Computer Science and Information Systems 23(1):113–132 https://doi.org/10.2298/CSIS250612004W

ASD-RRT*: An Enhanced Path Planning Algorithm Based on RRT* for Multi-Obstacle Environments

Chao Wang and Wenbin Li

Computer School, Beijing Information Science and Technology University
Beijing 100192, China
wangchao@bistu.edu.cn
emaillwb@163.com

Abstract. The efficiency of sampling-based motion planning brings wide application in autonomous vehicles. The conventional rapidly exploring random tree (RRT) algorithm and its variants have gained significant successes, but there are still challenges for the efficient motion planning in complex and multi-obstacles environments. Conventional sampling methods perform unconstrained sampling across the entire search space, often resulting in suboptimal paths. In this paper, we propose a novel algorithm, Adaptive Sampling and Densification RRT* (ASD-RRT*), for path planning in multi-obstacle environments. Our method extends RRT*-based sampling methods by incorporating adaptive sampling to enhance performance in complex environments. The adaptive sampling approach allows the algorithm to focus on effective regions, reducing sampling of irrelevant points and finding feasible solutions with fewer samples while maintaining the asymptotic optimality of RRT*. Further, we introduce a new optimization method for high-curvature paths and a routing strategy that satisfies vehicle dynamics constraints, aiming to improve path quality. The effectiveness and efficiency of the proposed ASD-RRT* are proved through comparative experiments in different environments. Experimental results demonstrates our method offers a reduction of 55.7% in planning times and 18.7% in path lengths over RRT* in a variety of environments.

Keywords: Motion planning, path planning, RRT*, adaptive sampling, dynamic smoothing.

1. Introduction

Autonomous driving technologies hold promise for reducing road traffic accidents, enhancing traffic efficiency and providing convenient travel solutions [1]. Typical on-board modules in autonomous vehicles include perception, prediction, localization, planning and control [2]. Among these, trajectory planning is crucial as it aims to devise a safe and comfortable path for an autonomous vehicle from its starting position to its destination, considering the environment and constraints. A safe path avoids collisions, while a comfortable path minimizes sudden changes in speed or lane [3]. Numerous researchers have contributed to this field, proposing methods such as curve planning, artificial potential field method, sampling method, model predictive control algorithm and data-driven planning algorithm [4, 5].

Curve planning is widely employed for vehicle lane changing, turning and local trajectory smoothing [6]. While it produces smooth trajectories, it often struggles to explore all feasible paths and may become trapped in local optima, making it less effective in complex environments [7]. The artificial potential field method simulates interactions between objects to plan paths, offering simplicity, intuitiveness, and computational efficiency, making it suitable for real-time obstacle avoidance and path smoothing [8]. However, improperly constructed potential fields can lead to poor path smoothness, coherence, or entrapment in local optima [9]. Sampling-based methods [10, 11] excel in real-time performance and can accommodate various constraints during planning, yet their paths often exhibit zigzag patterns and lack smoothness. Model predictive control(MPC) [12] integrates environmental and vehicle dynamics constraints effectively through well-designed objective functions and constraints, but it relies heavily on on-board sensor data and incurs significant computational overhead. Deep learning-based end-to-end motion planning algorithms [13, 14] map sensor inputs directly to control outputs, mimicking human driving behaviors. Despite this, they struggle with generalizing to extreme scenarios and lack interpretability. Reinforcement learning-based motion planning algorithms [15] learn optimal paths through trial and error interactions with the environment, but they are highly dependent on the design of the reward function and need to train the model in advance [16].

