for the next planning cycle. A path is generated by concatenating the curve of every node in the set of node solution.

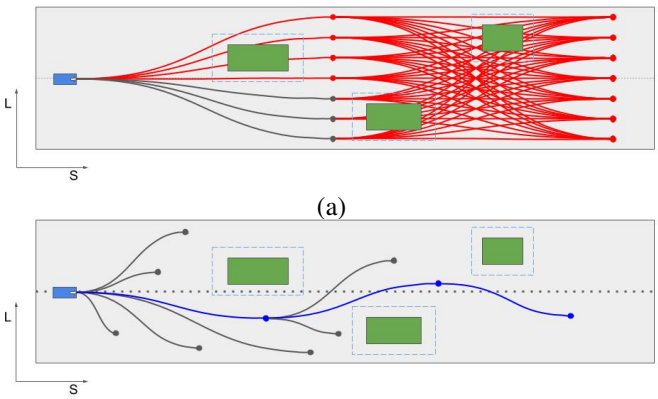


Fig. 4. Navigation through many obstacles. (a) Lattice path planner fails to generate a path. (b) HSL-RRT* successfully finds a path using sampling-based approach. Video comparison is provided at `https://goo.gl/1U7tu7`

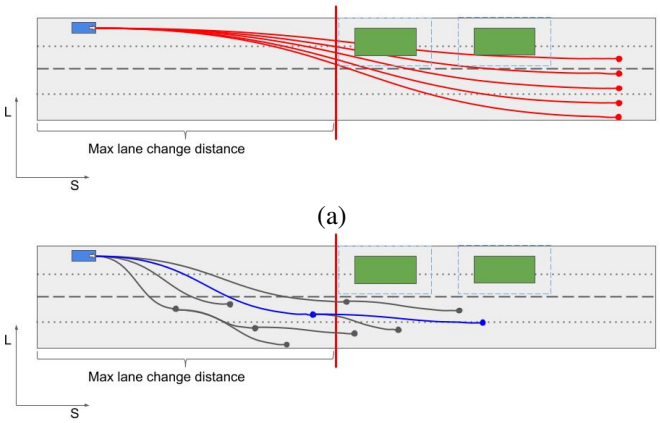


Fig. 5. Short lane change. (a) Lattice path planner fails to generate a path for a short lane change. (b) HSL-RRT* successfully finds a path using sampling-based approach. Video comparison is provided at `https://goo.gl/Sc1mEk`

IV. Velocity Planner

This section presents a NMPC controller, which is used to generate a velocity profile for a reference path generated from path planner in the previous section.

NMPC can be employed to formulate the velocity profile generation problem. The vehicle model (1) is discretized with interval Δt such that

$$q_{k+1} = q_k + \begin{bmatrix} v_k \Delta t \cos(\phi_k) \\ v_k \Delta t \sin(\phi_k) \\ \omega_k \Delta t \end{bmatrix} \quad (3)$$

where $\phi_k = \theta_k + \frac{1}{2}\omega_k \Delta t$ and the index k represents the k^{th} step. Then, the model can be rewritten in a compact form as

$$q_{k+1} = f^d(q_k, u_k) \quad (4)$$

where $f^d()$ is a function of the vehicle state q_k and the control input $u_k = [v_k, \omega_k]^T$.

A set of the vehicle state $\mathbf{q}$ and the control input $\mathbf{u}$ with the given prediction horizon H can be defined as

$$\mathbf{q} = \{q_k | q_{min} < q_k < q_{max}\}, k \in \{1, \cdots, H\} \quad (5)$$

and

$$\mathbf{u} = \{u_k | u_{min} < u_k < u_{max}\}, k \in \{1, \cdots, H\} \quad (6)$$

where q_{min} and q_{max} denote the lower and upper bounds of vehicle state, and u_{min} and u_{max} denote the lower and upper bounds of control input to the vehicle. The path generated by HSL-RRT* planner can be further smoothed by tracking the path with NMPC algorithm with the vehicle model (3). Thus, the NMPC optimization problem can be formulated as

$$\min_{\mathbf{u}} \mathbf{J}(\mathbf{q}, \mathbf{u}) \quad (7)$$

subject to:

$$\begin{cases} q_{k+1} = f^d(q_k, u_k), & k \in \{1, \cdots, H\} \quad (8a) \\ q_{min} < q_k < q_{max}, & k \in \{1, \cdots, H\} \quad (8b) \\ u_{min} < u_k < u_{max}, & k \in \{1, \cdots, H\} \quad (8c) \end{cases}$$

