Adaptive Pure Pursuit: A Real-Time Path Planner Using Tracking Controllers to Plan Safe and Kinematically Feasible Paths

Bai Li, *Member, IEEE*, Yazhou Wang, Siji Ma, *Member, IEEE*, Xuepeng Bian, Hu Li, Tantan Zhang, Xiaohui Li, and Youmin Zhang, *Fellow, IEEE*

***Abstract*—Path planning is an essential function in an intelligent vehicle, especially when driving in scenarios cluttered by large-scale static obstacles. Traditional path planners often struggle to find a balance among speed, accuracy, and optimality in their solutions. In this article, we introduce an Adaptive Pure Pursuit (APP) planner, which is designed to be fast and near-optimal for autonomous driving in cluttered environments. The APP planner generates feasible paths through a simulated closed-loop tracking control process of a virtual vehicle. If a derived path encounters obstacles, an adaptive refinement step is taken to locally reduce these collisions. Unlike search-based planners that suffer from the "curse of dimensionality" and optimization-based methods that often run slowly, the APP planner operates extremely fast. The high speed stems from the fact that both the virtual controller simulation and the refinement step involve computations with zero degrees of freedom. The proposed APP planner outperforms the prevalent optimization-based and search-based path planners, as shown by comparative simulations.**

***Index Terms*—Path planning, autonomous driving, cluttered environment, pure pursuit control, adaptive pure pursuit (APP).**

Manuscript received 13 June 2023; revised 13 July 2023; accepted 15 July 2023. Date of publication 18 July 2023; date of current version 20 October 2023. This work was supported in part by the National Natural Science Foundation of China under Grants 62103139 and 62003362, in part by the Natural Science Foundation of Hunan Province under Grant 2021JJ40114, and in part by the 2022 Opening Foundation of State Key Laboratory of Management and Control for Complex Systems under Grant E2S9021119. *(Corresponding author: Xiaohui Li.)*

Bai Li is with the State Key Laboratory of Advanced Design and Manufacturing for Vehicle Body, Hunan University, Changsha 410082, China, and also with the College of Mechanical and Vehicle Engineering, Hunan University, Changsha 410082, China (e-mail: libai@zju.edu.cn).

Yazhou Wang, Hu Li, and Tantan Zhang are with the College of Mechanical and Vehicle Engineering, Hunan University, Changsha 410082, China (e-mail: albert@hnu.edu.cn; lihu@hnu.edu.cn; zhangtantan@hnu.edu.cn).

Siji Ma is with the Faculty of Innovation Engineering, Macau University of Science and Technology, Macau 999078, China (e-mail: sijima@ieee.org).

Xuepeng Bian is with the Tencent Automatic Drive Lab, Tencent.Com Inc., Beijing 100193, China (e-mail: waldronbian@tencent.com).

Xiaohui Li is with the College of Intelligence Science, National University of Defense Technology, Changsha 410073, China (e-mail: xiaohui_lee@nudt.edu.cn).

Youmin Zhang is with the Department of Mechanical, Industrial and Aerospace Engineering, Concordia University, Montreal, QC H3G 1M8, Canada (e-mail: ymzhang@encs.concordia.ca).

Real-world experiments were also conducted to validate the APP planner, and its source codes are provided at https://github.com/libai1943/Adaptive_Pure_Pursuit_Planner.

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TIV.2023.3296435.

Digital Object Identifier 10.1109/TIV.2023.3296435

I. Introduction

AUTONOMOUS vehicles (AVs) are responsible for performing complicated tasks, such as exploration [1], rescue [2], and surveillance [3]. These tasks often involve driving in complex environments cluttered with numerous small-sized obstacles. When an AV drives in a highly cluttered environment, path planning is a crucial component that ensures traverse safety [4]. In that case, the challenge of path planning is to quickly identify a way that avoids massive obstacles while ensuring that the path is kinematically feasible.

Broadly speaking, the complexity of path planning arises from the need to simultaneously consider multiple constraints, such as kinematic constraints and collision-avoidance constraints, while exploring the vast solution space for the optimal path [5], [6]. There have been many path planners [7], [8], most of which struggle to balance solution speed, accuracy, and completeness in cluttered environments filled with large-scale obstacles. This article aims to propose a path planner that offers both speed and safety for autonomous driving in highly cluttered environments.

A. Related Works

This subsection reviews the existing path planners suitable for a car-like robot or wheeled robot in a cluttered environment. The involved planners are divided into three categories, namely reaction-based, sampling-based, and refinement-based methods, the details of which are analyzed as follows.

A reaction-based path planner is featured by modifying the ego vehicle's motion according to its collision levels with surrounding obstacles reactively [9]. Typical reaction-based planners include the potential field method (PFM) [10] and vector field histogram (VFH) [11]. In PFM, obstacles are regarded as negative magnets to repel the ego vehicle while the goal is regarded as a positive magnet that attracts the ego vehicle; the motion of the ego vehicle at each step is influenced by both repulsive and attractive forces, leading to a path that balances traverse efficiency and collision safety. PFM runs fast because reactively simulating the adjustment forces is a zero-degree-of-freedom process. Similar with that in PFM, a polar histogram is deployed in VFH to model the repulsive force information. Li et al. [12] utilized the social force model to describe the agent-to-goal and agent-to-obstacle forces when generating coarse trajectories for multiple vehicles. Although reaction-based planners

are remarkably fast, they have two common limitations. First, reaction-based planners focus on the microscopic behaviors of the ego vehicle but ignore the overall path/trajectory smoothness from a macroscopic viewpoint [13]. Second, the paths derived by reaction-based planners do not guarantee to be kinematically feasible, especially when the derived paths are curvy.

Sampling-based planners are widely used in AV navigation [14], [15]. The sampling efficiency is enhanced to tackle cluttered workspaces enriched by obstacles. Qureshi and Ayaz [16] proposed a bi-directional rapidly-exploring random trees (Bi-RRT) method, which samples path primitives from both the initial and goal poses to promote runtime efficiency. Similarly, a multi-stage hybrid A$*$ search algorithm [17] partitions the whole sampling process if there exist bottlenecks narrowed by cluttered obstacles halfway, which decouples a global sampling process into several local ones and thus saves runtime. A generalized Voronoi diagram (GVD) guided RRT is proposed by Chi et al. [18]. Since GVD is insensitive to the density of obstacles in a workspace, a global reference line provided by GVD facilitates the sampling process. Li et al. [19] noticed that massive obstacles would render massive homotopy classes, thus an optimal homotopy class deserves to be greedily selected in a parallel-computation architecture. Similarly, an optimality-enhanced hybrid A$*$ search algorithm [20] is proposed, which continues to sample better paths even after a feasible one is already available. The output of a sampling-based planner is kinematically feasible and collision-free, but the derived path is commonly sub-optimal, jerky, and non-smooth [21]. The solution quality of a sampling-based planner is influenced by the search algorithm associated with the sampler, but a search algorithm typically suffers from the curse of dimensionality [22], thus making a sampling-based planner fail to balance planning quality and speed.

Refinement-based methods are typically employed during the refinement stage of a coarse-to-fine planning process [23], [24]. Numerical optimization is a common approach employed in the refinement stage [3], [25], [26]. However, numerical optimization has two limitations. First, it is computationally heavy, which degrades the time efficiency of a path planner. Second, the output of numerical optimization is deeply influenced by the initial guess because a gradient-based optimizer only finds a local optimum close to the initial guess. While non-optimization methods are also utilized for refinement [27], [28], these methods primarily enhance the smoothness of a coarse path and offer limited capabilities for ensuring path safety. This indicates that a qualified refinement-based planner should be aware of collision risks, in addition to the basic smoothing-path duty. If a smoothed path involved collisions, the conflicting waypoints are repeatedly modified until a simulated tracking controller does not render collisions [29], or a B-spline curve is safe [30]. Particularly regarding how to modify the conflicting waypoints, a collision-based path deformation (CBPD) strategy was proposed by Hu et al. in [31], which pushes each conflicted waypoint in a direction that can reduce the collision level. However, CBPD has a few limitations. First, it works well on a circular robot but is inapplicable to a rectangular car-like robot. Second, CBPD only deforms the waypoints once, thus it does not guarantee the refined path is safe, especially in dealing with complex cases. Therefore, the existing refinement-based methods have room for improvement in time efficiency and solution safety.

As a conclusion of the whole subsection, the prevalent path planners for cluttered environments are imperfect in balancing feasibility, safety, quality, speed, and completeness.

B. Motivations and Contributions

The aforementioned three types of planners, i.e., the reaction-based, sampling-based, and refinement-based planners, have their strengths and limitations. It is natural to grasp their merits and build an integrated planner that quickly finds kinematically feasible and safe paths. Concretely, a sampling-based planner can roughly generate a global path for homotopic guidance; a reaction-based planner can quickly enhance the kinematic feasibility along the global path; if the enhanced path involves minor collisions, then it is polished via a refinement planner. This idea motivates us to propose a three-stage planner with sampling-, reaction-, and refinement-based methods involved.

The proposed path planning methodology is named the Adaptive Pure Pursuit (APP) planner, which has the following two contributions.

First, a reaction-based method is proposed to find a kinematically feasible path in a fast and lightweight way. As opposed to the prevalent reaction-based planners that often involve potentials or forces, we deploy a controller to drive a virtual chassis, thereby ensuring that the tracked path is naturally feasible w.r.t. kinematics. The generation of such a feasible path is fast because driving a virtual chassis under certain control laws is a zero-degree-of-freedom simulation process. Users may even adopt a complicated model to obtain a dynamically feasible path in the framework of our proposed method.

Now that kinematically feasible paths are available with ease, the remaining challenge is how to modify a kinematically feasible path to make it free from collisions. An adaptive refinement strategy is proposed to resolve the conflicts in a kinematically feasible path, which is the second contribution of this study. In the adaptive refinement strategy, locally conflicting segments of a path are nudged according to collision depth information, which brings sufficient flexibility to the APP planner in contrast with rigid-sampling-based planners. The adaptive refinement process is iterative, with increased emphasis on integrating locally conflicting segments, thereby enhancing the smoothness of the whole path and avoiding finding jerky paths.

C. Organization

In the remainder of this article, Section II formulates the concerned path planning problem, and Section III introduces our APP planner. Simulation results are discussed in Section IV. Real-world tests are reported in Section V before conclusions are drawn in Section VI.

