IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received June 28, 2021, accepted July 14, 2021, date of publication July 20, 2021, date of current version July 27, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3098676

Autonomous Parking Trajectory Planning With Tiny Passages: A Combination of Multistage Hybrid A-Star Algorithm and Numerical Optimal Control

WEITIAN SHENG[1], BAI LI[2], (Member, IEEE), AND XIANG ZHONG[2]
[1]Changsha Intelligent Driving Institute, Changsha 410208, China
[2]College of Mechanical and Vehicle Engineering, Hunan University, Changsha 410082, China

Corresponding author: Xiang Zhong (zx5587@126.com)

This work was supported by the Fundamental Research Funds for the Central Universities of China under Grant 531118010509.

ABSTRACT This paper introduces an autonomous parking trajectory planning method in an unstructured environment with narrow passages. The proposed hierarchical trajectory planner consists of a graph search layer and a numerical optimal control layer. The contribution mainly lies in the graph search layer, wherein a multistage hybrid A* algorithm is proposed to handle narrow passages formed by obstacles in the cluttered environment. In the multistage hybrid A* algorithm, a 2-dim A* search is conducted to find a global route that connects the starting and goal points. Along the derived global route, subtle segments that traverse narrow passages are extracted. Thereafter, the hybrid A* algorithm is used to plan kinematically feasible subpaths that connect the boundary points of each subtle segment. The hybrid A* algorithm is also used to find linking paths that connect adjacent subpaths. Combining all the subpaths and linking paths in a sequence yields a coarse path, which is converted into a coarse trajectory by attaching a time-optimal velocity profile to it. The coarse trajectory is fed into the numerical optimization layer as the initial guess. Simulation results indicate that the hierarchical trajectory planner runs much faster than prevalent ones in dealing with unstructured environments with narrow passages.

INDEX TERMS Autonomous parking, hybrid A* algorithm, optimal control, trajectory planning.

I. INTRODUCTION

Autonomous driving techniques have been widely developed in recent years due to the potential to relieve traffic jams, reduce air pollution, and prevent traffic accidents caused by human errors [1]. Autonomous parking in an unstructured environment remains a challenge because the perception, planning, and control modules of a vehicle may not be as efficient as they are on a structured highway [2]. This work focuses on the autonomous parking trajectory planning scheme in an unstructured scenario.

Prevalent trajectory planners suitable for parking in an unstructured environment may be classified into sampling- and optimization-based categories [3]. Sampling-based methods can be further classified by sampling the input or state space. Typical samplers in the input space include the dynamic window algorithm [4] and the hybrid A* algorithm [5], while typical samplers in the state space include the state lattice planner [6] and rapidly exploring random trees (RRT) [7], [8]. A sampling-based method, in most of the cases, runs fast to derive a resolution-feasible path/trajectory. However, it may fail when the environment is maze-like or contains narrow passages, wherein the incremental sample and search would be stuck. In addition, the resultant path or trajectory is only resolution-feasible, which means it may be infeasible w.r.t. the interior vehicle kinematic constraints or the exterior collision-avoidance constraints.

Optimization-based methods, on the contrary, are promising to provide precise solutions free from the issue of resolution [9]. But such methods only find local optima, thus they need an identified neighborhood in the solution space wherein a good local optimum exists.

The associate editor coordinating the review of this manuscript and approving it for publication was Okyay Kaynak.

A sampling-based method and an optimization-based method work better when they are combined [3], [10]–[13]. A sampling-based method provides a resolution-feasible coarse path/trajectory, which serves as the initial guess in solving an optimization problem via a local optimizer. However, such a hierarchical method still runs slowly because the concerned problem contains large-scale nonconvex and nondifferentiable collision-avoidance constraints [14]. As we will show in the simulations, directly combining a typical sampling-based method and an optimization-based method costs excessive CPU time. To conclude, the sampling and numerical optimization layers in the current hierarchical planning framework are imperfect.

In this paper, we propose a trajectory planner that adopts the hierarchical planning framework, wherein the sampling-based layer is developed based on the conventional hybrid A* algorithm, and the optimization-based layer is conducted with an improved safe travel corridor (STC)-based optimization approach. The core contribution lies in the proposal of a multistage hybrid A* algorithm and an improved STC model, which are particularly suitable for autonomous parking scenarios with narrow passages. The multistage hybrid A* algorithm identifies narrow passages from the environment first and searches paths through them individually before connecting them all together to form a global trajectory. The improved STC method then regards it as the initial guess and solves an optimal control problem (OCP), in which a set of within-STC constraints replaces the commonly used full-scale collision-avoidance constraints. The benchmark cases presented in this paper demonstrate that the combination of the multistage hybrid A* algorithm and improved STC-based optimization provides a remarkable advantage on cost efficiency over conventional optimization-based counterparts and has potential in dealing with on-site autonomous planning cases.

A. RELATED WORK

Sampling-based planners, especially those with RRT, have been investigated intensively to deal with narrow passages in unstructured environments but have shown limitations there. Dong *et al.* [15] proposed a skeleton-biased locally seeded RRT, which generates random seeds around a global route so that narrow passages can be tackled easily. Works based on RRT and bridge test [16] have also identified narrow passages on a probabilistic roadmap. Wang *et al.* [17] used a two-stage approach to address narrow passages by first identifying them and then applying two varieties of RRTs to search for local and global paths. Shu *et al.* [18] proposed a locally guided RRT that can rapidly find routes in the clustered environments. A learning-based RRT approach [19] that uses a reinforcement learning method to enhance local exploration capability has also been proposed to tackle path planning in a narrow environment. Such RRT-based planners, although run fast, are not deterministic, thus causing trouble to keep the planning module stable. Expect from the RRT series, a fast marching method (FMM) has been proposed with a support vector machine (SVM) to plan paths through narrow passages [20]. However, using SVM to keep a good distance from the obstacles requires remarkable computation sources and thus is unsuitable in a dynamic environment.

Regarding the optimization-based planners, a common limitation is that they cannot rapidly handle intractably scaled collision-avoidance constraints. Sequential convex programming (SCP)-based methods [21] have been applied to linearize nonconvex collision-avoidance constraints. Zhang *et al.* [10] introduced costate variables to make collision-avoidance constraints nearly linear. Nonetheless, the scale of collision-avoidance constraints is still intractable. Inspired by the concept of safe flight corridor in unmanned aerial vehicle path planning [22], Li *et al.* [13] paved the aforementioned STC to separate an ego vehicle from the surrounding obstacles with the homotopy class selected by a sampling-based planner. Although STC makes the problem dimensionally irrelevant to the environment complexity, it cannot cover all the cases with narrow passages because the STC method has to leave out semilunar buffers around the ego vehicle.

B. ORGANIZATION

The remainder of this paper is organized as follows. Section II provides the details of the proposed multistage hybrid A* algorithm, followed by an introduction of the improved STC model in Section III. In Section IV, simulation results of hundreds of benchmarks are presented to demonstrate the efficiency and robustness of the proposed trajectory planner. Finally, conclusions and future works are indicated in Section V.