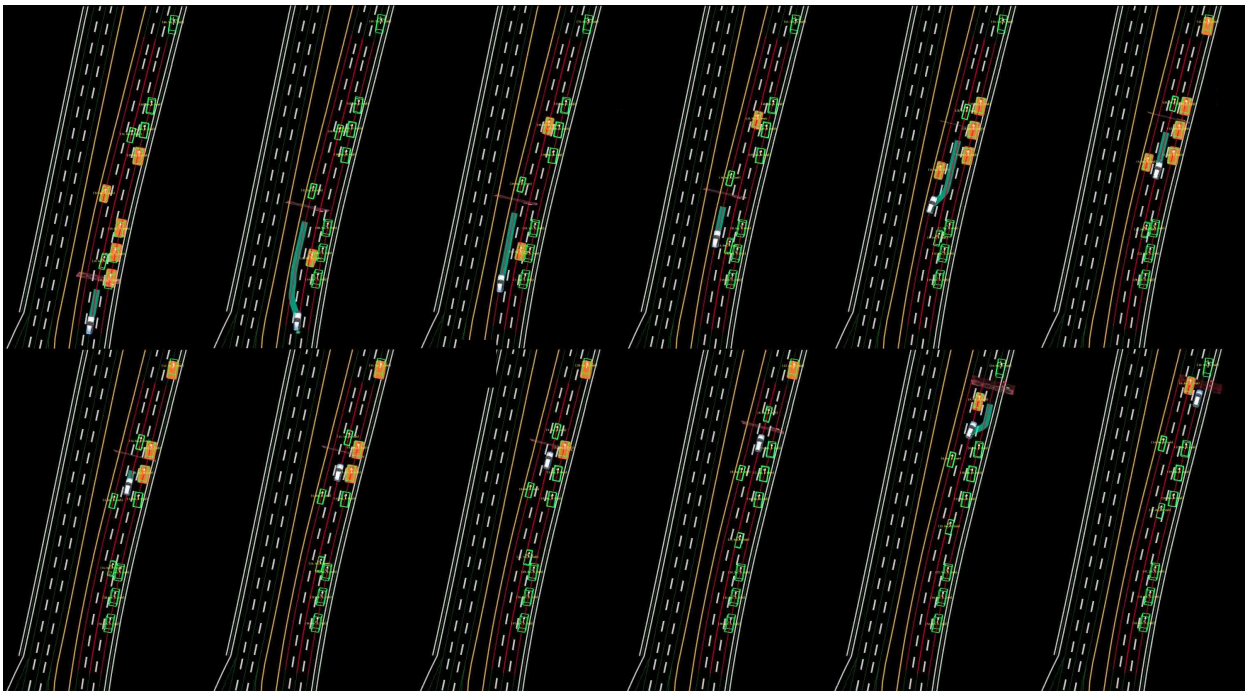


Fig. 6. Pull over for emergency or ride-sharing. Video is provided at `https://youtu.be/I8HMVVUcgMg`