II. Problem Statement

This section formulates the path planning problem and provides definitions of necessary variables. As a fundamental assumption, the environment layout and obstacle locations are static and fully available while moving obstacles are not considered. The ego vehicle is designated to reach a goal pose from a specified initial pose, during which the ego vehicle drives with its kinematic capability and avoids collisions with obstacles in the cluttered environment. The concerned path planning scheme is stated as the following optimal control problem (OCP):

$$\begin{aligned}
&\underset{\mathbf{x}(s),\mathbf{u}(s),s_{\max}}{\text{Minimize}} \quad J\left(\mathbf{x}(s),\mathbf{u}(s),s_{\max}\right),\\
&\text{s.t.}\\
&\frac{\mathrm{d}\mathbf{x}(s)}{\mathrm{d}s}=f_{\text{kinematics}}\left(\mathbf{x}(s),\mathbf{u}(s)\right),s\in[0,s_{\max}];\\
&\underline{\mathbf{x}}\le\mathbf{x}(s)\le\overline{\mathbf{x}},\underline{\mathbf{u}}\le\mathbf{u}(s)\le\overline{\mathbf{u}},s\in[0,s_{\max}];\\
&\mathbf{x}(0)=\mathbf{x}_{\text{init}},\mathbf{u}(0)=\mathbf{u}_{\text{init}},\\
&\mathbf{x}(s_{\max})=\mathbf{x}_{\text{goal}},\mathbf{u}(s_{\max})=\mathbf{u}_{\text{goal}};\\
&\Pi\left(\mathbf{x}(s)\right)\cap\Upsilon_{\text{obs}}\subseteq\emptyset,s\in[0,s_{\max}].
\end{aligned}\tag{1}$$

In this problem, variable $s_{\max}$ denotes the length of the to-be-planned path (unknown *a priori*), variable $s\in[0,s_{\max}]$ denotes the shift index, $\mathbf{x}(s)$ stands for state variables, and $\mathbf{u}(s)$ refers to control variables. The cost function *J* is related to $s_{\max}$, $\mathbf{x}(s)$, and $\mathbf{u}(s)$. $\mathrm{d}\mathbf{x}(s)/\mathrm{d}s=f_{\text{kinematics}}(\mathbf{x}(s),\mathbf{u}(s))$ denotes kinematic constraints. $\mathbf{u}(s)$ and $\mathbf{x}(s)$ are bounded by $\underline{\mathbf{u}}$, $\overline{\mathbf{u}}$, $\underline{\mathbf{x}}$, and $\overline{\mathbf{x}}$. Two-point boundary-value constraints are imposed for $\mathbf{u}(s)$ and $\mathbf{x}(s)$ via $\mathbf{x}_{\text{init}}$, $\mathbf{x}_{\text{goal}}$, $\mathbf{u}_{\text{init}}$, and $\mathbf{u}_{\text{goal}}$, respectively. Π is a mapping from the vehicle state to its footprint, i.e., the region that the ego vehicle occupies on the 2-dim ground space. Υ_{obs} denotes the region occupied by environmental obstacles on the 2-dim ground space. Thus $\Pi(\mathbf{x}(s))\cap\Upsilon_{\text{obs}}=\emptyset,\forall s$ denotes the nominal collision-avoidance constraints. The details behind this OCP are presented as follows.

A. Vehicle Kinematic Constraints

For non-extreme driving conditions, the well-known single-track model is capable of describing the kinematic principle of the ego vehicle [32]. Thus, $\mathrm{d}\mathbf{x}(s)/\mathrm{d}s=f_{\text{kinematics}}(\mathbf{x}(s),\mathbf{u}(s))$ is presented as

$$\begin{aligned}
\frac{\mathrm{d}x(s)}{\mathrm{d}s}&=\cos\theta(s),\\
\frac{\mathrm{d}y(s)}{\mathrm{d}s}&=\sin\theta(s),\\
\frac{\mathrm{d}\theta(s)}{\mathrm{d}s}&=\frac{\tan\phi(s)}{\mathrm{L_W}},s\in[0,s_{\max}].
\end{aligned}\tag{2}$$

These equations indicate that $\mathbf{x}(s)\equiv[x(s),y(s),\theta(s)]$ and $\mathbf{u}(s)\equiv\phi(s)$, wherein (x,y) denotes the location of a reference point on the ego vehicle (we choose the midpoint of the rear axle in this study), θ stands for the yaw angle, and ϕ is the steering angle. $\mathrm{L_W}$ denotes the wheelbase of the ego vehicle.

As the sole control variable in this concerned problem, $\phi(s)$ $(s\in[0,s_{\max}])$ determines the overall shape of a planned path. It is required that the steering angle is mechanically bounded:

$$|\phi(s)|\le\phi_{\max},\forall s\in[0,s_{\max}],\tag{3}$$

where $\phi_{\max}$ stands for the maximum allowable steering angle.

B. Collision-Avoidance Constraints

Let us define the four vertexes of the ego vehicle as *A*, *B*, *C*, and *D*. $\Pi(\mathbf{x}(s))$ denotes the rectangle $A(s)B(s)C(s)D(s)$, which should not overlap with any of the static obstacles in the environment. This work assumes that each obstacle is convex. Concave obstacles, if exist, need to be divided into convex ones to meet this assumption. An overlap between any obstacle and the ego vehicle's footprint is prohibited at any $s\in[0,s_{\max}]$. An analytical method to model inter-polygon collision-avoidance constraints is the triangle-area criterion [33].

C. Cost Function

A cost function is deployed to encourage finding smooth and short paths:

$$J=s_{\max}+\mathrm{w}_{\text{smoothness}}\cdot\int_{\varsigma=0}^{s_{\max}}\left(\phi(\varsigma)\right)^2\mathrm{d}\varsigma,\tag{4}$$

wherein $\mathrm{w}_{\text{smoothness}}>0$ is a weighting parameter.

III. Principle of APP Planner

This section introduces the proposed APP path planner. We present the overall framework before entering into the technical details of each module.

A. Overall Framework

The overall framework is presented in Algorithm 1. Most procedures of the proposed planner lie in a *while* loop, where an intermediate path is iteratively updated until it becomes totally collision-free.

Concretely, each iteration of the while loop ends with an intermediate path, which would be further checked for collisions in the next iteration. If an intermediate path involves collisions, then it is divided into local segments where conflicting waypoints are clustered. Herein, a conflicting waypoint refers to an invalid vehicular pose along the intermediate path that makes the ego vehicle collide with surrounding obstacles. After the local segments are defined, they should be polished sequentially. Alternatively, the local segments can be processed in a faster way via parallel computing because the segments are mutually independent. When all local segments are polished, a global path is generated via simulating a zero-degree-of-freedom tracking process, which is the end of the current iteration. If the derived global path still involves collisions, it deserves to be further processed in the next iteration, otherwise the algorithm should break out of the while loop.

The core contribution of Algorithm 1 is the local refinement function PolishLocalSegment(), where a local path is sampled by simulating the tracking process of a virtual vehicle chassis.

Algorithm 1: Overall Framework of Proposed APP Planner.

Function $[is\_completed, smooth\_path_{\text{global}}] \leftarrow \text{PlanPath}(map, pose)$

1. $carrot\_path_{\text{global}} \leftarrow \text{SearchA*Path}(map, pose)$;
2. Initialize $is\_completed \leftarrow 0, iter \leftarrow 0, buffer_{\text{left}} \leftarrow \text{bf}^{\text{init}}_{\text{left}}, buffer_{\text{right}} \leftarrow \text{bf}^{\text{init}}_{\text{right}}$;
3. **while** $iter <$ outer_iter$_{\text{max}}$, **do**
4. $[smooth\_path_{\text{global}}, carrot\_path_{\text{global}}] \leftarrow \text{SamplePurePursuitPath}(carrot\_path_{\text{global}}, pose)$;
5. $conflict\_id \leftarrow \text{IdentifyConflictingPoints}(smooth\_path_{\text{global}}, map)$;
6. $conflict\_segms \leftarrow \text{CreateConflictingSegments}(conflict\_id, buffer_{\text{left}}, buffer_{\text{right}})$;
7. **if** $conflict\_segms \subseteq \emptyset$, **then**
8. $is\_completed \leftarrow 1$;
9. **return**;
10. **end if**
11. Initialize $polished\_segms \leftarrow \emptyset$;
12. **for each** $segm \in conflict\_segms$,**do**
13. $carrot\_path_{\text{local}} \leftarrow \text{PolishLocalSegment}(map, segm, smooth\_path_{\text{global}}, carrot\_path_{\text{global}}, conflict\_id)$;
14. $polished\_segms \leftarrow polished\_segms \cup carrot\_path_{\text{local}}$;
15. **end for**
16. $carrot\_path_{\text{global}} \leftarrow \text{StitchPolishedSegments}(carrot\_path_{\text{global}}, polished\_segms, conflict\_segms)$;
17. $buffer_{\text{left}} \leftarrow buffer_{\text{left}} + \Delta\text{bf}_{\text{left}}, buffer_{\text{right}} \leftarrow buffer_{\text{right}} + \Delta\text{bf}_{\text{right}}$;
18. $++iter$;
19. **end while**
20. **return**;

This idea has the merit that a sampled path is naturally feasible w.r.t vehicle kinematics or even dynamics. The sampling process is embedded in an iterative framework so that the sampled path can adjust with sufficient flexibility to avoid collisions with environmental obstacles.

Algorithm 1 maintains two types of paths, namely the carrot path and the smooth path. A carrot path refers to one that is tracked by a pure-pursuit controller. This controller determines the steering angle of our ego vehicle based on the relative position between the vehicle's current pose and a point ahead on the carrot path. This forward point is referred to as the "carrot point", a term inspired by the image of a donkey chasing a carrot that remains just out of reach. The movement of these carrot points creates a path called the carrot path. The smooth path, on the other hand, denotes the trajectory of a fixed reference point on the virtual chassis (typically the movement of the rear axle midpoint) when it tracks the carrot path.

The next few subsections provide detailed principles of the functions that appear in Algorithm 1.

B. Find a Global Route

This subsection outlines the principle of SearchA*Path() in Line 1 of Algorithm 1. The function's input includes obstacle data from the environment (labeled as *map*) and the initial/goal poses (referred to as *pose*). We model the workspace as a map divided into grid cells, each marked as 1 (occupied) or 0 (unoccupied). Following this, the map is dilated by a radius equivalent to the half-width of the ego vehicle. This dilation results in the expansion of the occupied grid cells. Compared with the original map, the dilated map expands the scale of those grids with the status of 1. For navigation, we utilize the A$*$ search algorithm [34] to establish a path on the dilated map from the grid cell of the initial pose to the grid cell containing the goal pose. The derived path is denoted as $carrot\_path_{\text{global}}$, which consists of a sequence of points defined by their x and y coordinates.

C. Sample a Smooth Path by Pure-Pursuit Tracking Control

The principle of SamplePurePursuitPath() , which appears in Line 4 of Algorithm 1, is introduced in this subsection. The function's input includes a carrot path $carrot\_path_{\text{global}}$ and the aforementioned *pose*. A virtual chassis is set up to model the kinematics of the ego vehicle. The virtual chassis is originally placed at the initial pose recorded in *pose*. Thereafter, the virtual chassis begins to move by tracking the carrot path via a pure-pursuit controller, which determines the ego vehicle's front wheel steering angle based on the relative position between the vehicle's current pose and a carrot point along the carrot path [35]. The principle of the basic pure pursuit controller is briefly presented in Appendix.