II. MULTISTAGE HYBRID A* ALGORITHM

Before the proposed multistage hybrid A* algorithm is described, the trajectory planning problem concerned in this paper is briefly stated. Given a region of interest Ω, an obstacle space Ω_{obs}, and the start and end states of a vehicle, the proposed trajectory planner tackles a problem defined as follows: finding a kinematic feasible vehicle trajectory along with its controls that connects the start and end states within the free space $\Omega_{free} = \Omega \backslash \Omega_{obs}$.

Prior to the introduction of the multistage hybrid A* algorithm in Section II.B, the conventional hybrid A* algorithm is briefly presented in Section II.A as a preliminary.

A. CONVENTIONAL HYBRID A* ALGORITHM

The conventional hybrid A* algorithm extends the well-known A* algorithm for nonholonomic autonomous vehicles. It samples the region of interest Ω (Fig. 1) and the ego vehicle's orientation angle into a graph of grids, in which each grid node N_i is labeled $(x_i,\ y_i,\ \theta_i)$, $i \in [1,\ N]$. Here x_i, y_i and θ_i denote the position and orientation angle of the vehicle at node i, respectively. N denotes the total number of nodes in the graph. Thus each node can represent some vehicle states ranging in an interval. Through searching in the discretized space, the hybrid A* algorithm aims to find a trajectory that

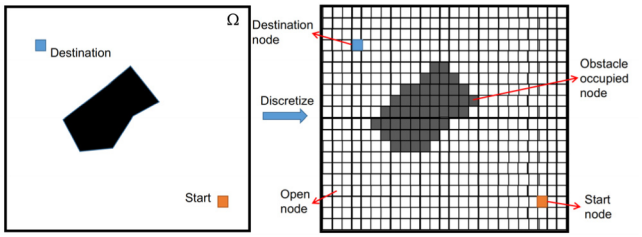

FIGURE 1. Discretized map in the hybrid A* algorithm.

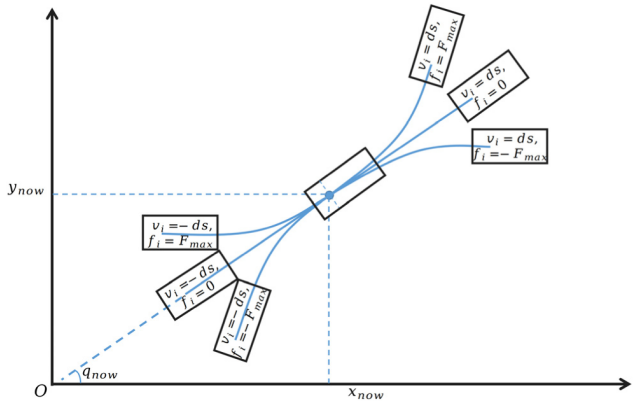

FIGURE 2. Extending a number of child nodes from a parent node (x_{now}, y_{now}, q_{now}) based on the vehicle kinematic principle and forward simulation.

connects the given starting point and destination with the vehicle dynamics considered.

The cost of a node is the sum of cost-so-far and the heuristic function named cost-to-go function. The cost-to-go function is obtained based on a grid cost map computed using the common A* [5]. The aforementioned priority queue is initialized with the start node. In each search iteration, the node with the lowest cost is selected to expand several child nodes, which will be added to the priority queue for future selection. For each of the nodes selected from the queue, the Reeds–Shepp (RS) curve [23] might be generated to connect the node with the destination (there should be a rule to judge whether to activate the RS curve generation procedure or not for the current node). If the identified RS curve is collision-free, the hybrid A* algorithm returns with a complete path, which consists of the link from the start node towards the current node as well as the RS curve. A trajectory is determined by attaching a velocity along the complete path. Conversely, if the RS curve involves collisions, the search iteration continues. The child nodes are obtained via forward simulation (Fig. 2). The child nodes are then added to the priority queue the parent-child path segment is collision-free. When a child node already exists in the queue, the hybrid A* algorithm updates its cost if the cost computed in this iteration is smaller. Aside from finding a valid RS curve, the algorithm also exits if a child node is found to be the destination node. The final trajectory is then resembled by tracing back from the final node to the start node.

Ideally, if the heuristic function is selected wisely, the hybrid A* algorithm can efficiently navigate through the sampling space and reach the destination without exploring excessive nodes. However in dealing with an environment with narrow passages, the heuristic function can hardly find the very few feasible nodes in the narrow passage quickly before wasting much time to extend the invalid nodes. Our multistage hybrid A* algorithm is proposed to deal with this problem.

B. MULTISTAGE HYBRID A* ALGORITHM

The proposed multistage hybrid A* algorithm follows the strategy of divide and conquer. With a pair of starting and ending configurations assigned, the multistage hybrid A* algorithm first generates a path connecting them using the common A* method [24]. This path may go through some narrow passages. Next, the derived A* path is divided into segments. This division procedure is used to identify which interval of the derived A* path lies in a narrow passage. Such intervals would be marked as subtle segments (SSs). Thereafter, the conventional hybrid A* search is conducted to find kinematically feasible passing subpaths (PSPs) connecting the local starting and ending points of each SS, followed by using the hybrid A* algorithm again to find linking subpaths (LSPs) connecting those PSPs. After all the subpaths are obtained, the proposed planner combines them all together and attaches a minimal-time velocity profile to form a complete coarse trajectory, which would serve as the initial guess in numerical optimal control.

The aforementioned procedures are summarized into the following three steps:

Step 1: identifying all narrow passages within the region of interest with the common A* algorithm and obtaining SSs that traverse these passages

Step 2: using the hybrid A* algorithm to search for kinematically feasible PSPs through those passages in accordance with the SSs obtained in step 1, then using the hybrid A* algorithm again to obtain LSPs that connect adjacent PSPs

Step 3: combining all the PSPs and LSPs, and attaching a time-optimal velocity profile to obtain a trajectory connecting the start and destination nodes.

These three steps are described in detail as follows:

1) STEP 1

The identification of the narrow passages is expected to perform in a light-weighted way. To this end, the 2-dim A* algorithm is first used to obtain a path c^{2d} from the start to end node (Fig. 3a). This process does not require additional efforts because the path is already found when A* is used to obtain a heuristic cost grid map. It simply needs to record when A* reaches the start node and retrieve the path through its parent node all the way back to the destination node.

When c^{2d} is available, SSs should be identified as the next step, which is achieved by running a loop over all the grids on c^{2d}. Whether or not a grid i, namely $N_i^{2d}(x_i,\ y_i)$, lies in a narrow passage is judged by computing four points around: $D_1(x_i + d,\ y_i)$, $D_2(x_i - d,\ y_i)$, $D_3(x_i,\ y_i + d)$, and $D_4(x_i,\ y_i - d)$, where d is a user-defined parameter.