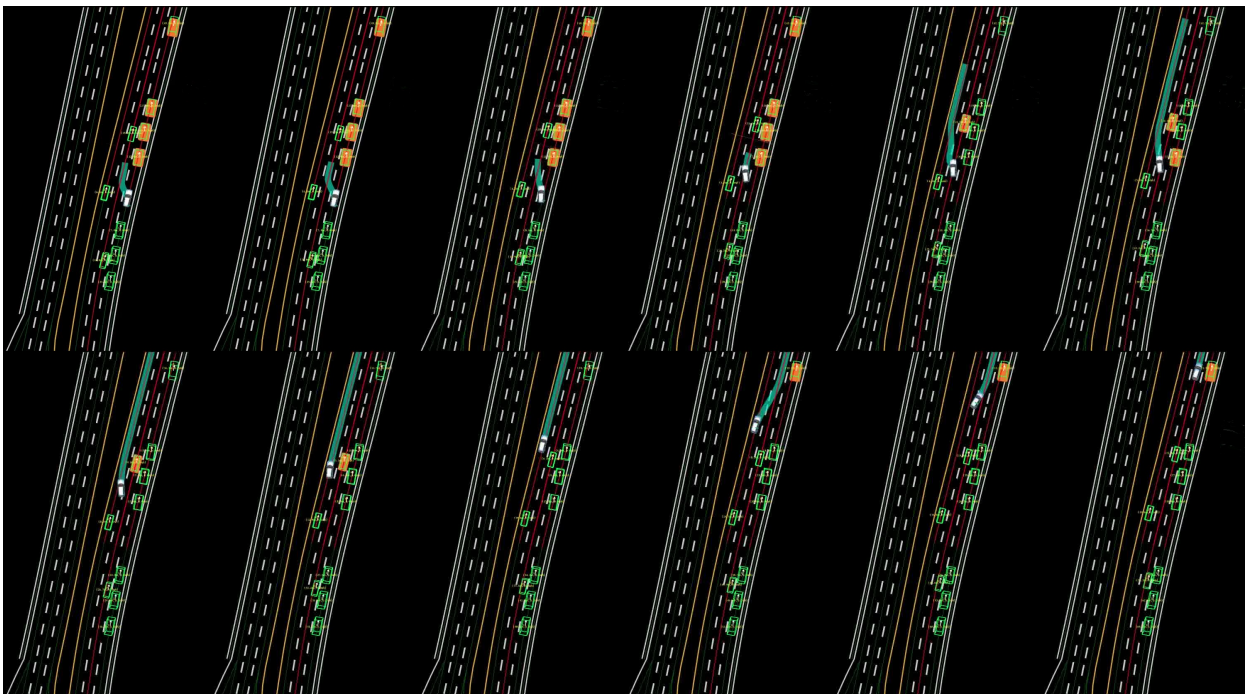


Fig. 7. Merge into traffic flow after pull-over for emergency or ride-sharing. Video is provided at `https://youtu.be/thZBK7Gw82g`

where $\mathbf{J}$ is an objective function to be determined subsequently. Additional constraints (e.g., jerk, energy consumption, etc.) can also be integrated into the optimization problem.

The objective function for the path tracking error is given by quadratic form as

$$\mathbf{J} = \frac{1}{2} e_H^T \mathbf{P} e_H + \frac{1}{2} \sum_{k=1}^{H-1} (e_k^T \mathbf{Q} e_k + u_k^T \mathbf{M} u_k) \qquad (9)$$

where $\mathbf{P} \in \mathbf{R}^{3\times3}$, $\mathbf{Q} \in \mathbf{R}^{3\times3}$, $\mathbf{M} \in \mathbf{R}^{2\times2}$ are the weighting matrices, $e_H = q_H^r - q_H^p$ and $e_k = q_k^r - q_k^p$, and q_k^r and q_k^p are the k^{th} reference and predicted points, respectively. A good

reference of choosing the proper weighting matrices was discussed in [23]. The velocity profile can then be obtained by minimizing the objective function and the final smooth trajectory can be fed to the low-level controller to follow.

V. Results

A. Analysis on Path Generation

1) Completeness and robustness: Compared to a lattice path planner, which heavily depends on grid size to generate a solution, HSL-RRT* performs better in term of completeness and robustness. Fig.4 and Fig.5 show that a lattice path planner easily fails to generate a trajectory for a complex scenario such as navigation through many obstacles, or short lane change. For such scenarios, parameter tuning (change the grid size) is required to get a solution in path planner. On the contrary, HSL-RRT* does not requires such parameter tuning. HSL-RRT* inherits the probabilistic completeness of RRT approach, which could find a solution if there is any.