In simulating the movement of the virtual chassis, the pure-pursuit controller updates the control demand in every $\Delta\text{t}_{\text{sim}}$ seconds. A smooth path is derived if one observes the motion of the virtual chassis until it is close to the goal pose. However, the obtained smooth path is not continuous in the curvature. This is because the continuity of curvature is directly related to the continuity of steering angle, which jumps to a different value at every $\Delta\text{t}_{\text{sim}}$ seconds. To guarantee the obtained smooth path is curvature-continuous, this study additionally requires that the derivative of steering angle is upper bounded by a user-specified parameter $\omega_{\text{max}} > 0$. Specifically, suppose that a previous steering angle command is ϕ_0 and a new steering angle command ϕ_1 is available at $t = \text{t}_1$. We require that the virtual chassis responds during $t \in (\text{t}_1, \text{t}_1 + \Delta\text{t}_{\text{sim}})$ in the following way: the steering

angle changes from ϕ_0 to ϕ_1 continually and quickly. This means that $\phi(t)$ is the solution to the following optimal control problem (OCP), where $\omega_{\max}$ denotes the maximum allowable value of the steering angle derivative.

$$\begin{aligned}\min_{\phi(t)} \int_{t=\mathrm{t}_1}^{\mathrm{t}_1+\Delta\mathrm{t}_{\mathrm{sim}}} (\phi(t)-\phi_1)^2 \mathrm{d}t \\ \text{s.t.,} \phi(\mathrm{t}_1)=\phi_0, \\ \left|\frac{\mathrm{d}\phi(t)}{\mathrm{d}t}\right| \le \omega_{\max}, \\ |\phi(t)| \le \phi_{\max}. \end{aligned} \tag{5}$$

The analytical solution to OCP (5) can be easily found by Pontryagin's Maximum Principle [36]. Notably, $\phi(\mathrm{t}_1+\Delta\mathrm{t}_{\mathrm{sim}})$ is not necessarily equal to ϕ_1, but $\phi(t)$ would try its best to reach ϕ_1 earlier than $t=\mathrm{t}_1+\Delta\mathrm{t}_{\mathrm{sim}}$.

With the obtained $\phi(t)$ and a constant velocity $v(t)\equiv \mathrm{v}_{\mathrm{const}}$, one can simulate the motion of the virtual chassis during $t\in[\mathrm{t}_1,\mathrm{t}_1+\Delta\mathrm{t}_{\mathrm{sim}}]$.The x–y–θ state that the virtual chassis reaches at the moment $t=\mathrm{t}_1+\Delta\mathrm{t}_{\mathrm{sim}}$ is recorded in a waypoint and further stored in a vector. The x–y coordinate of the carrot point that the virtual chassis tracks during $t\in[\mathrm{t}_1,\mathrm{t}_1+\Delta\mathrm{t}_{\mathrm{sim}}]$ is recorded in another vector.

The outputs of SamplePurePursuitPath() are the two vectors mentioned above, which are recorded as $smooth\_path_{\mathrm{global}}$ and $carrot\_path_{\mathrm{global}}$. The waypoint number in $smooth\_path_{\mathrm{global}}$ is identical to that in $carrot\_path_{\mathrm{global}}$. Let us define the waypoint number as N_{path} for future usage.

D. Identify Conflicting Local Segments Along Global Path

This subsection introduces IdentifyConflictingPoints() and CreateConflictingSegments().

The function IdentifyConflictingPoints outputs the indices of invalid waypoints in $smooth\_path_{\mathrm{global}}$. Concretely, the ith waypoint x_i–y_i–θ_i is regarded as invalid if the corresponding footprint involves collisions, i.e., $\Pi([x_i,y_i,\theta_i])\cap\Upsilon_{\mathrm{obs}}\not\subset\emptyset$. Detailed principle of the collision checker is introduced in [37]. The output of this function is called $conflict\_id$.

Invalid indices $conflict\_id$ would be further processed in the function CreateConflictingSegments to form local segments. Concretely, a one-dimensional array Λ is formed, which consists of as many as N_{path} 0-valued elements. As we have mentioned earlier, N_{path} is the count of waypoints in $smooth\_path_{\mathrm{global}}$. All elements in Λ are initially set to 0, and the elements whose indices are recorded in $conflict\_id$ would be called *seeding elements*. For a seeding element indexed as i, we trace backward to $i-1, i-2, \ldots,$ until we find $i-\mathrm{k}$, which satisfies the following criterion:

$$\left(\sum_{j=i-1}^{i-\mathrm{k}} \|(x_{j+1}-x_j, y_{j+1}-y_j)\| \ge buffer_{\mathrm{left}}\right) \vee (i-\mathrm{k}=1). \tag{6}$$

In this context, (x_j,y_j) represents the coordinate of the jth waypoint in $smooth\_path_{\mathrm{global}}$, while $buffer_{\mathrm{left}}$ is the trace-back distance threshold. Specifically, (6) stipulates that the distance between the seeding element and the $(i-\mathrm{k})$th waypoint along the smooth path should exceed $buffer_{\mathrm{left}}$, or alternatively, the $(i-\mathrm{k})$th waypoint is the first element in $smooth\_path_{\mathrm{global}}$. Once k is determined, the elements in Λ at indices $i-1, i-2, \ldots, i-\mathrm{k}$ are set to 1.

A similar forward trace to $i+1, i+2$, etc., is performed until an index $i+\mathrm{m}$ is found that satisfies (7), wherein $buffer_{\mathrm{right}}$ is the trace-forward distance threshold. The elements in Λ at indices $i+1, i+2, \ldots, i+\mathrm{m}$ are set to 1.

$$\left(\sum_{j=i+1}^{i+\mathrm{m}} \|(x_{j-1}-x_j, y_{j-1}-y_j)\| \ge buffer_{\mathrm{right}}\right) \vee (i+\mathrm{m}==N_{\mathrm{path}}). \tag{7}$$

In essence, these operations ensure that the left and right neighboring elements of a seeding element are set to 1, akin to planting a seed that grows in both directions in array Λ. The seed growing intensity is determined by user-specified parameters $buffer_{\mathrm{left}}, buffer_{\mathrm{right}}\ge 0$, which are initialized in Line 2 of Algorithm 1 via user-specified parameters $\mathrm{bf}_{\mathrm{left}}^{\mathrm{init}}$ and $\mathrm{bf}_{\mathrm{right}}^{\mathrm{init}}$. $buffer_{\mathrm{left}}$ and $buffer_{\mathrm{right}}$ would further increase in Line 17 of Algorithm 1, which is introduced in Section III-G.

After all of the seeding elements are processed, we check Λ and discard those 0-value elements, thus dividing the original array into local segments. Each segment contains an ascending sequence of indices. The output of CreateConflictingSegments is denoted as a vector $conflict\_segms$. If $conflict\_segms$ is empty, then it indicates that all of the sampled waypoints along the $smooth\_path_{\mathrm{global}}$ are collision-free, thus the entire planner exits with the current $smooth\_path_{\mathrm{global}}$, which is safe and kinematically feasible (Line 7–10). Particularly in Line 8 of Algorithm 1, $is\_completed$ is a boolean variable that indicates whether the APP planner finds a valid path finally (1 = success, 0 = failure). If $conflict\_segms$ is not empty, then an extra procedure is needed to refine the conflicting local segments, which would be introduced in the next subsection.

E. Refine Local Segments via Adaptive Adjustments

This subsection introduces Lines 11–15 of Algorithm 1, which are about how to refine each local segment in the vector $conflict\_segms$ in a *for* loop. Without loss of generality, we focus on one local segment $segm\in conflict\_segms$.

Local segment refinement is done in PolishLocalSegment(), the pseudo-code of which is listed in Algorithm 2.

In Algorithm 2, Line 1 defines the starting and ending indices, i.e., id_{start} and id_{end}. Line 2 defines the initial and goal poses in the local segment refinement scheme. Lines 3–5 cut variables $smooth\_path_{\mathrm{global}}$, $carrot\_path_{\mathrm{global}}$, and $conflict\_id$ to concentrate on the concerned local segment. A *for* loop, spanning from Lines 6 to 28 of Algorithm 2, is deployed to repeatedly refine the local path segment, up to a maximum of $\mathrm{inner}\_iter_{\max}$ iterations. In each iteration, we assess whether each waypoint along the local smooth path (denoted as $smooth\_path_{\mathrm{local}}$ in Line 4) is conflicting (Line 24). If one waypoint is conflicting (Line 9), then we evaluate the collision rates on both halves of the vehicle footprint (Line 10), that is,

Algorithm 2: Local Segment Refinement.

Function $carrot\_path_{\text{local}} \leftarrow \text{PolishLocalSegment}(map, segm, smooth\_path_{\text{global}}, carrot\_path_{\text{global}}, conflict\_id)$

1. $id_{\text{start}} \leftarrow segm[1], id_{\text{end}} \leftarrow segm[\text{end}]$;
2. $pose_{\text{local}} \leftarrow [smooth\_path_{\text{global}}[id_{\text{start}}], smooth\_path_{\text{global}}[id_{\text{end}}]]$;
3. Initialize $conflict\_id_{\text{local}} \leftarrow conflict\_id[id_{\text{start}} : id_{\text{end}}]$;
4. Initialize $smooth\_path_{\text{local}} \leftarrow smooth\_path_{\text{global}}[id_{\text{start}} : id_{\text{end}}]$;
5. Initialize $carrot\_path_{\text{local}} \leftarrow carrot\_path_{\text{global}}[id_{\text{start}} : id_{\text{end}}]$;
6. **for** $(iter = 1; iter \leq \text{inner\_}iter_{\max}; ++iter)$, **do**
7. **for** $(i = 1; i \leq id_{\text{end}} - id_{\text{start}}; ++i)$, **do**
8. $wp = smooth\_path_{\text{local}}[i]$;
9. **if** $conflict\_id_{\text{local}}[i] == 1$, **then**
10. $[rate_{\text{left}}, rate_{\text{right}}] \leftarrow \text{MeasureCollisionRates}(map, wp)$;
11. $tb \leftarrow |rate_{\text{left}} - rate_{\text{right}}| \times \text{traceback\_}length_{\max}$;
12. $id \leftarrow \text{FindMatchedCarrotPointID}(carrot\_path_{\text{local}}, i, tb)$;
13. **if** $rate_{\text{left}} \geq rate_{\text{right}}$, **then**
14. $carrot\_path_{\text{local}}[id].x \leftarrow carrot\_path_{\text{local}}[id].x + \Delta\text{s} \cdot \sin(wp.\theta)$;
15. $carrot\_path_{\text{local}}[id].y \leftarrow carrot\_path_{\text{local}}[id].y - \Delta\text{s} \cdot \cos(wp.\theta)$;
16. **else**
17. $carrot\_path_{\text{local}}[id].x \leftarrow carrot\_path_{\text{local}}[id].x - \Delta\text{s} \cdot \sin(wp.\theta)$;
18. $carrot\_path_{\text{local}}[id].y \leftarrow carrot\_path_{\text{local}}[id].y + \Delta\text{s} \cdot \cos(wp.\theta)$;
19. **end if**
20. **end if**
21. **end for**
22. $[smooth\_path_{\text{local}}, carrot\_path_{\text{local}}] \leftarrow \text{SamplePurePursuitPath}(carrot\_path_{\text{local}}, pose_{\text{local}})$;
23. $[smooth\_path_{\text{local}}, carrot\_path_{\text{local}}] \leftarrow \text{ResamplePaths}(smooth\_path_{\text{local}}, carrot\_path_{\text{local}})$;
24. $conflict\_id_{\text{local}} \leftarrow \text{IdentifyConflictingPoints}(smooth\_path_{\text{local}}, map)$;
25. **if** $\sum_j conflict\_id_{\text{local}}[j] == 0$, **then**
26. **break**;
27. **end if**
28. **end for**
29. **return**;