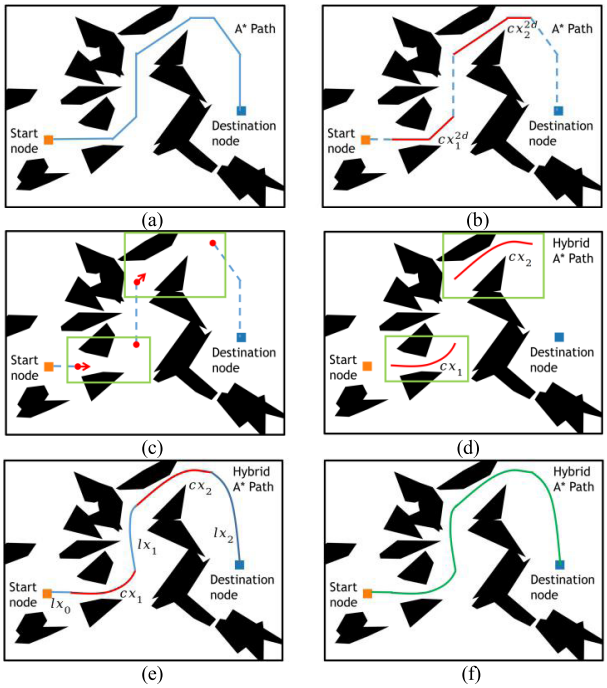

FIGURE 3. Schematics on the three steps in the multistage hybrid A* algorithm: (a) Generating a 2-dim A* path to connect the start and destination nodes. (b) Obtaining SSs through narrow passages. (c) Defining a local search region (light green box) and start and end nodes. (d) Computing PSPs within the local search region via hybrid A* search. (e) Computing LSPs also via hybrid A* search. (f) Combining all subpaths to obtain a complete coarse trajectory.

If D_1 and D_2 (or D_3 and D_4) are both in the obstacles, then the grid N_i^{2d} is taken as in a narrow passage, and the Boolean value is_narrow(N_i^{2d}) of this grid is set to *true*. A consecutive section of nodes with is_narrow(N_i^{2d}) $=$ *true* is recorded. If the length of such a section is larger than a threshold *Thres*, then this section is determined to be a valid SS going through a narrow passage and denoted as cx_k^{2d} if it is the kth SS (Fig. 3b).

Upon exiting the loop, a set denoted as CX of such SSs is obtained. Each of the SSs in CX is further extended by *Thres* nodes from their two boundaries. Step 1 is summarized in Alg. 1.

2) STEPS 2 AND 3

Although a set of SSs has been obtained in step 1, they are not kinematically feasible, and the hybrid A* searcher is adopted to refine these coarse paths. During the refinement, only the local start and end grids of each SS are fixed while the intermediate grids are flexible to make the SS better satisfy the kinematic constraints. Given that the A* is performed over the 2-dim Cartesian space, defining the start node for the hybrid A* algorithm requires setting the vehicle orientation angle manually, and multiple trials may be needed to find a good direction to begin the narrow passage traverse (Fig. 3c). Let the position of the start of cx_k^{2d} be $(x_k^s,\ y_k^s)$, the start grid for the hybrid A* algorithm is defined as $(x_k^s,\ y_k^s,\ \theta_k^{ini}+nd\theta)$. Here, θ_k^{ini} is an arbitrary initial trial angle, $d\theta$ is the unit step size ($n = 0,\ \ldots,\ N_{trial}$), where N_{trial} denotes the total

Algorithm 1 Finding SSs Going Through Narrow Passages

Input: 2-dim discretized map, starting point and destination
Output: path set containing SSs in narrow passages

```
Initialize empty node set cx^tmp, empty path set CX
Adopt the 2-dim A* algorithm to obtain c^2d
if finding c^2d fails; return false
for each node N_i^2d in c^2d, do
    Compute D_1, D_2, D_3, and D_4; set is_narrow(N_i^2d)
    if is_narrow(N_i^2d) is true, then
        Push N_i^2d into cx^tmp
    else
        if length(cx^tmp) > Thres, then
            Push cx^tmp into CX
        end if
        cx^tmp = ∅
    end if
end for
for each path cx_k^2d in CX, do
    Extending cx_k^2d by Thres nodes from local start and end nodes
    if overlap with cx_{k-1}^2d, then
        Combine cx_k^2d and cx_{k-1}^2d
    end if
end for
```

number of trials predefined by the user. Given that the end grid of cx_k^{2d} also has only position information (denoted as $(x_k^e,\ y_k^e)$), the hybrid A* algorithm simply needs to find a node with a matching position to exit.

With local start and end grids properly defined, the hybrid A* algorithm is used again to search for kinematically feasible PSPs, thereby linking those narrow passages. These PSPs, denoted as $cx_k (k = 1,\ \ldots,\ K)$, are used to replace the corresponding SSs in CX. The local search space in each hybrid A* search can be slightly larger than the passage so that the search process is ensured to finish fast even if multiple orientation angles of the start grid are tried (Fig. 3d). Given that these PSPs are separated from one another and usually do not connect with the global start and destination nodes, the hybrid A* algorithm is used again to find LSPs, denoted as $lx_m,\ m = 0,\ \ldots,\ K$, to connect them and stored in a set LX. Together, PSPs and LSPs constitute a complete path from the global start to the destination (Fig. 3e).

In step 3, all nodes in the subpaths are combined to form a coarse path, which becomes a trajectory with an additional time-optimal velocity profile attached (Fig. 3f). The complete multistage hybrid A* algorithm is summarized in Alg. 2.

Using the hybrid A* algorithm for PSPs and LSPs is generally fast and reliable because the search space is confined to a small area in each run. For the PSPs, the search space is small, and the local starting point is close to the entry of the narrow passage, leading to an easy search of the correct node into the narrow passage. In addition, since the LSPs always locate in wide-open spaces, a valid RS curve can be found with ease. On the contrary, validating whether an RS curve is collision-free during every search iteration in the hybrid A* algorithm is ineffective and time-consuming. To improve efficiency, RS curve validation is activated only

Algorithm 2 Multistage Hybrid A* Algorithm

Input: 2-dim discretized map, starting point, and destination
Output: coarse trajectory connecting the starting point and destination

Call algorithm 1 to obtain CX, **if** fail, **return** false;
Continue using A* to obtain a heuristic cost map
for each subpath cx_k^{2d} in CX, **do**
  Choose θ_k^{ini} and $d\theta$
  for $n = 0$ to N_{trial}, **do**
   Try hybrid A* to connect $(x_k^s,\ y_k^s,\ \theta_k^{ini}+nd\theta)$ and $(x_k^e,\ y_k^e)$
    if success, **then**
      Obtain PSP cx_k and $CX[k] = cx_k$
      break
    end if
  end for
  if all trials fail, **return** false
end for
Initialize LSP set LX with size $K + 1$
Try hybrid A* to connect global start and start node of cx_1
if fail, **return** false; **else**, push path to LX
for $k = 2$ to K, **do**
  Use hybrid A* to connect start of cx_k and end of cx_{k-1}
  if fail, **return** false; **else**, push path to LX
end for
Try hybrid A* to connect end node of cx_{K-1} and global destination
if fail, **return** false; **else**, push path to LX
Combine all subpaths in CX and LX to obtain global path
Attach dynamics profile to global path to obtain coarse trajectory

when the curve length is no smaller than the current holonomic heuristic cost.