2) Computational performance: As mentioned above, HSL-RRT* operates in the SL coordinate and provides a boost for the tree exploration. Thus it provides better performance in term of processing time than tradition planners in the Cartesian coordinate. For instance, the processing time for lattice planner depends on the grid size, while that of some other path planning approaches are sensitive to the number of variant/constraints, which cannot guarantee real-time applications. In contrast, the processing time of the proposed path planner can always be limited in a time window. Although an optimal solution is not be found in one single planning cycle, by carrying planning result from the previous cycle to the next cycle, the path solution will gradually converge to the optimal one. As a result, this approach can guarantee that the path planner not only always runs within a limited time window, but also provides an optimal solution.

B. Simulation Results

Our proposed framework is implemented in Baidu Apollo Open Autonomous Driving Platform. In comparison with Apollo motion planner called Expectation Maximization (EM) planner, our framework provide a more robust solution which can handle more difficult scenarios where EM planner fail such as navigate through many obstacles (Fig.4) or short change lane (Fig.5). Two on-road situations are also simulated and compared with EM planner.

Pull over for emergency or ride-sharing: Fig. 6 shows a scenario in which the ego vehicle needs a pullover to the right shoulder (represented in the rightest lane) and stops when an emergency vehicle is approaching fast or ego vehicle needs to pick up a passenger, where the ego vehicle is surrounded by three moving obstacle vehicles in constant speed, and seven static obstacle vehicles are parking in the right shoulder with two free spaces. The ego vehicle first accelerates for the oncoming gap, goes on the left lane in the S-shaped curve, and then decelerates to a target velocity to fit into the gap in the middle lane. After that, it accelerates to make a right lane change into the middle lane and then decelerates to a target velocity until the gap is open enough for pullover in the shoulder. The ego vehicle accelerates and finally decelerates to pull over on the shoulder.

Merge after the pull over: Fig. 7 shows a scenario in which the ego vehicle needs to merge into the two-lane road where there are three moving obstacle vehicles after the emergency has passed or the passenger has been picked up. The ego vehicle first accelerates for the oncoming gap, goes onto the middle lane in the S-shaped curve, and then decelerates to a target velocity for a second left lane change. After entering the left lane, the ego vehicle accelerates to make a right lane change into the middle lane.

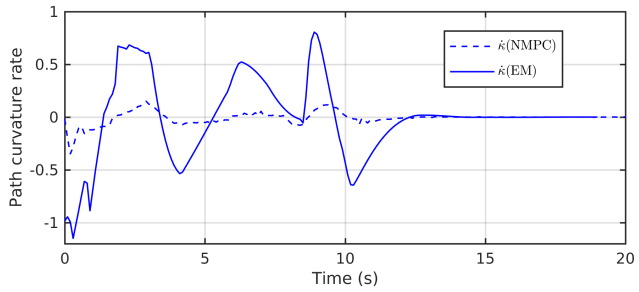


Fig. 8. Example of a trajectory calculated for merging into traffic flow after the emergency pull over. ($\dot{\kappa}$) the path curvature rate of a trajectory. The scenario is depicted in Fig. 7

Fig. 8 shows the derivative of the curvature of both NMPC optimization and the optimization in EM planner. The derivative of the curvature directly influences the comfort of passengers. The comparison shows that, the NMPC optimization has smaller and smoother derivative of curvature, which will make the trajectory easier to follow and improve the comfort of passengers.

For the real-time performance, the whole motion planning framework should provide a trajectory in under 0.1 second. To archive this performance, we set the tree expansion time (as the terminal condition, described in Algorithm1:5) to 0.03 seconds, as discussed in section V-A.2, the algorithm can always provide a solution in a limited time. The radius to get the the nearest neighbors (Algorithm 2:1) is set to the half of reference length. All the results in Figs. 6 and Figs.7 are generated from the same setting of the proposed planner.

VI. Conclusion

In this paper, a real-time motion planning framework is proposed for on-road autonomous driving systems. The framework enhances the robustness, efficiency, and comfort for the autonomous driving vehicle and guarantees the optimality, feasibility and the real-time processing. Simulations show that the proposed framework can robustly handle different dynamic on-road driving scenarios, some of which are challenging even to human drivers. We are currently implementing the approach in a test vehicle to gather real-world validation results.

REFERENCES

[1] 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.

[2] ——, "Path planning for autonomous vehicles in unknown semi-structured environments," *The International Journal of Robotics Research*, vol. 29, no. 5, pp. 485–501, 2010.