$rate_{\text{left}}, rate_{\text{right}} \in [0, 1]$. If the collision rate is higher on the vehicle's left half side ($rate_{\text{left}} \geq rate_{\text{right}}$, Line 13), then the local carrot path $carrot\_path_{\text{local}}$ is nudged rightwards to reduce $rate_{\text{left}}$. Conversely, if $rate_{\text{left}} < rate_{\text{right}}$, then the local carrot path nudged leftwards. The method for determining the nudge depth (Lines 10–19) is a key innovation of this article and will be introduced in more details later. The nudging effects of all conflicting waypoints are integrated into $carrot\_path_{\text{local}}$ before a virtual chassis is used to track this local carrot path in Line 22 of Algorithm 2. Notably, the derived $smooth\_path_{\text{local}}$ and $carrot\_path_{\text{local}}$ are resampled to ensure the number of elements in the local smooth path or local carrot path always equals $(id_{\text{end}} - id_{\text{start}})$. Maintaining a consistent path scale is crucial, as it enables the local carrot path $carrot\_path_{\text{local}}$ to perfectly replace the corresponding segment in the global carrot path later (Line 16 of Algorithm 1). Line 24 of Algorithm 2 examines the resampled local smooth path $smooth\_path_{\text{local}}$ for potential collisions. If no collision is found (Line 25), then the function PolishLocalSegment() exits, otherwise a new iteration is executed until the maximum iteration count $\text{inner\_}iter_{\max}$ is reached.

The principle to define the nudge depth is introduced at the end of this subsection. As depicted in Fig. 1, the function

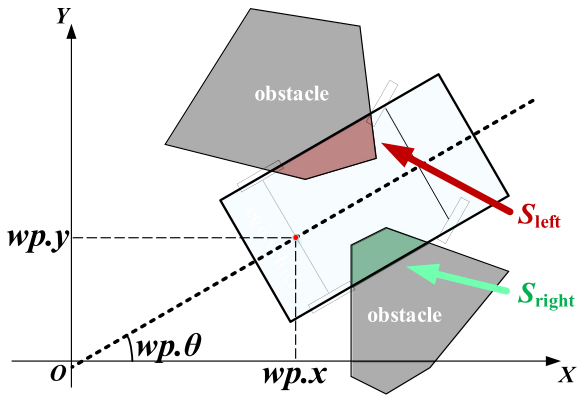


Fig. 1. Schematics on overlap regions between ego vehicle's two haves and environmental obstacles.

$\text{MeasureCollisionRates}(map, wp)$ measures the overlap rates between the two halves of the vehicle body and the obstacles if the vehicle stays at a pose wp. As defined in Line 8 of Algorithm 2, wp denotes the configuration of one specified waypoint along $smooth\_path_{\text{local}}$. The two halves are formed by dividing the ego vehicle along its longitudinal axle. Suppose

that the vehicle length is $\mathrm{L_L}$, vehicle width is $\mathrm{L_B}$, the overlapped area in the left half is S_{left}, and the overlapped area in the right half is S_{right}, then $rate_{\mathrm{left}}$ and $rate_{\mathrm{right}}$ are defined as

$$rate_{\mathrm{left}} = \frac{S_{\mathrm{left}}}{0.5\mathrm{L_L} \times \mathrm{L_B}},$$
$$rate_{\mathrm{right}} = \frac{S_{\mathrm{right}}}{0.5\mathrm{L_L} \times \mathrm{L_B}}. \tag{8}$$

Equation (8) indicates that $rate_{\mathrm{left}}, rate_{\mathrm{right}} \in [0, 1]$.

tb, a variable representing the trace-back distance, is defined in Line 11 of Algorithm 2, where $|rate_{\mathrm{left}} - rate_{\mathrm{right}}|$ ranges from 0 to 1 and $\mathrm{traceback\_}length_{\max} > 0$ is a user-specified parameter defining the maximum trace-back length. The reason why we use $|rate_{\mathrm{left}} - rate_{\mathrm{right}}| \times \mathrm{traceback\_}length_{\max}$ is given as follows.

When one waypoint wp is found conflicting, a natural idea is to adjust the carrot point cp associated with wp. This idea is intuitively correct because a conflict that happens on wp is exactly caused by tracking cp. However, we notice that adjusting cp is empirically ineffective because it is "too late" to avoid a collision. Therefore, we anticipate by adjusting an earlier carrot point (denoted as cp_{new}) that is ahead of cp along the local carrot path $carrot\_path_{\mathrm{local}}$. Suppose that cp is indexed i in the $carrot\_path_{\mathrm{local}}$ while cp_{new} is indexed $id (1 \le id < i)$, id is determined as follows:

$$\arg\min_{id} \left| tb - \sum_{k=id}^{i-1} \|(x_{k+1} - x_k, y_{k+1} - y_k)\| \right|, \tag{9}$$

where (x_k, y_k) denotes the coordinate of the kth point along $carrot\_path_{\mathrm{local}}$. Equation (9) indicates that tb determines the anticipation intensity, that is, the distance between cp_{new} and cp. Recall that tb defined in Line 11 of Algorithm 2 is in proportion to the collision degree. At this point, our idea is that, a more severe conflict degree $|rate_{\mathrm{left}} - rate_{\mathrm{right}}|$ deserves a longer trace-back length tb. The aforementioned procedure defines the function FindMatchedCarrotPointID() in Line 12 of Algorithm 2. Once id is determined, cp_{new} could be identified accordingly. Lines 13–19 present the nudging rule to adjust cp_{new} along the local carrot path. Let us take the branch $rate_{\mathrm{left}} \ge rate_{\mathrm{right}}$ for example. Given that the collision degree is higher on the left side, cp_{new} should be moved rightward. As illustrated in Fig. 2, the nudging direction α is orthogonal to the orientation angle of the ego vehicle at wp, that is, $\alpha = wp.\theta - \pi/2$. Fixing the unit nudging distance to a user- specified parameter $\Delta\mathrm{s} > 0$ yields that

$$cp_{\mathrm{new}}.x \leftarrow cp_{\mathrm{new}}.x + \Delta\mathrm{s} \cdot \cos\alpha,$$
$$cp_{\mathrm{new}}.y \leftarrow cp_{\mathrm{new}}.y + \Delta\mathrm{s} \cdot \sin\alpha, \tag{10}$$

which is in accordance with Lines 14 and 15 if one replaces α with $wp.\theta - \pi/2$. The opposite case in Lines 17 and 18 could be analyzed in the same way.

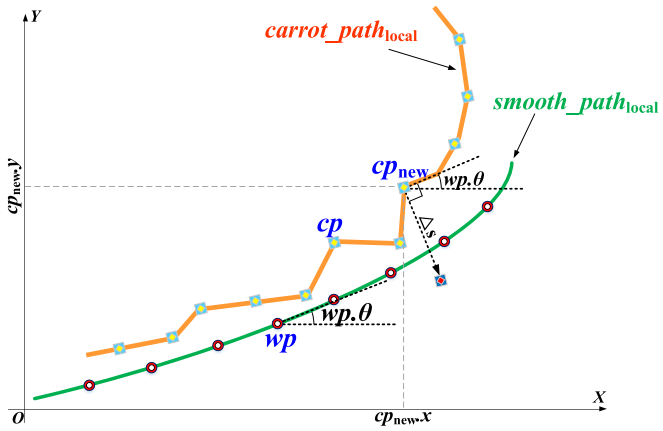


Fig. 2. Schematics on nudging procedure for reshaping local carrot path.

Algorithm 3: Locally Refined Segment Integration.

Function $carrot\_path_{\mathrm{global}} \leftarrow$ StitchPolishedSegments $(carrot\_path_{\mathrm{global}}, polished\_segms, conflict\_segms)$
1. **for** $(i = 1; i \le polished\_segms.\mathrm{size}(); ++i)$, **do**
2. $conflict\_segm \leftarrow conflict\_segms[i]$;
3. $id_{\mathrm{start}} \leftarrow conflict\_segm[1]$, $id_{\mathrm{end}} \leftarrow conflict\_segm[\mathrm{end}]$;
4. $carrot\_path_{\mathrm{global}}[id_{\mathrm{start}} : id_{\mathrm{end}}] = polished\_segms[i]$;
5. **end for**
6. **return**;

The output of PolishLocalSegment() is $carrot\_path_{\mathrm{local}}$, which consists of $(id_{\mathrm{end}} - id_{\mathrm{start}})$ elements. All of the adjusted local carrot paths are gathered in a vector called $polished\_segms$ (Line 14 of Algorithm 2).

F. Integrate Refinements in Local Segments

The function StitchPolishedSegments() is used to update the global carrot path $carrot\_path_{global}$ by integrating the local refinement achievements stored in $polished\_segms$ (Line 16 of Algorithm 1). As stated in the preceding subsection, the scale of each local carrot path is not altered after the polishment, thus StitchPolishedSegments() simply uses each refined local carrot path to replace the corresponding segment along the global carrot path. The pseudo-code is listed in Algorithm 3.

G. Prepare for a New Iteration in Outer Loop

Once $carrot\_path_{global}$ is updated in Line 16 of Algorithm 1, one needs to make preparations for the next iteration. As shown in Line 17 of Algorithm 1, $buffer_{\mathrm{left}}$ and $buffer_{\mathrm{right}}$ are increased by $\Delta\mathrm{bf}_{\mathrm{left}}$ and $\Delta\mathrm{bf}_{\mathrm{right}}$, respectively. Herein, $\Delta\mathrm{bf}_{\mathrm{left}}$ and $\Delta\mathrm{bf}_{\mathrm{right}}$ are user-specified parameters that decide how fast the variables $buffer_{\mathrm{left}}$ and $buffer_{\mathrm{right}}$ should increase during the while-loop iteration. When $buffer_{\mathrm{left}}$ and $buffer_{\mathrm{right}}$ are larger, the conflicting waypoints identified by the function IdentifyConflictingPoints() tend to gather in a smaller number of segments in CreateConflictingSegments(). In Line 18 of Algorithm 1, the index $iter$ is added by 1 to record the iteration cycle. If a maximum cycle is reached (Line 3 of Algorithm 1), the entire planning algorithm exits with a failure flag $is\_completed = 0$.