If no narrow passage ever exists, the multistage hybrid A* algorithm reduces to the conventional hybrid A* algorithm. Hence, it does not consume additional time in dealing with mild cases. In summary, through divide and conquer, our proposal decomposes the originally complex searching scheme into several simple ones. Solving those simple problems sequentially is considerably easier and faster.

III. IMPROVED STC-BASED TRAJECTORY OPTIMIZATION

An optimization-based trajectory planning method solves an OCP consisting of a cost function and many constraints. Given that the general nonconvex and nondifferentiable collision-avoidance constraints are difficult to handle, Li *et al.* [13] used STCs to separate the ego vehicle from all of the surrounding obstacles, thereby replacing the nominal collision-avoidance constraints with within-STC constraints. This work follows the basic optimization framework of [13] to refine the coarse trajectory derived in the preceding section.

The cost function f in the concerned OCP is designated to improve the smoothness of the trajectory and to reduce the traverse time.

The single-track bicycle model is enough to describe the low-speed vehicle kinematics:

$$\frac{\mathrm{d}}{\mathrm{d}t}\begin{bmatrix} x(t) \\ y(t) \\ v(t) \\ \phi(t) \\ \theta(t) \end{bmatrix} = \begin{bmatrix} v(t)\cos\theta(t) \\ v(t)\sin\theta(t) \\ a(t) \\ \omega(t) \\ v(t)\tan\phi(t)/\mathrm{L_w} \end{bmatrix}, \quad t \in [0,\ T], \tag{1}$$

where $x(t)$ and $y(t)$ are the positions of mid-point of the vehicle's rear wheels at time t. $v(t)$, $a(t)$, $\phi(t)$, $\omega(t)$, and $\theta(t)$ are the vehicle's velocity, acceleration profile, steering angle, angular velocity, and orientation angle at t, respectively. $\mathrm{L_w}$ refers to the wheelbase, and T denotes the completion time of the entire parking process. The boundaries of the vehicle's state/control profiles are

$$\begin{matrix} |a(t)| \le a_{\max} \\ |v(t)| \le v_{\max} \\ |\omega(t)| \le \Omega_{\max} \\ |\phi(t)| \le \Phi_{\max} \end{matrix}, \quad t \in [0,\ T], \tag{2}$$

where $a_{\max}$, $v_{\max}$, $\Omega_{\max}$, and $\Phi_{\max}$ are the maximum limits of acceleration, velocity, angular velocity, and steering angle, respectively. The boundary conditions of the initial and terminal states of the vehicle are defined as

$$\begin{bmatrix} x(0) \\ y(0) \\ \theta(0) \\ v(0) \\ \phi(0) \\ a(0) \\ \omega(0) \end{bmatrix} = \begin{bmatrix} x_0 \\ y_0 \\ \theta_0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \text{ and } \begin{bmatrix} x(T) \\ y(T) \\ \theta(T) \\ v(T) \\ \phi(T) \\ a(T) \\ \omega(T) \end{bmatrix} = \begin{bmatrix} x_T \\ y_T \\ \theta_T \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \tag{3}$$

where x_0, y_0, θ_0, x_T, y_T, and θ_T are the initial and terminal configurations of the vehicle, respectively. Here, the vehicle is assumed to be completely stopped before and after the parking process.

The within-STC constraints are established by the following steps. First, two discs are deployed to cover the vehicle body. By dilating the obstacles by the radius of each disc and shrinking the vehicle discs to their centers, we convert the original problem into a new form [13]. Next, the trajectories of the two disc centers are determined according to the coarse trajectory derived in the preceding section. Lastly, STCs are generated by gradually enlarging a regularly placed box at each sampling point along either of the two trajectories until all four sides of each box reach obstacles or a threshold length $\mathrm{L_{max}}$. STCs are generated in accordance with the dilated map, such that requiring only the two discs' center to be within STCs is adequate to guarantee that the ego vehicle keeps clear of the surrounding obstacles. The details of the STC generation process can be found in [13]. Given that the within-STC constraints are box constraints and extremely friendly to general nonlinear programming (NLP) engines, the trajectory optimization problem is easy to solve and completely independent from the complexity of the surrounding environment [13].

Despite the bright side, the STC generation approach may be computationally expensive in some specific environments, thus it deserves further improvement.

To enlarge the rectangle surrounding a sample point gradually, a step size dl must be identified. The choice of this step size is critical because it can neither be extremely large nor extremely small. If it is set too large, STCs become overcautious, and the resultant trajectory suffers from optimality loss

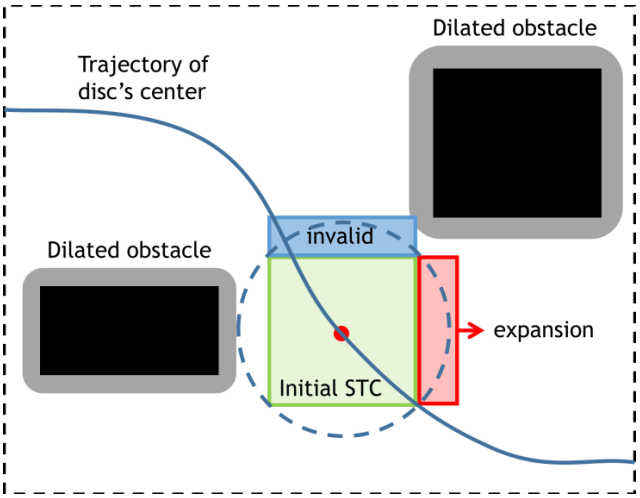

FIGURE 4. Generating STC with dilated obstacles. The red point is a sample point on the trajectory of one disc's center. The green box is the initial STC square whose circumcircle reaches the nearest dilated obstacle. In the expansion process, each side of STC is extended by dl until it reaches a dilated obstacle.

or even a solution failure. Conversely, if dl is set too small, the computational cost would be unaffordable. Empirically, dl has to be small, otherwise the OCP solution success rate would be affected by being overcautious, which renders troubles. Given that dl is small, we would make efforts to reduce the runtime.

The idea to reduce the STC identification runtime is about starting the process from a rectangle whose circumcircle reaches the nearest obstacle instead of starting from scratch (Fig. 4). Identifying the nearest obstacle is computationally cheap because it does not necessarily require calculating the exact distance between all surrounding obstacles and the sample point [25]. After the nearest obstacle is specified, the exact distance between it and the sample point can be calculated, and the initial STC square is built by setting the distance as half of the diagonal length. The initial STC square is then gradually enlarged until all four sides reach obstacles or the length exceeds L_{max}. This improvement, although being simple, offers a remarkable advantage over the original method because acquiring the initial square is faster than enlarging a rectangle from a mass point. The enlargement procedure after determining the initial square oftentimes completes very soon because the initial rectangle is not far from the obstacles. Moreover, when the nearest obstacle is already outside the largest box defined by L_{max}, the original enlarging process, which is computationally expensive, can be avoided.