[3] J. Ziegler, P. Bender, M. Schreiber, H. Lategahn, T. Strauss, C. Stiller, T. Dang, U. Franke, N. Appenrodt, C. G. Keller, *et al.*, "Making Bertha drive-an autonomous journey on a historic route." *IEEE Intell. Transport. Syst. Mag.*, vol. 6, no. 2, pp. 8–20, 2014.

[4] T. Gu and J. M. Dolan, "On-road motion planning for autonomous vehicles," in *International Conference on Intelligent Robotics and Applications*. Springer, 2012, pp. 588–597.

[5] D. Kogan and R. M. Murray, "Optimization-based navigation for the DARPA grand challenge," 2006.

[6] M. Werling, J. Ziegler, S. Kammel, and S. Thrun, "Optimal trajectory generation for dynamic street scenarios in a Frenét Frame," *2010 IEEE International Conference on Robotics and Automation*, pp. 987–993, 2010.

[7] H. Fan, F. Zhu, C. Liu, L. Zhang, L. Zhuang, D. Li, W. Zhu, J. Hu, H. Li, and Q. Kong, "Baidu apollo EM motion planner," *arXiv preprint arXiv:1807.08048*, 2018.

[8] S. M. LaValle and J. J. Kuffner Jr, "Randomized kinodynamic planning," *The international journal of robotics research*, vol. 20, no. 5, pp. 378–400, 2001.

[9] P. Cheng, E. Frazzoli, and S. LaValle, "Improving the performance of sampling-based motion planning with symmetry-based gap reduction," *IEEE Transactions on Robotics*, vol. 24, no. 2, pp. 488–494, 2008.

[10] L. Jaillet, A. Yershova, S. M. La Valle, and T. Siméon, "Adaptive tuning of the sampling domain for dynamic-domain RRTs," in *Intelligent Robots and Systems, 2005.(IROS 2005). 2005 IEEE/RSJ International Conference on*. IEEE, 2005, pp. 2851–2856.

[11] S. Kiesel, E. Burns, and W. Ruml, "Abstraction-guided sampling for motion planning." in *SoCS*, 2012.

[12] E. Plaku, L. E. Kavraki, and M. Y. Vardi, "Motion planning with dynamics by a synergistic combination of layers of planning," *IEEE Transactions on Robotics*, vol. 26, no. 3, pp. 469–482, 2010.

[13] D. Le and E. Plaku, "Guiding sampling-based tree search for motion planning with dynamics via probabilistic roadmap abstractions," in *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems, Chicago, IL, USA, September 14-18, 2014*, 2014, pp. 212–217.

[14] H. M. Choset, S. Hutchinson, K. M. Lynch, G. Kantor, W. Burgard, L. E. Kavraki, and S. Thrun, *Principles of robot motion: theory, algorithms, and implementation*. MIT press, 2005.

[15] S. M. LaValle, *Planning algorithms*. Cambridge university press, 2006.

[16] S. Karaman and E. Frazzoli, "Incremental sampling-based algorithms for optimal motion planning," *Robotics Science and Systems VI*, vol. 104, p. 2, 2010.

[17] X. Qian, I. Navarro, A. de La Fortelle, and F. Moutarde, "Motion planning for urban autonomous driving using bzier curves and mpc," 11 2016, pp. 826–833.

[18] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968.

[19] A. Stentz, "Optimal and efficient path planning for partially known environments," in *Intelligent Unmanned Ground Vehicles*. Springer, 1997, pp. 203–220.

[20] L. Kavraki, P. Svestka, and M. H. Overmars, *Probabilistic roadmaps for path planning in high-dimensional configuration spaces*. Unknown Publisher, 1994, vol. 1994.

[21] 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.

[22] I. Noreen, A. Khan, and Z. Habib, "Optimal path planning using RRT* based approaches: a survey and future directions," *Int. J. Adv. Comput. Sci. Appl*, vol. 7, pp. 97–107, 2016.

[23] R. M. Murray, "Control and dynamical systems (Lecture 2 - LQR Control)," University Lecture, pp. 2–3, 2006.