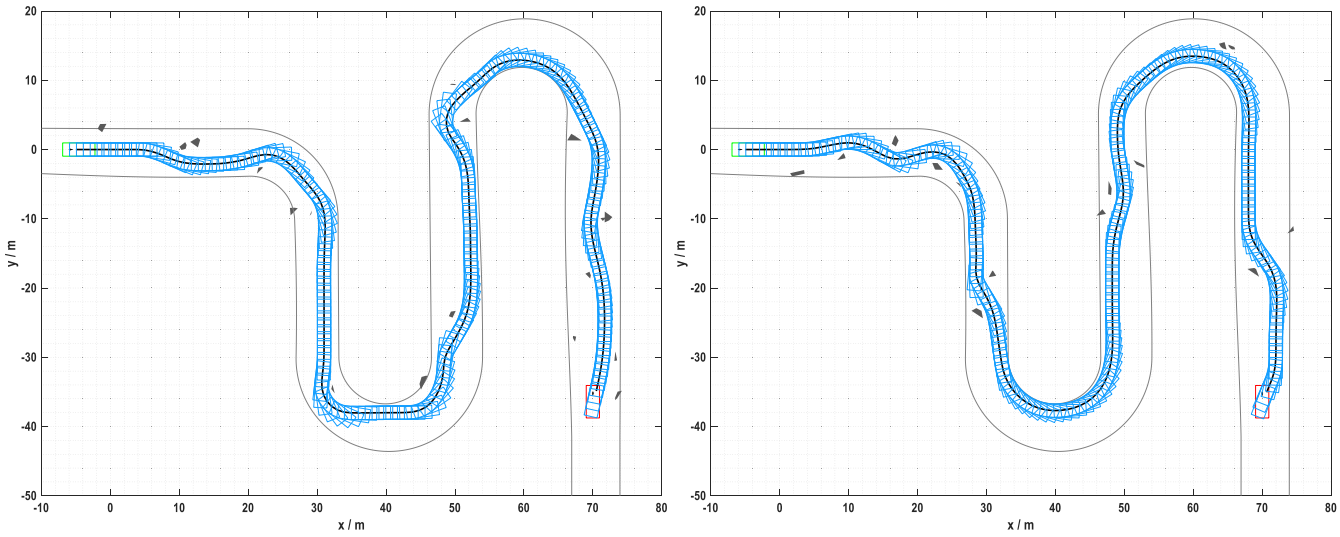


Fig. 3. Planned paths and corresponding footprints of two simulation cases. This figure is seen more clearly if zoomed in.

TABLE I
PARAMETRIC SETTINGS FOR SIMULATIONS

Parameter	Description and unit	Value
L_W	Wheelbase of the ego vehicle (m)	2.800
L_B	Width of the ego vehicle (m)	1.942
L_L	Gross length of the ego vehicle (m)	4.689
L_R	Rear hang length of the ego vehicle (m)	0.929
Φ_{max}	Upper bound of $\lvert\phi(t)\rvert$ (rad)	0.7
ω_{max}	Upper bound of $\lvert d\phi(t)/dt\rvert$ (rad/s)	2.5
outer_iter$_{max}$	Maximum iteration number of the *while* loop in Algorithm 1	10
bf^{init}_{left}, bf^{init}_{right}	Initial values of trace-back and track-forward thresholds used for determining conflicting local segments (m)	4.0, 4.0
Δbf_{left}, Δbf_{right}	Additional values of trace-back and track-forward thresholds (m)	5.0, 5.0
Δt_{sim}	Unit time step in simulating pure pursuit control process (s)	1.0
v_{const}	Constant velocity in simulating tracking control process (m/s)	1.0
inner_iter$_{max}$	Maximum iteration number of the *for* loop in Algorithm 2	200
traceback_length$_{max}$	Maximum trace-back length used for determining the matched carrot point in Line 11 of Algorithm 2 (m)	4.0
Δs	Unit nudging step length (m)	0.1

IV. SIMULATION RESULTS AND DISCUSSIONS

Simulations are conducted to investigate the efficiency of the proposed APP planner.

A. Simulation Setup

Simulations are executed on an i9-9900 CPU that runs at 2×3.10 GHz. Basic parametric settings are listed in Table I. A typical curvy road scenario containing a right-turn and two U-turns is defined, where static obstacles are irregularly placed to clutter the drivable area along the road [38].

B. On the Efficiency of APP Planner

This subsection presents the efficiency of the proposed APP path planner in dealing with cluttered environments. We defined two distinct path-planning scenarios by placing 16 and 20 static polygonal obstacles in a randomized pattern. Fig. 3 visually illustrates the generated paths along with the corresponding vehicle footprints, demonstrating the APP planner's capability to maneuver around environmental barriers. Fig. 4 further validates that the path curvatures in both cases are not only continuous but also within reasonable limits. This indicates that the paths planned by our APP planner can be easily tracked by a low-level controller.

It's worth noting, as Fig. 3 illustrates, that the APP planner is not able to reach a predefined goal pose strictly. This inherent limitation arises because the proposed APP planner indirectly manipulates the smooth path via the carrot path without an explicit strategy to fix the end of the smooth path to a specific pose. One potential solution is to define an adaptive strategy similar to how the planner deals with obstacles. Specifically, a simulated smooth path that ends with an incorrect goal pose would be considered invalid, prompting adjustments to the carrot path. However, we have opted not to incorporate this modification into the APP planner, as it would add complexity to the entire algorithm. More importantly, this is not a serious issue if the proposed planner is utilized for online planning in a receding-horizon manner.

In both of the aforementioned cases, Fig. 5 depicts the progression of the smooth path $smooth\_path_{\text{global}}$ during the iterations within Algorithm 1's while loop, with the path color transitioning from red to purple. This evolution of the smooth path demonstrates the APP planner's ability to concentrate on

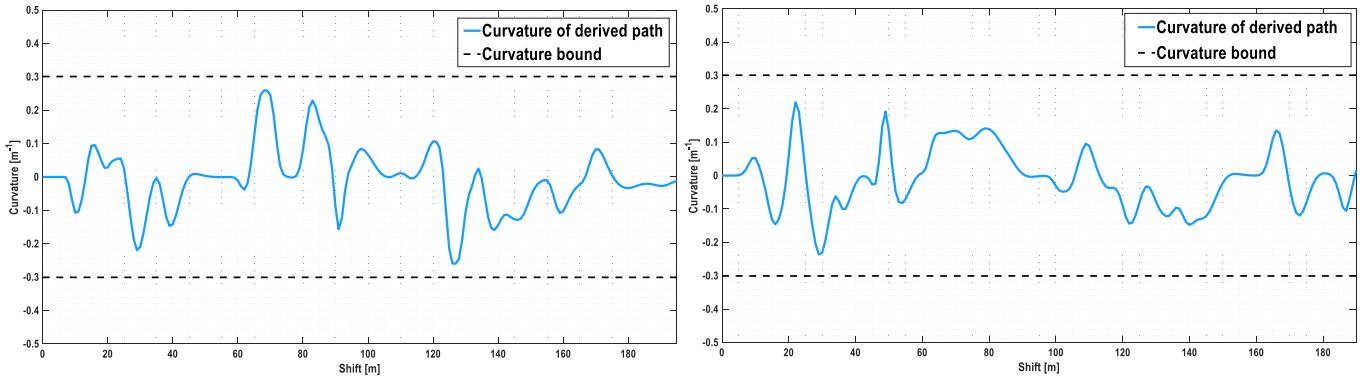


Fig. 4. Curvatures of planned paths in two simulated cases.

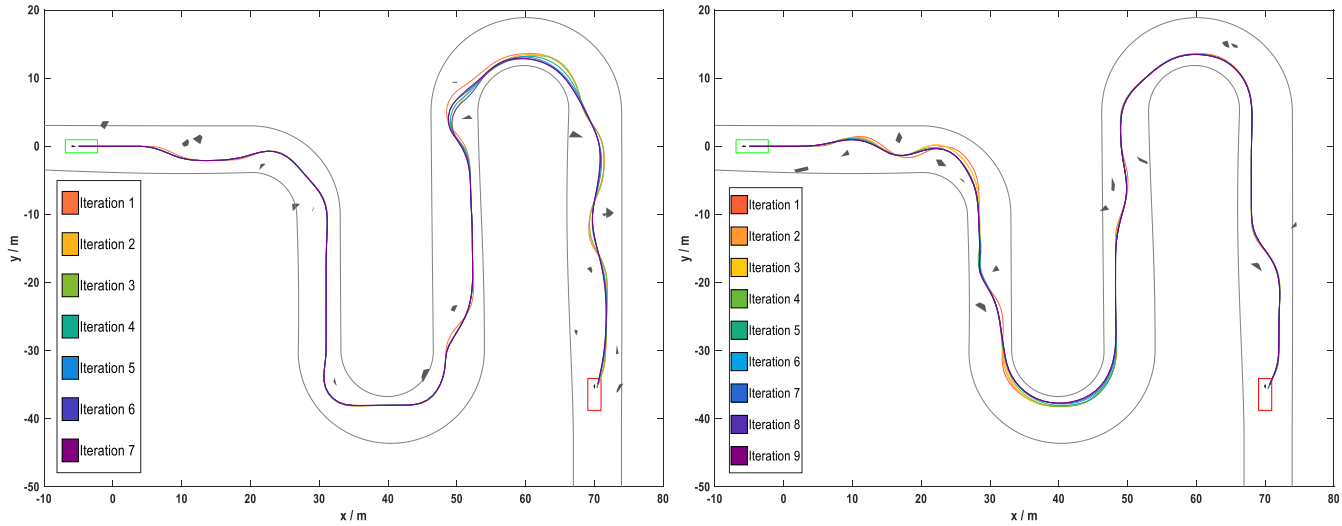


Fig. 5. Evolution of smooth path in using APP planner. The iteration numbers are 7 and 9 in the two simulation cases, respectively. This figure is seen more clearly if zoomed in.

the conflicting local segments and dynamically adjust the waypoints. As opposed to processing the entire path, the change to focusing only on local segments as needed guarantees that the APP planner runs fast. Similar with Fig. 5, Fig. 6 depicts the evolution of the carrot path $carrot\_path_{\text{global}}$ during Algorithm 1's iterative process, aligning with the aforementioned analysis.

C. Comparisons With State-of-the-Art Path Planners

This subsection evaluates the performance of the APP planner by comparing it with some existing path planners. We selected the Dynamic Programming (DP) path planner [39] and the Lightweight Iterative Optimization Method (LIOM) [38], respectively representing the sampling- and refinement-based planners. Concretely, the DP method samples equidistant layers of equidistant nodes and uses quantic polynomials to connect two nodes in adjacent layers. LIOM refines a coarse path via numerically solving an OCP. As a refinement-based planner, LIOM depends heavily on the initial guess quality. To make comparisons fair, the coarse path derived by SearchA*Path() in Section III-B is used to warm-start LIOM.