Let us denote the jth sample points on the trajectories of the front and rear disc centers as $P_f(j)$ and $P_r(j)$, respectively, and the spaces enclosed by the corresponding STCs as $S_f(j)$ and $S_r(j)$. The within-STC constraints are written as

$$\begin{aligned} P_f(j) &\in S_f(j), \\ P_r(j) &\in S_r(j), \end{aligned} \tag{4}$$

where $j = 1, \ \ldots, \ N_s$, and N_s is the total number of sampling points on the disc center's trajectory. The OCP can now be formulated as

$$\begin{aligned} &\text{minimize cost function } f \\ &s.\,t.\ \text{kinematic constraints (1), (2);} \\ &\qquad \text{boundary conditions (3);} \\ &\qquad \text{within-STC conditions (4).} \end{aligned} \tag{5}$$

The optimization problem (5) can be discretized in time, such that all of the state and control profiles are presented by a finite number of to-be-optimized grids. The converted optimization problem is a typical NLP problem to be solved by a standard NLP engine, such as IPOPT [26]. The NLP initial guess is provided by the coarse trajectory generated in the preceding section.

TABLE 1. Simulation parameters.

Description	Value
Front hang length of the vehicle	0.96 m
Wheelbase of the vehicle	2.80 m
Rear hang length of the vehicle	0.929 m
Width of the vehicle	1.942 m
Minimum turn radius	3.32 m
Maximum velocity forward	2.0 m/s
Maximum velocity backward	1.0 m/s
Maximum acceleration	2.0 m/s^2
Maximum deceleration	2.0 m/s^2
Maximum steering angle	0.7 rad
Maximum steering rate	0.5 rad/s
Hybrid A* algorithm grid resolution	0.3 m
Hybrid A* algorithm step size	0.5 m
Maximum side length of STCs	20 m
STC generation unit step size	0.05 m

IV. SIMULATION RESULTS

The proposed trajectory planner is implemented using C++, and all simulations are performed on an i7-10700F CPU running at 2.90 GHz with 16.0 GB RAM. Parametric settings are listed in Table 1. All of the simulation results obtained using our proposed planner (referred to as the MSH-STC planner) are compared with the ones obtained from two other competitive planners: one uses the conventional hybrid A* algorithm and aforementioned STC-based optimization (referred to as the H-STC planner), and the other is the TDR-OBCA planner [27] from Baidu Apollo [28]. TDR-OBCA is chosen as a competitor because it is used in the industry-recognized Baidu Apollo autonomous driving platform and has a similar planning process to our proposed planner but TDR-OBCA still employs full-scale collision-avoidance constraints in optimization. Thus it would be interesting to see our efforts to make the collision-avoidance constraints tractably scaled. Each single CPU time reported is the mean value measured from 10 repeated runs of the same simulation.

A. VALET PARKING SCENARIO

The proposed planner is first used in a simple and obstacle-free valet parking scenario. In this scenario, the vehicle is parked in a structured lot from a nearby road, where the

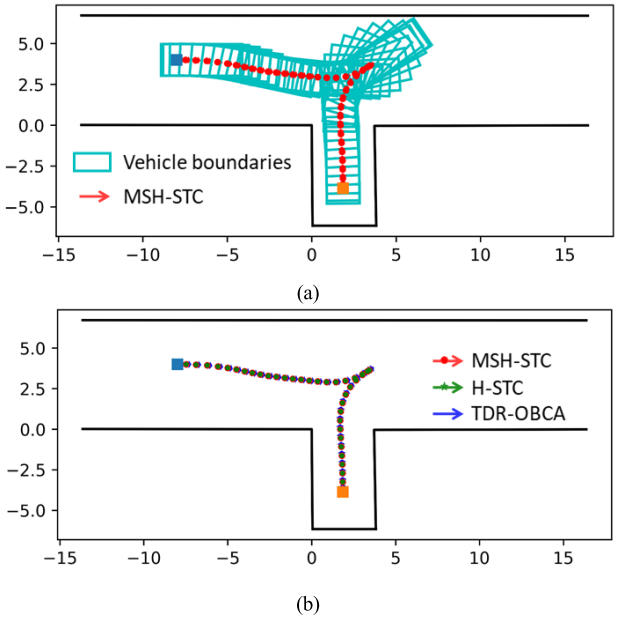

FIGURE 5. Valet parking trajectories obtained using (a) the proposed MSH-STC planner; and (b) H-STC and TDR-OBCA planner. The solid blue and orange dots represent the start and destination nodes, respectively.

TABLE 2. CPU Time of different planners.

Scenario	Planner	Coarse trajectory generation		OCP
		CPU Time / s	Number of searched nodes	CPU Time / s
Valet parking	MSH-STC	0.0285	2674	0.0386
	H-STC	0.0281	2674	0.0384
	TDR-OBCA	0.0280	2674	0.253
Random Scenario 1	MSH-STC	0.220	14525	0.312
	H-STC	4.468	501053	0.412
	TDR-OBCA	4.470	501053	148.79
Random Scenario 2	MSH-STC	0.076	6105	0.360
	H-STC	3.172	323587	0.185
	TDR-OBCA	3.205	323587	39.13
Random Scenario 3	MSH-STC	0.112	6039	0.155
	H-STC	2.337	9258	0.144
	TDR-OBCA	2.380	9258	26.94

start node $(x_0,\ y_0,\ \theta_0)$ and destination node $(x_T,\ y_T,\ \theta_T)$ are at $(-8, 4, 0)$ and $(1.86, -3.86, 1.58)$, respectively. The paths, velocity, acceleration profile, and steering angle of the vehicle obtained using each planner are shown in Figs. 5 and 6. The simulation results indicate that all of the three planners generate almost identical trajectories that are smooth and collision-free. Thus our proposed planner can generate trajectories that meet industry standards and can be used in practice. CPU time measurements in this scenario are listed in Table 2. It is clear that our proposed planner does not consume much time when used in a simple scenario, and it is much faster than TDR-OBCA that uses full collision-avoidance constraints even in a simple scenario. In addition, the total CPU time measured for the proposed planner is not marginally less than 100 ms, indicating that it can be reliably used in a real-time autonomous driving planning module for valet parking Pscenarios.