A benchmark set consisting of 1000 simulation cases was established to ensure the comparisons would be thoroughly made. The number of environmental obstacles in each case was randomly set between 10 and 27 (averaging at 16.77), and these obstacles were distributed randomly within the drivable region of the road scene. Notably, some of the path planning cases in the established benchmark set might be invalid, i.e., no path efficiently connects the initial and goal poses. We deliberately designed the benchmark set like this because a qualified planner should be fail-fast, i.e., terminates quickly when it encounters an invalid path planning case. Table II reports the comparative simulation results. Fig. 7 provides a direct comparison among the planned paths of the three involved planners in a typical benchmark case.

The DP planner runs slowly due to the exhaustive collision checks required by the DP search strategy. The success rate of DP planner is low because it lacks the flexibility to explore every inch of the drivable region. As pointed out by [40], the lack of sampling flexibility is a typical limitation of sampling-based planners with rigidly sampled nodes. Increasing the sampling

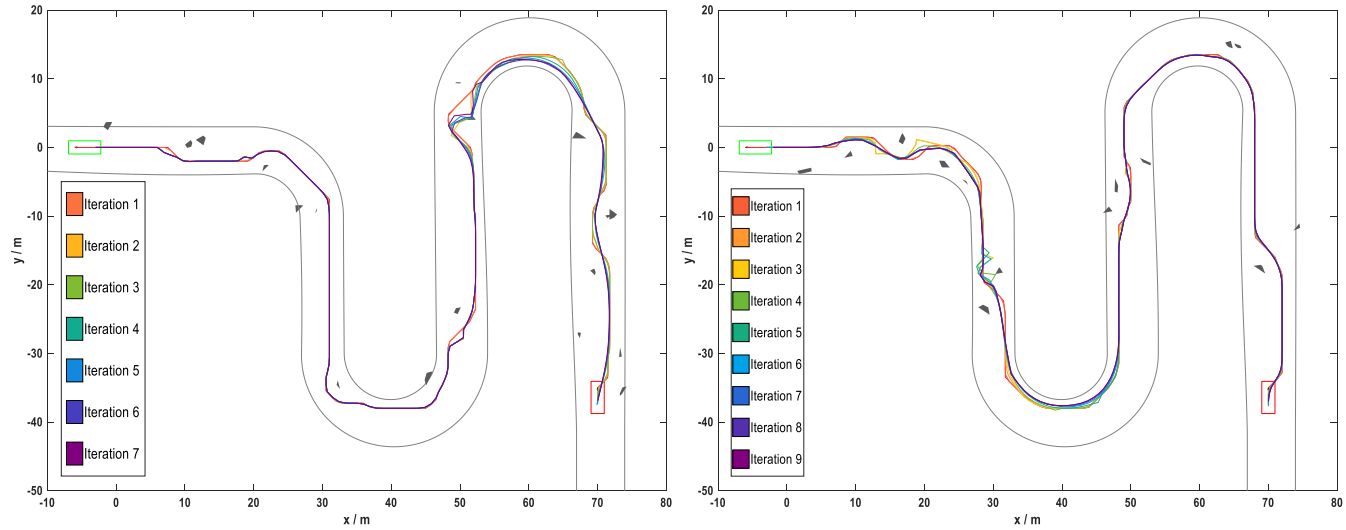


Fig. 6. Evolution of carrot path in using APP planner. This figure is seen more clearly if zoomed in.

TABLE II
COMPARATIVE SIMULATION RESULTS

Planner ID	Success rate	Average CPU runtime (ms)	Average collision check times
APP	89.20%	51.3327	13,738.203
DP	21.10%	413.0506	86,044.101
LIOM	2.90%	976.8490	63,372.333

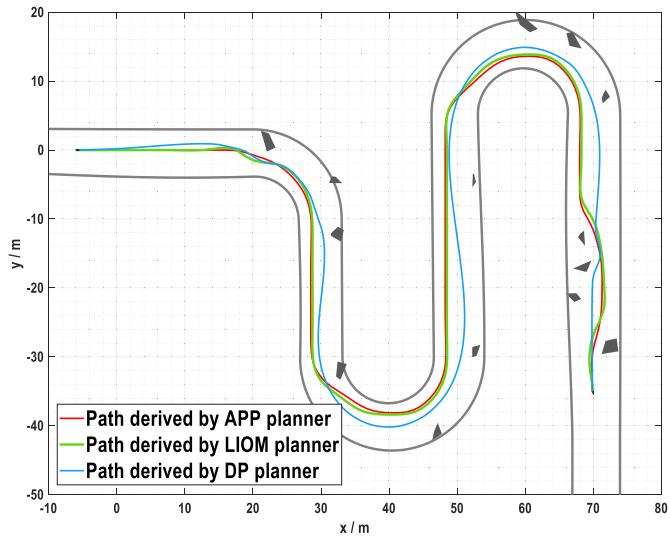


Fig. 7. Typical comparison among APP, DP, and LIOM planners.

resolution to improve the success rate would exacerbate the curse of dimensionality issue, leading to more runtime consumed. A comparison between the DP planner and the proposed APP planner highlights the importance of sampling flexibility.

The average CPU runtime consumed by LIOM is obviously longer than that of the proposed APP planner. The long runtime consumed on safe travel corridor construction and NLP solution is a contributing factor to the limited widespread application of optimization-based planners in the research area of path planning for intelligent vehicles. LIOM failed in most of the cases because the safe travel corridor construction renders buffers around the ego vehicle, which easily blocks the tiny road in our benchmark cases. Therefore, LIOM is not efficacious in dealing with scenarios narrowed by cluttered obstacles. This conclusion is also reflected in Fig. 7, where the path derived by LIOM is more conservative than the one derived by APP.

D. Limitation of APP Planner

A typical limitation of the APP planner is that the planner does not guarantee to reach the goal pose (including the location and orientation) strictly. This is intuitively understandable because sampling in the control space cannot guarantee to reach a specified state. An similar example is the hybrid A$*$ search algorithm, which cannot reach a specific goal pose without the embedded Reeds-Shepp primitive connection step. Thus the APP planner is suitable for planning schemes that do not care much about the planning errors in the goal pose.

V. FIELD TESTS

Indoor experiments were carried out on a 1.700m $\times$ 2.900m track, uniquely characterized by straight lines in the middle and circular arcs at both ends. A car-like robot of dimensions 0.211m $\times$ 0.191m was utilized as the ego vehicle. Six infrared sensors from the NOKOV Motion Capture System were deployed for precise vehicle and obstacle localization. Data from these sensors were compiled on a desktop computer with an AMD Ryzen 5 4600H CPU that runs at 6$\times$3.00 GHz, where the proposed APP planner was also implemented (Fig. 8). Driving commands were wirelessly dispatched to the car-like robot via ZigBee. Proportional-derivative control was adopted for longitudinal

Fig. 8. Field test scenario layout and device setups.

TABLE III
COMPARATIVE CLOSED-LOOP TRACKING CONTROL RESULTS

ω_{max} (rad/s)	Closed-loop lateral tracking error (mm)			
	Maximum	Average	Median	99% percentile
0.60	22.0269	3.6181	2.6645	20.3909
$+\infty$	69.3837	10.4562	3.7599	67.1326

tracking while a pure-pursuit controller was used to mitigate lateral deviations.

A 14-minute field test was executed, during which environmental obstacles were intermittently relocated to impede the robot's path. The planning module was implemented in an event-triggered way, i.e., the path planner would be implemented only when environmental obstacles update. The field test performance is shown at https://www.bilibili.com/video/BV1MP411y7oF/. As we observed, the average planning runtime of the APP planner was 3.909 ms and the success rate is 100% among 21 triggered times (Fig. 9). This indicates the proposed planner is fast, efficacious, and stable. Particularly, the average runtime to sample a global smooth path in the while loop was 0.180ms, which indicates that the proposed planner archives kinematic feasibility in real time.

Recall that the cost function (4) contains a weighted term about path smoothness. The APP planner cannot manipulate the path smoothness by changing the parameter $w_{\text{smoothness}}$. Instead, the APP planner determines the path smoothness by making changes in the function $\mathsf{SamplePurePursuitPath}()$. Concretely, one may set a lower $\omega_{\max}$ to make the virtual chassis steer the front wheels in a lower speed, thus improving the smoothness of the tracked path $smooth\_path_{\text{global}}$. With the purpose of investigating how path smoothness would influence the tracking performance, we conducted tests under different settings of $\omega_{\max}$. Results are reported in Table III and Fig. 10. Setting $\omega_{\max}$ to infinity means that the virtual chassis is allowed to steer its front wheels discontinuously. Tracking with such a virtual chassis makes our modified tracking controller regrades to the conventional pure pursuit controller. As Table III indicates, the conventional pure pursuit controller caused more tracking errors and inevitably renders curvature-discontinuous paths, which are hard to track (Fig. 10(a)). This indicates that our proposal in Section III-C makes sense. Setting $\omega_{\max}$ to a small value would improve the smoothness of the planned path and reduce the tracking error (Fig. 10(b)).

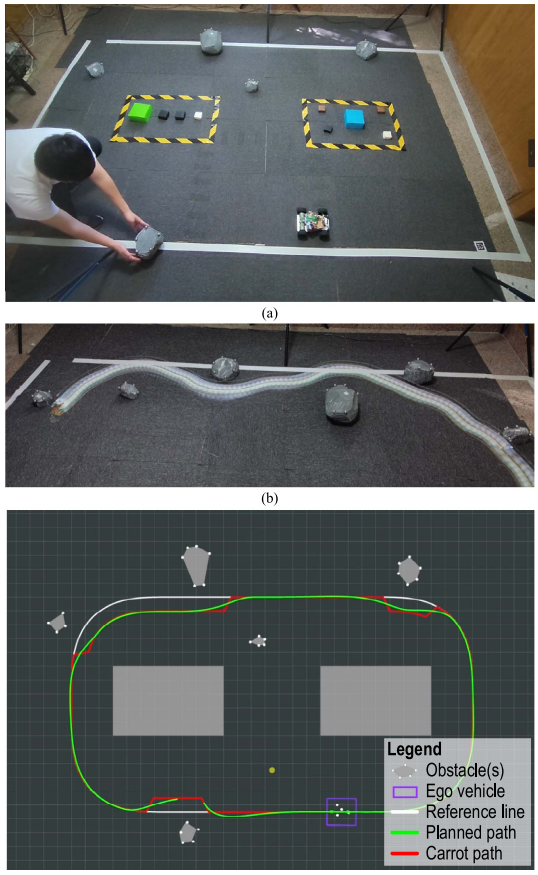
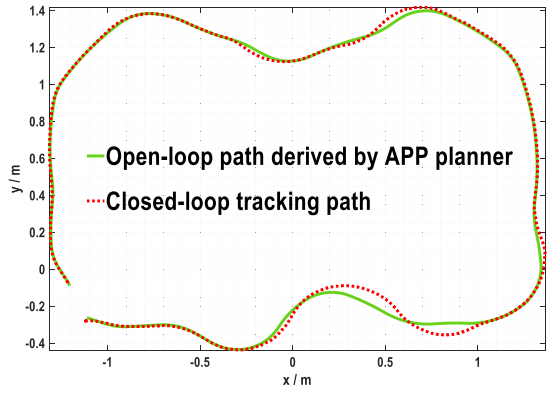


Fig. 9. Typical experimental operations and results: (a) Relocation of an obstacle during the real-time tracking process; (b) evasive maneuvers of ego vehicle captured via time-lapse photography; (c) environmental layout and technical details of APP planner visualized in ROS rviz interface.