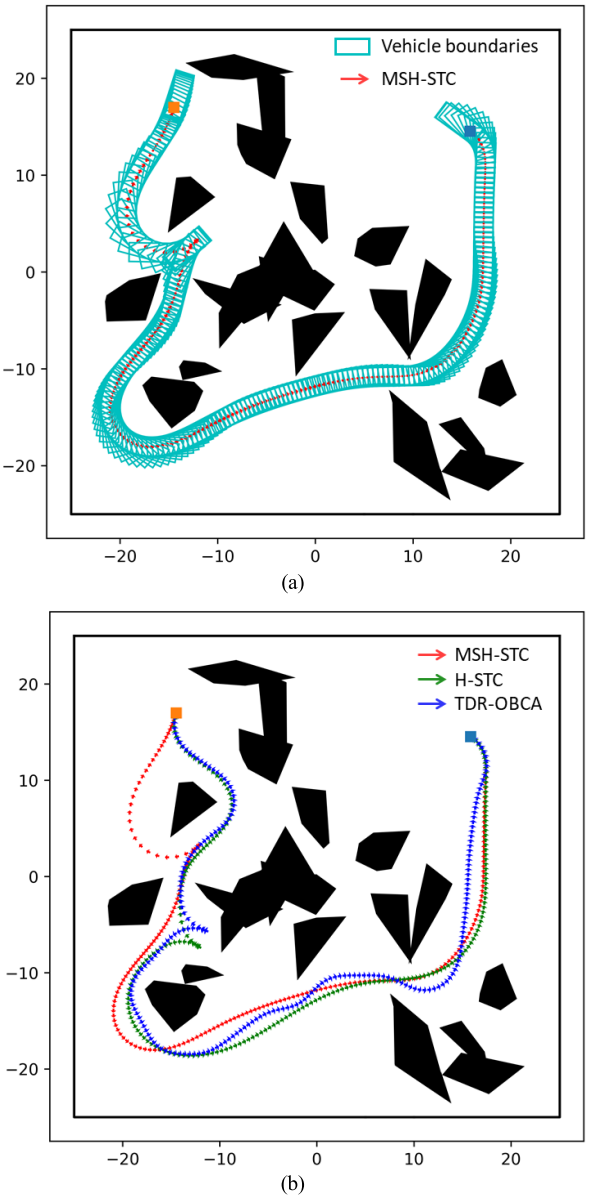

FIGURE 6. Random scenario 1: Trajectories obtained using (a) the proposed MSH-STC planner, and (b) H-STC and TDR-OBCA planners.

B. SCENARIOS WITH NARROW PASSAGES

To demonstrate the efficiency of the proposed planner, the simulation results for three random scenarios with narrow passages are presented. All scenarios are within a 50m × 50m area that mimics a ground mining site, where some unstructured debris is randomly placed.

1) RANDOM SCENARIO 1

In the first scenario, the start $(x_0,\ y_0,\ \theta_0)$ and destination nodes $(x_T,\ y_T,\ \theta_T)$ are at (15.85, 14.51, 2.50) and $(-14.47,$ 16.97, 1.19), respectively. The results of trajectories from the three planners are shown in Fig. 7. As many as 23 obstacles are presented in this scenario, seemingly forming two narrow passages. The results show that the final trajectory from

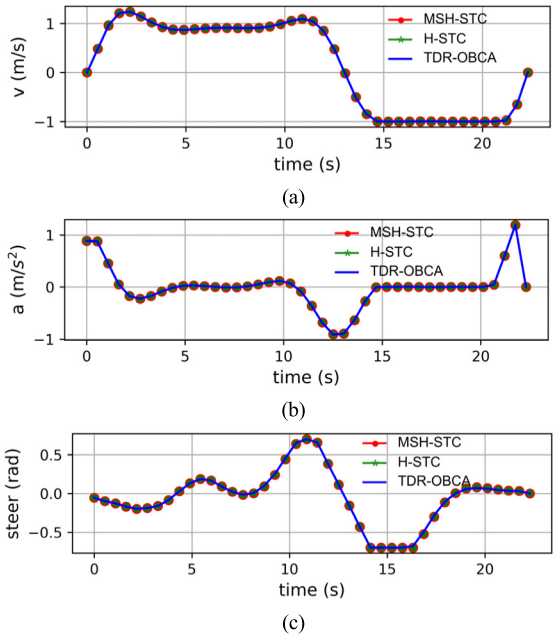

FIGURE 7. Resultant vehicle state profiles reflecting the satisfaction of vehicle kinematic constraints. The solutions are obtained using MSH-STC, H-STC, and TDR-OBCA in a valet parking scenario.

MSH-STC is well through those narrow passages, kept clear of obstacles, and reasonably smooth, although it needs to divide the region of interest for initial coarse planning.

2) RANDOM SCENARIO 2

The start and destination nodes for the second scenario are located at $(x_0,\ y_0,\ \theta_0) = (-10.51,\ -4.34, 2.78)$ and $(x_T,\ y_T,\ \theta_T) = (16.74, 3.12, 3.03)$, respectively. Trajectories are illustrated in Fig. 8. Although this scenario is simpler than the first one with only 16 obstacles, the narrow passage requires the vehicle to traverse carefully because it decelerates before the entry in all three trajectories (arrows are dense before the passage entry). Again, the results show that the MSH-STC planner can generate intuitively understandable trajectories.

3) RANDOM SCENARIO 3

The third scenario contains 22 randomly deployed obstacles. $(x_0,\ y_0,\ \theta_0)$ and $(x_T,\ y_T,\ \theta_T)$ are set to (16.25, 11.08, 0.81) and $(-2.54, -10.62, -2.54)$, respectively. Trajectories from the three planners are shown in Fig. 9. In this case, the ego vehicle's traversed distance is less than those in the previous scenarios, but it encounters a narrow passage just in the way between the start point and the destination. The passage is nearly perpendicular to the vehicle's initial heading angle, mounting a challenge to the conventional method. The challenge comes from the early activation of RS curve validation because the holonomic heuristic cost is smaller than the length of the RS curve from the beginning of the search iteration. Thus, as summarized in Table 2, although the vehicle is not far from the entry, the conventional hybrid

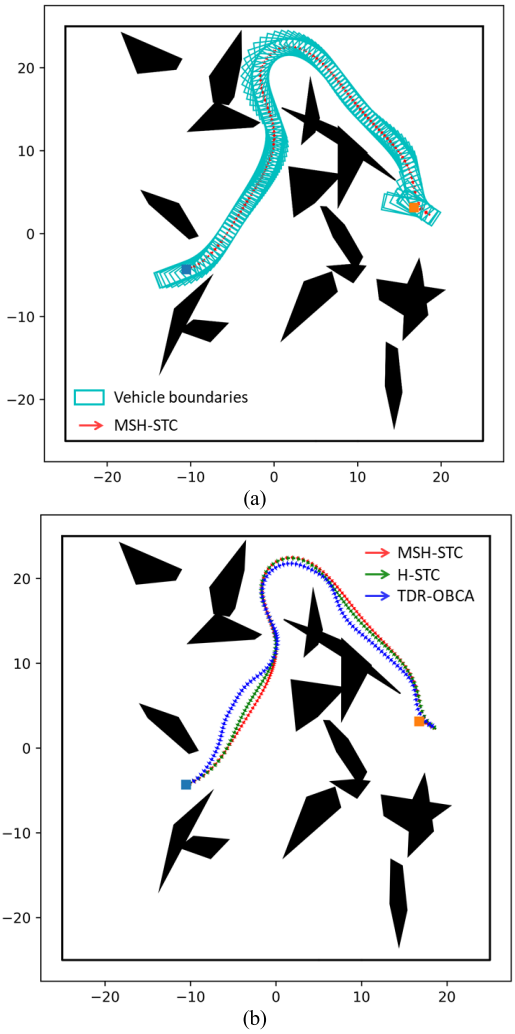

FIGURE 8. Random scenario 2: Trajectories obtained using (a) the proposed MSH-STC planner, and (b) H-STC and TDR-OBCA planners.

A* algorithm still requires much time because it needs to validate a long RS curve during each iteration. By contrast, the proposed planner is more than 20× faster in generating a coarse trajectory due to a decomposed search space.