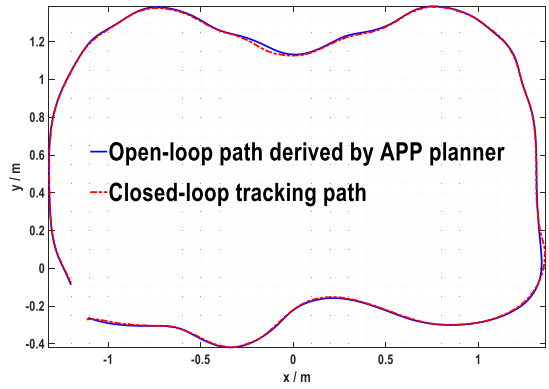
VI. CONCLUSION

This article has proposed a real-time path planning method, named Adaptive Pure Pursuit (APP) planner, for autonomous driving in cluttered environments. Efficiency of the proposed planner has been validated through comparative simulations and field tests.

Our contribution in this article goes beyond the mere formulation of an algorithm; it introduces a fresh paradigm to tackle path planning problems. Traditionally, path planning faced challenges in striking a balance between vehicle kinematic constraints and collision-avoidance constraints. In contrast to widely-used refinement-based planners like LIOM, which prioritize collision-avoidance constraints while attempting to regain kinematic feasibility, our proposed APP planner shows a paradigm shift. It effortlessly satisfies vehicle kinematic constraints, while collision avoidance is achieved adaptively via a series of trials and errors. When compared with sampling-based planners that generate kinematically feasible path candidates/primitives before sorting or searching, the APP planner



(a)



(b)

Fig. 10. Comparative open-loop and closed-loop tracking paths under different settings of $\omega_{\max}$:(a) $\omega_{\max} = +\infty$; (b) $\omega_{\max} = 0.6\text{rad/s}$.

stands out. The flexibility during the sampling process and the local focus on conflict resolution provide the APP planner with a distinct advantage. The proposed paradigm is generic because it accommodates a variety of vehicle kinematic/dynamic models and tracking controllers other than the pure pursuit controller considered in this work. If the ego vehicle drives at a high speed, then the vehicle dynamics should be considered when modeling the virtual chassis. Our proposed APP planner supports the usage of vehicle dynamic models.

Further advancements in this path-planning paradigm are needed to incorporate the ability to plan backward maneuvers. A theoretical convergence analysis is needed to guide the optimal parametric settings in PolishLocalSegment(). A limitation of the APP planner is that it cannot guarantee finding paths with strict satisfaction of goal-pose constraints. Future work is needed to improve the APP planner at that point.

APPENDIX

This section provides the basic principle of pure pursuit controller. As depicted in Fig. 11, the ego vehicle is currently staying at point P and tracking the carrot path. A look-ahead carrot point Q along the carrot path is determined by a distance $\mathrm{L_{fc}}$ ahead of point P. Once the carrot point Q is determined, one can derive the steering angle profile $\phi(t)$ so that the ego vehicle would track point Q in the next $\Delta \mathrm{t_{sim}}$ seconds.

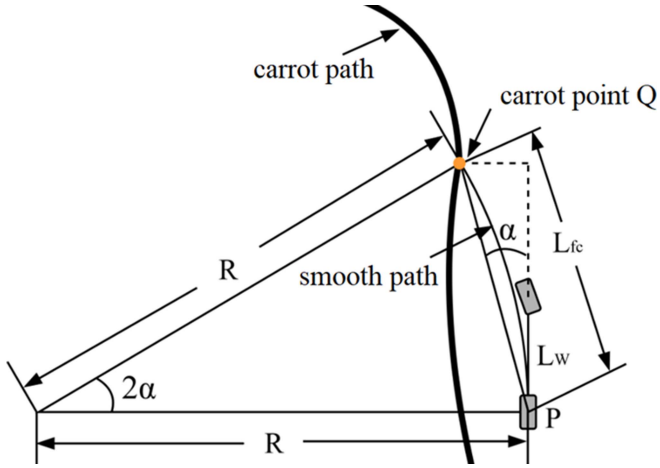


Fig. 11. Schematic of the pure pursuit tracking controller.

The carrot point Q is determined by exploring along the carrot path for a qualified point such that the distance between that point and P is $\mathrm{L_{fc}}$, which is a user-specified parameter that sets the look-ahead distance. The pure pursuit controller expects that the ego vehicle would reach Q with a circular path segment. Due to the geometrics presented in Fig. 11, the radius of the aforementioned circular arc R satisfies

$$\frac{\mathrm{L_{fc}}}{\sin(2\alpha)} = \frac{\mathrm{R}}{\sin(\frac{\pi}{2} - \alpha)}, \tag{A1}$$

which yields that

$$\frac{\mathrm{L_{fc}}}{2\sin(\alpha)\cdot\cos(\alpha)} = \frac{\mathrm{R}}{\cos(\alpha)}. \tag{A2}$$

With an assumption that $\cos(\alpha) \neq 0$, one has

$$\mathrm{R} = \frac{\mathrm{L_{fc}}}{2\sin(\alpha)}, \tag{A3}$$

and thus

$$\phi \equiv \arctan\left(\frac{\mathrm{L_W}}{\mathrm{R}}\right) = \arctan\left(\frac{2\mathrm{L_W}\cdot\sin\alpha}{\mathrm{L_{fc}}}\right). \tag{A4}$$

The ego vehicle would track Q for $\Delta \mathrm{t_{sim}}$ seconds, which indicates the steering angle is set constant during that period. The longitudinal velocity is a user-specified constant value then.

ACKNOWLEDGMENT

The authors are grateful to Yakun Ouyang, Shiqi Tang, Tianxing Yang, Xiaoyan Peng, Xinwei Wang, Chen Li, Zhe Luo, Zhou Liu, Kun Li, Yi Liu, Yong Fang, and Ping Shi for their support in this study.

REFERENCES

[1] J. R. Sánchez-Ibánez, C. J. Pérez-del-Pulgar, M. Azkarate, L. Gerdes, and A. García-Cerezo, "Dynamic path planning for reconfigurable rovers using a multi-layered grid," *Eng. Appl. Artif. Intell.*, vol. 86, pp. 32–42, 2019.

[2] J. S. Chou, M. Y. Cheng, Y. M. Hsieh, I. T. Yang, and H. T. Hsu, "Optimal path planning in real time for dynamic building fire rescue operations using wireless sensors and visual guidance," *Automat. Construction*, vol. 99, pp. 1–17, 2019.