Table 2 reports the CPU time consumed to plan the trajectory in each scenario using each planner. Our proposed MSH-STC planner is at least 20 times faster than H-STC when generating a coarse trajectory in the three scenarios, demonstrating the superiority of the proposed planner under the presence of narrow passages. This improvement in CPU time in the first two random scenarios comes from a smaller number of nodes searched using the multistage hybrid A* algorithm compared with those using the conventional hybrid A* algorithm (Table 2). Regarding the third random scenario, despite the difference in searched nodes is minimal, the time of the conventional hybrid A* algorithm is still remarkably longer than that of the proposed algorithm due to the validation of RS curves during search iterations.

Moreover, the time required by the STC-based optimizer is drastically less than that of TDR-OBCA (at least 173× faster).

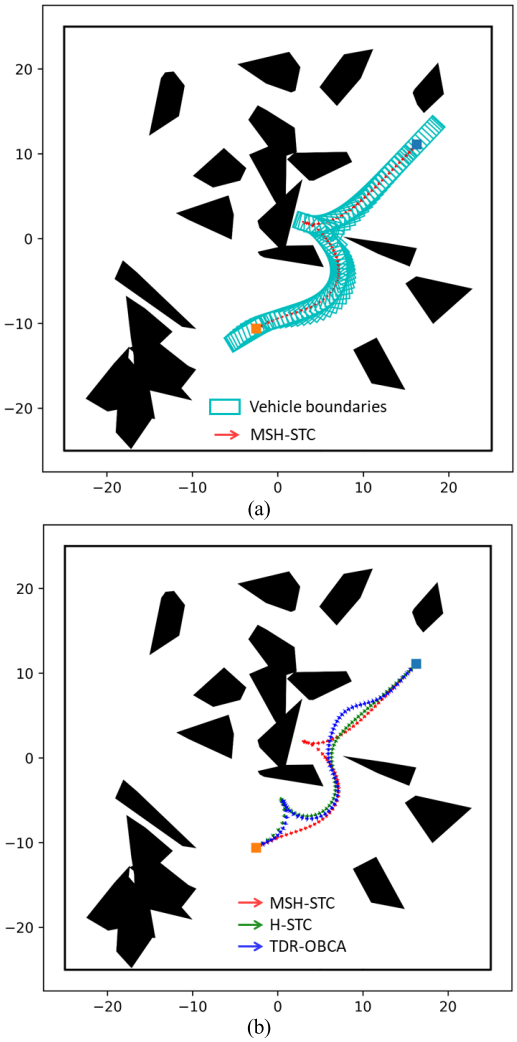

FIGURE 9. Random scenario 3: Trajectories obtained using (a) the proposed MSH-STC planner, and (b) H-STC and TDR-OBCA planners.

This result demonstrates a remarkable advantage of using STCs as collision-avoidance constraints over the full-scale models.

C. BENCHMARK OF 500 RANDOM CASES

500 random unstructured scenarios are simulated for fully testing the performance of the proposed planner. All these scenarios are within a 50m × 50m region of interest, in which the number, shape, and location of obstacles are randomly decided. Specifically, the obstacles are modeled as random polygons with several sides selected from 3 to 8, and the number of obstacles in one case is randomly chosen from 6 to 26. When generating the benchmark, viable paths between the starting point and destination are not guaranteed, such that planning failure is possible. Table 3 summarizes the statistics of this benchmark. The proposed MSH-STC planner presents better performance than the other two planners in all aspects. Compared with that of H-STC, the 99% CPU time of MSH-STC is approximately 20% less, showing that it is more capable of solving difficult cases. For the most challenging cases, the proposed planner can save even more, given that the max CPU time is 75% less than that of H-STC. MSH-STC and H-STC are approximately 100 times faster than TDR-OBCA. Hence, the STC-based optimization method is much more efficient than using full-scale collision-avoidance constraints while keeping a good optimization success rate.

TABLE 3. Benchmark CPU time.

Index	MSH-STC	H-STC	TDR-OBCA
Mean CPU time	0.154 s	0.176 s	19.48 s
Max CPU time	1.177 s	4.596 s	186.6 s
Median CPU time	0.116 s	0.131 s	10.47 s
99% CPU time	0.910 s	1.140 s	102.8 s
Number of failed cases in the coarse trajectory generation phase	7	7	7
Number of failed cases in the numerical OCP phase	0	0	0

V. CONCLUSION AND FUTURE WORK

A trajectory planner combining the multistage hybrid A* algorithm and an improved STC-based optimization method is proposed for autonomous driving planning in a complex and unstructured environment. The proposed planner is specially designed for environments with narrow passages. The planner first uses the multistage hybrid A* algorithm to generate a coarse trajectory. The multistage hybrid A* algorithm divides the path obtained using A* between the start and destination nodes and identifies those SSs that locate in narrow passages. The planner next uses the conventional hybrid A* algorithm to find kinematically feasible PSPs and LSPs and combines them to obtain a global coarse trajectory as the initial guess for optimization. The improved STC-based optimization method uses STCs to replace conventional full-scale collision-avoidance constraints with simple and optimization-friendly box constraints so that the optimization problem can be solved much more efficiently. It also uses a new strategy for generating STCs to achieve better performance. Simulation results leverage the advantage of our proposal over predominant methods in dealing with complicated cases with narrow passages.

Although the proposed planner shows improvements over some other planners, it still solves a relatively hard NLP problem to find an optimal trajectory, which runs not as fast as on-road trajectory planning. Further reducing the runtime would be our future work.

REFERENCES

[1] L. Claussmann, M. Revilloud, D. Gruyer, and S. Glaser, "A review of motion planning for highway autonomous driving," *IEEE Trans. Intell. Transp. Syst.*, vol. 21, no. 5, pp. 1826–1848, May 2019.

[2] C. Sun, Q. Li, B. Li, and L. Li, "A successive linearization in feasible set algorithm for vehicle motion planning in unstructured and low-speed scenarios," *IEEE Trans. Intell. Transp. Syst.*, early access, Jan. 8, 2021, doi: 10.1109/TITS.2020.3041075.

[3] B. Li, T. Acarman, Y. Zhang, L. Zhang, C. Yaman, and Q. Kong, "Tractor-trailer vehicle trajectory planning in narrow environments with a progressively constrained optimal control approach," *IEEE Trans. Intell. Vehicles*, vol. 5, no. 3, pp. 414–425, Sep. 2020.
[4] D. Fox, W. Burgard, and S. Thrun, "The dynamic window approach to collision avoidance," *IEEE Robot. Autom. Mag.*, vol. 4, no. 1, pp. 23–33, Mar. 1997.
[5] D. Dmitri, S. Thrun, M. Montemerlo, and J. Diebel, "Path planning for autonomous vehicles in unknown semi-structured environments," *Int. J. Robot. Res.*, vol. 29, no. 5, pp. 485–501, 2010.
[6] M. Pivtoraiko and A. Kelly, "Generating near minimal spanning control sets for constrained motion planning in discrete state spaces," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, Aug. 2005, pp. 3231–3237.
[7] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *Int. J. Robot. Res.*, vol. 30, no. 7, pp. 846–894, 2011.
[8] J.-H. Jhang and F.-L. Lian, "An autonomous parking system of optimally integrating bidirectional rapidly-exploring random Trees* and parking-oriented model predictive control," *IEEE Access*, vol. 8, pp. 163502–163523, 2020.
[9] B. Li, K. Wang, and Z. Shao, "Time-optimal maneuver planning in automatic parallel parking using a simultaneous dynamic optimization approach," *IEEE Trans. Intell. Transp. Syst.*, vol. 17, no. 11, pp. 3263–3274, Nov. 2016.
[10] X. Zhang, A. Liniger, A. Sakai, and F. Borrelli, "Autonomous parking using optimization-based collision avoidance," in *Proc. IEEE Conf. Decis. Control (CDC)*, Dec. 2018, pp. 4327–4332.
[11] G. Bitar, A. B. Martinsen, A. M. Lekkas, and M. Breivik, "Two-stage optimized trajectory planning for ASVs under polygonal obstacle constraints: Theory and experiments," *IEEE Access*, vol. 8, pp. 199953–199969, 2020.
[12] J. Zhou, R. He, Y. Wang, S. Jiang, Z. Zhu, J. Hu, J. Miao, and Q. Luo, "Autonomous driving trajectory optimization with dual-loop iterative anchoring path smoothing and piecewise-jerk speed optimization," *IEEE Robot. Autom. Lett.*, vol. 6, no. 2, pp. 439–446, Apr. 2021.
[13] B. Li, T. Acarman, X. Peng, Y. Zhang, X. Bian, and Q. Kong, "Maneuver planning for automatic parking with safe travel corridors: A numerical optimal control approach," in *Proc. Eur. Control Conf. (ECC)*, May 2020, pp. 1993–1998.
[14] A. Eele and A. Richards, "Path-planning with avoidance using nonlinear branch-and-bound optimization," *J. Guid., Control, Dyn.*, vol. 32, no. 2, pp. 384–394, Mar. 2009.
[15] Y. Dong, E. Camci, and E. Kayacan, "Faster RRT-based nonholonomic path planning in 2D building environments using skeleton-constrained path biasing," *J. Intell. Robotic Syst.*, vol. 89, nos. 3–4, pp. 387–401, Mar. 2018.
[16] Z. Sun, D. Hsu, T. Jiang, H. Kurniawati, and J. H. Reif, "Narrow passage sampling for probabilistic roadmap planning," *IEEE Trans. Robot.*, vol. 21, no. 6, pp. 1105–1115, Dec. 2005.
[17] W. Wang, X. Xu, Y. Li, J. Song, and H. He, "Triple RRTs: An effective method for path planning in narrow passages," *Adv. Robot.*, vol. 24, no. 7, pp. 943–962, Jan. 2010.
[18] X. Shu, F. Ni, Z. Zhou, Y. Liu, H. Liu, and T. Zou, "Locally guided multiple Bi-RRT* for Fast path planning in narrow passages," in *Proc. IEEE Int. Conf. Robot. Biomimetics (ROBIO)*, Dec. 2019, pp. 2085–2091.
[19] W. Wang, L. Zuo, and X. Xu, "A learning-based multi-RRT approach for robot path planning in narrow passages," *J. Intell. Robotic Syst.*, vol. 90, nos. 1–2, pp. 81–100, May 2018.
[20] Q. H. Do, S. Mita, and K. Yoneda, "Narrow passage path planning using fast marching method and support vector machine," in *Proc. IEEE Intell. Vehicles Symp.*, Jun. 2014, pp. 630–635.
[21] F. Augugliaro, A. P. Schoellig, and R. D'Andrea, "Generation of collision-free trajectories for a quadrocopter fleet: A sequential convex programming approach," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, Oct. 2012, pp. 1917–1922.
[22] S. Liu, M. Watterson, K. Mohta, K. Sun, S. Bhattacharya, C. J. Taylor, and V. Kumar, "Planning dynamically feasible trajectories for quadrotors using safe flight corridors in 3-D complex environments," *IEEE Robot. Autom. Lett.*, vol. 2, no. 3, pp. 1688–1695, Feb. 2017.
[23] J. A. Reeds and L. A. Shepp, "Optimal paths for a car that goes both forwards and backwards," *Pacific J. Math.*, vol. 145, no. 2, pp. 367–393, Oct. 1990.
[24] 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.
[25] S. A. Nene and S. K. Nayar, "A simple algorithm for nearest neighbor search in high dimensions," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 19, no. 9, pp. 989–1003, Sep. 1997.
[26] A. Wächter and L. T. Biegler, "On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming," *Math. Program.*, vol. 106, no. 1, pp. 25–57, May 2006.
[27] R. He, J. Zhou, S. Jiang, Y. Wang, J. Tao, S. Song, J. Hu, J. Miao, and Q. Luo, "TDR-OBCA: A reliable planner for autonomous driving in free-space environment," 2020, *arXiv:2009.11345*. [Online]. Available: http://arxiv.org/abs/2009.11345
[28] *Apollo Auto*. [Online]. Available: https://github.com/ApolloAuto/apollo

WEITIAN SHENG received the B.S. degree in electronic science and technology from Tongji University, Shanghai, China, in 2013, and the M.S. and Ph.D. degrees in electrical engineering from the University of Michigan, Ann Arbor, MI, USA, in 2015 and 2018, respectively. From 2018 to 2020, he was a Lead Software Engineer with Cadence Design Systems, Inc., San Jose, CA, USA. Since 2021, he has been the Principal Scientist with the Changsha Intelligent Driving Institute, Changsha, Hunan, China. He has authored or coauthored more than 16 articles in peer-reviewed journals or conference proceedings. His research interests include across multiple academic areas, including fast solvers of integral equations in computational electromagnetics, uncertainty quantification methods, RF circuit simulation methods, and trajectory planning methods in autonomous driving. He was a recipient of the Electrical Engineering and Computer Science Departmental Fellowship of the University of Michigan, in 2013.

BAI LI (Member, IEEE) received the B.S. degree from the School of Advanced Engineering, Beihang University, China, in 2013, and the Ph.D. degree from the College of Control Science and Engineering, Zhejiang University, China, in 2018. From November 2016 to June 2017, he visited the Department of Civil and Environmental Engineering, University of Michigan, Ann Arbor, USA, as a Joint Training Ph.D. Student. Prior to teaching with Hunan University, China, he worked with JDX Research and Development Center of Automated Driving, JD Inc., China, from 2018 to 2020, as an Algorithm Engineer. He is currently an Associate Professor with the College of Mechanical and Vehicle Engineering, Hunan University. He has been the first author of nearly 60 journals/conference papers and two books in the community of robotics. His research interests include motion planning and control of automated vehicles. He was a recipient of the International Federation of Automatic Control (IFAC) 2014–2016 Best Journal Paper Prize.

XIANG ZHONG received the B.S. degree from the Department of Electrical Engineering, Hunan University, in 1993, and the M.S. degree from the School of Software, Hunan University, in 2014. He is currently an Assistant Professor with the College of Mechanical and Vehicle Engineering, Hunan University, the President of Hunan Intelligent Transportation Industry Association, and the Deputy Director of Hunan Provincial Key Laboratory of Big Data Research and Applications. His current research interests include intelligent transportation systems, traffic big data analysis, and vehicle road collaboration.