[3] T. Wang, P. Huang, and G. Dong, "Modeling and path planning for persistent surveillance by unmanned ground vehicle," *IEEE Trans. Automat. Sci. Eng.*, vol. 18, no. 4, pp. 1615–1625, Oct. 2021.
[4] Y. Huang, S. Z. Yong, and Y. Chen, "Stability control of autonomous ground vehicles using control-dependent barrier functions," *IEEE Trans. Intell. Veh.*, vol. 6, no. 4, pp. 699–710, Dec. 2021.
[5] L. Schäfer, S. Manzinger, and M. Althoff, "Computation of solution spaces for optimization-based trajectory planning," *IEEE Trans. Intell. Veh.*, vol. 8, no. 1, pp. 216–231, Jan. 2023.
[6] B. Li et al., "Optimization-based trajectory planning for autonomous parking with irregularly placed obstacles: A lightweight iterative framework," *IEEE Trans. Intell. Transp. Syst.*, vol. 23, no. 8, pp. 11970–11981, Aug. 2022.
[7] B. Li et al., "Online competition of trajectory planning for automated parking: Benchmarks, achievements, learned lessons, and future perspectives," *IEEE Trans. Intell. Veh.*, vol. 8, no. 1, pp. 16–21, Jan. 2023.
[8] L. Chen et al., "Milestones in autonomous driving and intelligent vehicles: Survey of surveys," *IEEE Trans. Intell. Veh.*, vol. 8, no. 2, pp. 1046–1056, Feb. 2023.
[9] J. Minguez, L. Montano, and J. Santos-Victor, "Reactive navigation for non-holonomic robots using the ego-kinematic space," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2002, vol. 3, pp. 3074–3080.
[10] O. Khatib, "Real-time obstacle avoidance for manipulators and mobile robots," in *Proc. IEEE Int. Conf. Robot. Automat.*, 1985, pp. 500–505.
[11] J. Borenstein and Y. Koren, "Real-time obstacle avoidance for fast mobile robots in cluttered environments," in *Proc. IEEE Int. Conf. Robot. Automat.*, 1990, vol. 1, pp. 572–577.
[12] B. Li et al., "Sharing traffic priorities via cyber–physical–social intelligence: A lane-free autonomous intersection management method in Metaverse," *IEEE Trans. Syst., Man, Cybern.: Syst.*, vol. 53, no. 4, pp. 2025–2036, Apr. 2023.
[13] H. Li, W. Liu, C. Yang, W. Wang, T. Qie, and C. Xiang, "An optimization-based path planning approach for autonomous vehicles using the DynEFWA-artificial potential field," *IEEE Trans. Intell. Veh.*, vol. 7, no. 2, pp. 263–272, Jun. 2022.
[14] J. Wang, T. Li, B. Li, and Q. H. Meng, "GMR-RRT∗: Sampling-based path planning using Gaussian mixture regression," *IEEE Trans. Intell. Veh.*, vol. 7, no. 3, pp. 690–700, Sep. 2022.
[15] H. Ma, F. Meng, C. Ye, J. Wang, and M. Q. H. Meng, "Bi-risk-RRT based efficient motion planning for autonomous ground vehicles," *IEEE Trans. Intell. Veh.*, vol. 7, no. 3, pp. 722–733, Sep. 2022.
[16] A. H. Qureshi and Y. Ayaz, "Intelligent bidirectional rapidly-exploring random trees for optimal motion planning in complex cluttered environments," *Robot. Auton. Syst.*, vol. 68, pp. 1–11, 2015.
[17] W. Sheng, B. Li, and X. Zhong, "Autonomous parking trajectory planning with tiny passages: A combination of multistage hybrid A-star algorithm and numerical optimal control," *IEEE Access*, vol. 9, pp. 102801–102810, 2021.
[18] W. Chi, Z. Ding, J. Wang, G. Chen, and L. Sun, "A generalized voronoi diagram-based efficient heuristic path planning method for RRTs in mobile robots," *IEEE Trans. Ind. Electron.*, vol. 69, no. 5, pp. 4926–4937, May 2022.
[19] B. Li, Z. Yin, Y. Ouyang, Y. Zhang, X. Zhong, and S. Tang, "Online trajectory replanning for sudden environmental changes during automated parking: A parallel stitching method," *IEEE Trans. Intell. Veh.*, vol. 7, no. 3, pp. 748–757, Sep. 2022.
[20] B. Li, Y. Ouyang, X. Li, D. Cao, T. Zhang, and Y. Wang, "Mixed-integer and conditional trajectory planning for an autonomous mining truck in loading/dumping scenarios: A global optimization approach," *IEEE Trans. Intell. Veh.*, vol. 8, no. 2, pp. 1512–1522, Feb. 2023.
[21] X. Zhang, Y. Jiang, Y. Lu, and X. Xu, "Receding-horizon reinforcement learning approach for kinodynamic motion planning of autonomous vehicles," *IEEE Trans. Intell. Veh.*, vol. 7, no. 3, pp. 556–568, Sep. 2022.
[22] B. Li, L. Li, T. Acarman, Z. Shao, and M. Yue, "Optimization-based maneuver planning for a tractor-trailer vehicle in a curvy tunnel: A weak reliance on sampling and search," *IEEE Robot. Automat. Lett.*, vol. 7, no. 2, pp. 706–713, Apr. 2022.
[23] K. Bergman, O. Ljungqvist, and D. Axehill, "Improved path planning by tightly combining lattice-based path planning and optimal control," *IEEE Trans. Intell. Veh.*, vol. 6, no. 1, pp. 57–66, Mar. 2021.
[24] B. Li, Y. M. Zhang, and Z. Shao, "Spatio-temporal decomposition: A knowledge-based initialization strategy for parallel parking motion optimization," *Knowl.-Based Syst.*, vol. 107, pp. 179–196, 2016.
[25] P. Scheffe, T. M. Henneken, M. Kloock, and B. Alrifaee, "Sequential convex programming methods for real-time optimal trajectory planning in autonomous vehicle racing," *IEEE Trans. Intell. Veh.*, vol. 8, no. 1, pp. 661–672, Jan. 2023.
[26] B. Li et al., "On-road trajectory planning with spatio-temporal RRT∗ and always-feasible quadratic program," in *Proc. IEEE 16th Int. Conf. Automat. Sci. Eng.*, 2020, pp. 942–947.
[27] S. Zhang, Z. Jian, X. Deng, S. Chen, Z. Nan, and N. Zheng, "Hierarchical motion planning for autonomous driving in large-scale complex scenarios," *IEEE Trans. Intell. Transp. Syst.*, vol. 23, no. 8, pp. 13291–13305, Aug. 2022.
[28] C. C. Tsai, H. C. Huang, and C. K. Chan, "Parallel elite genetic algorithm and its application to global path planning for autonomous robot navigation," *IEEE Trans. Ind. Electron.*, vol. 58, no. 10, pp. 4813–4821, Oct. 2011.
[29] X. Zhong, J. Tian, H. Hu, and X. Peng, "Hybrid path planning based on safe A∗ algorithm and adaptive window approach for mobile robot in large-scale dynamic environment," *J. Intell. Robot. Syst.*, vol. 99, pp. 65–77, 2020.
[30] C. Jiang et al., "R2-RRT∗: Reliability-based robust mission planning of off-road autonomous ground vehicle under uncertain terrain environment," *IEEE Trans. Automat. Sci. Eng.*, vol. 19, no. 2, pp. 1030–1046, Apr. 2022.
[31] B. Hu, Z. Cao, and M. Zhou, "An efficient RRT-based framework for planning short and smooth wheeled robot motion under kinodynamic constraints," *IEEE Trans. Ind. Electron.*, vol. 68, no. 4, pp. 3292–3302, Apr. 2021.
[32] X. Zhou, Z. Wang, H. Shen, and J. Wang, "Robust adaptive path-tracking control of autonomous ground vehicles with considerations of steering system backlash," *IEEE Trans. Intell. Veh.*, vol. 7, no. 2, pp. 315–325, Jun. 2022.
[33] B. Li and Z. Shao, "A unified motion planning method for parking an autonomous vehicle in the presence of irregularly placed obstacles," *Knowl.-Based Syst.*, vol. 86, pp. 11–20, 2015.
[34] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE Trans. Syst. Sci. Cybern.*, vol. 4, no. 2, pp. 100–107, Jul. 1968.
[35] R. C. Coulter, *Implementation of the Pure Pursuit Path Tracking Algorithm*. Pittsburgh, PA, USA: Carnegie-Mellon Univ. Pittsburgh PA Robotics Inst., 1992.
[36] R. F. Hartl, S. P. Sethi, and R. G. Vickson, "A survey of the maximum principles for optimal control problems with state constraints," *SIAM Rev.*, vol. 37, no. 2, pp. 181–218, 1995.
[37] J. Ziegler and C. Stiller, "Fast collision checking for intelligent vehicle motion planning," in *Proc. IEEE Intell. Veh. Symp.*, 2010, pp. 518–522.
[38] B. Li, Y. Ouyang, L. Li, and Y. Zhang, "Autonomous driving on curvy roads without reliance on frenet frame: A Cartesian-based trajectory planning method," *IEEE Trans. Intell. Transp. Syst.*, vol. 23, no. 9, pp. 15729–15741, Sep. 2022.
[39] W. Xu, J. Pan, J. Wei, and J. M. Dolan, "Motion planning under uncertainty for on-road autonomous driving," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2014, pp. 2507–2512.
[40] D. Le, Z. Liu, J. Jin, K. Zhang, and B. Zhang, "Historical improvement optimal motion planning with model predictive trajectory optimization for on-road autonomous vehicle," in *Proc. IEEE IECON 45th Annu. Conf. Ind. Electron. Soc.*, 2019, pp. 5223–5230.

Bai Li (Member, IEEE) received the B.S. degree from the School of Advanced Engineering, Beihang University, Beijing, China, in 2013, and the Ph.D. degree from the College of Control Science and Engineering, Zhejiang University, Hangzhou, China, in 2018. From 2016 to 2017, he visited the Department of Civil and Environmental Engineering, University of Michigan, Ann Arbor, MI, USA, as a joint training Ph.D. Student. He is currently an Associate Professor with the College of Mechanical and Vehicle Engineering, Hunan University, Changsha, China. Before teaching with Hunan University, Changsha, China, he was an Algorithm Engineer with the JDX R&D Center of Automated Driving, JD Inc., China, from 2018 to 2020. He has been the first author of nearly 80 journal/conference papers and two books in numerical optimization, motion planning, and robotics. His research focuses on motion planning methods in autonomous driving. He was the recipient of the International Federation of Automatic Control (IFAC) 2014–2016 Best Journal Paper Prize from Engineering Applications of Artificial Intelligence. He was the recipient of the 2022 Best Associate Editor Award of IEEE TRANSACTIONS ON INTELLIGENT VEHICLES. He is currently an Associate Editor for IEEE TRANSACTIONS ON INTELLIGENT VEHICLES.

Yazhou Wang received the B.S. degree in vehicle engineering in 2021 from Hunan University, Changsha, China, where he is currently working toward the master's degree with the College of Mechanical and Vehicle Engineering. His research interests include intelligent vehicle systems, with specific emphasis on decision making, trajectory planning, control, and software engineering aspects.

Siji Ma (Member, IEEE) received the B.S. degree from the School of Information Engineering, Hangzhou Dianzi University, Hangzhou, China, in 2022. He is currently working toward the master's degree with the Faculty of Innovation Engineering, Macau University of Science and Technology, Macau, China. His research interests include parallel intelligence, reinforcement learning, and decentralized autonomous organizations (DAOs) for intelligent vehicles.

Xuepeng Bian received the B.S. degree from the School of Electrical Engineering, Yanshan University, Qinhuangdao, China, in 2012, and the M.S. degree from the School of Automation, Beijing Institute of Technology, Beijing, China, in 2014. He is currently a Senior Engineer with the Tencent Automatic Drive Lab, part of Tencent.com Inc., wherein his primary work involves research and development of motion planning and control for an intelligent vehicle.

Hu Li received the B.S. degree from the School of Mechatronic Engineering, Chengnan College, part of Changsha University of Science and Technology, Changsha, China, in 2020. Since 2021, he has been working toward the master's degree with the College of Mechanical and Vehicle Engineering, Hunan University, Hunan, China. His research insterests include the trajectory planning and prediction of autonomous vehicles.

Tantan Zhang received the B.S. degree from Hunan University, Changsha, China, in 2012, the dual M.S. degrees from Politecnico di Torino, Turin, Italy, and Tongji University, Shanghai, China, in 2015, and the Ph.D. degree from Politecnico di Torino, in 2020. He is currently an Assistant Professor with the College of Mechanical and Vehicle Engineering, Hunan University. His research focuses on motion planning of automated vehicles.

Xiaohui Li received the B.S. degree in electrical engineering from the Harbin Institute of Technology, Harbin, China, in 2009, and the Ph.D. degree in control science and engineering from the College of Mechatronics and Automation, National University of Defense Technology (NUDT), Changsha, China, in 2016. He has been a visiting Ph.D. Student with Ohio State University, Columbus, OH, USA. He is currently an Associate Professor with the College of Intelligence Science and Technology, NUDT. His research activities include design and applications of complex control systems, decision-making, motion planning, and optimal control for autonomous vehicles. He was a Reviewer for a lot of Journals, including IEEE TRANSACTIONS ON INTELLIGENT TRANSPORTATION SYSTEMS, IEEE TRANSACTIONS ON VEHICULAR TECHNOLOGY, and IEEE TRANSACTIONS ON INDUSTRIAL ELECTRONICS.

Youmin Zhang (Fellow, IEEE) is currently a Professor with the Department of Mechanical, Industrial and Aerospace Engineering, Concordia University, Montreal, QC, Canada. He has authored or coauthored eight books, more than 600 journal and conference papers. His research interests include the areas of monitoring, diagnosis and physical fault/cyber-attack tolerant/resilient control, guidance, navigation and control of unmanned systems and smart grids, with applications to forest fires and smart cities in the framework of cyber-physical systems by combining with remote sensing techniques. Dr. Zhang is a Fellow of CSME, a Senior Member of AIAA, President of International Society of Intelligent Unmanned Systems (ISIUS) during 2019–2022, and a technical committee member of several scientific societies. He has been the Editor-in-Chief (EIC), Editorial Advisory Board Member of several journals, including as a Member of Board Member of Governors and Representatives for *Journal of Intelligent and Robotic Systems*, an Associate Editor for IEEE TRANSACTIONS ON INDUSTRIAL ELECTRONICS, IEEE TRANSACTIONS ON NEURAL NETWORKS & LEARNING SYSTEMS, IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS - II: EXPRESS BRIEFS, *IET Cyber-systems and Robotics*, *Unmanned Systems*, *Security and Safety*, and the Deputy EIC of *Guidance, Navigation and Control*.