Considering the strengths and weaknesses of these algorithms, sampling-based methods are notable for their computational efficiency, real-time performance, and adaptability to various driving scenarios. However, adjustments to their sampling strategies are necessary to satisfy different constraint conditions. This paper proposes an efficient adaptive sampling ASD-RRT* motion planning algorithm for autonomous global path planning. Initially, the sampling region is determined based on a simplified raster map, and then optimized to obtain a safe, collision-free drivable path. Compared to traditional sampling and optimization-based planning algorithms, the proposed method significantly enhances solution speed by optimizing the sampling region. Relative to the RRT* algorithm, the average computation time reduced by 57.63% and the average path length reduced by 18.7%. This algorithm integrates the RRT* algorithm with control point curve optimization. An initial guidance path is obtained using a graph search algorithm, which subsequently defines the sampling region for the RRT* algorithm. This approach not only ensures the completeness of the RRT* algorithm but also mitigates performance losses due to non-differentiable search spaces.

2. Related Work

Combining sampling-based and optimization-based methods is the preferred strategy for reducing solution time and ensuring high-quality paths in autonomous vehicle path planning. Sampling-based planning algorithms, such as probabilistic road map (PRM) [17] and rapidly-exploring random tree (RRT) [18], select the best path by randomly sampling candidate paths in the environment and evaluating their safety. PRM and its various improvements (Lazy PRM [19], Anytime PRM [20], PRM* [21], Fast Marching PRM [22], etc.) require environmental information to construct the probabilistic roadmap. Consequently, when the surroundings are unknown, finding a feasible solution may be impossible. Additionally, limited sampling density and high connection costs between sampled points reduce planning efficiency, particularly in dense obstacle areas or narrow passages, making it unsuitable for real-time applications in autonomous vehicle scenarios [23].

RRT algorithms are highly efficient and can plan collision-free feasible paths without requiring precise global map information. However, the paths generated by RRT are often tortuous and lack directional guidance in the search. To address these issues, researchers have proposed numerous RRT algorithm improvements for vehicle motion planning. To improve path quality, Karaman et al. [21] proposed the asymptotically optimal RRT* algorithm, which converges to the optimal solution given sufficient time by searching for low-cost parent nodes and adopting pruning optimization strategies. However, the process of finding new parent nodes and rewiring reduces the algorithm's efficiency. Webb et al. [24] addressed the issue of RRT* not meeting autonomous vehicle dynamic constraints by proposing the Kinodynamic RRT* algorithm. This method optimizes the trajectory planned by the RRT* algorithm using controllable linear dynamic systems, resulting in a trajectory that satisfies the dynamic constraints of autonomous vehicles. Nevertheless, there is still room for improvement in convergence speed and optimal path quality. Armstrong et al. [25] proposed AM-RRT*, which combines auxiliary metrics with Euclidean metrics to improve path quality in obstacle-rich environments while maintaining the expected performance of RRT variants. However, the planning performance is not satisfactory, particularly in terms of search time. Sun et al.[26] proposed NAMR-RRT, a neural adaptive motion planning algorithm for robots in dynamic environments. It uses neural network-generated heuristic regions to dynamically guide exploration, improving planning efficiency and reducing trajectory length.

To accelerate the algorithm's convergence speed and reduce search time, bidirectional search strategies are widely employed. Klemm et al. [27] proposed the RRT*-Connect algorithm, which builds search trees with the start and goal points as root nodes. When overlapping sub-nodes are discovered during the search, bidirectional backtracking is initiated, repeatedly computing paths until planning is completed, thereby improving search efficiency. Hussain [28] proposed the IB-RRT* algorithm, which uses a bidirectional tree structure and intelligent sample insertion heuristic algorithm to quickly converge to the optimal solution, enhancing the algorithm's efficiency and quality. Ma et al. [29] proposed the Bi-Risk-RRT, which constructs paths based on bidirectional search sampling strategies and predicts collision risks with dynamic and static obstacles, effectively improving the motion planning efficiency of autonomous vehicles in dynamic environments. However, the algorithm relies on predicting the motion of dynamic obstacles, and if the prediction model is inaccurate, the planned path may be infeasible in practice. Wang et al. [30] proposed the B2U-RRT, which also adopts a bidirectional search strategy and switches to single search mode when encountering the opposite tree to avoid boundary value issues during connection. Similarly, Wang et al. [31] proposed BI-AM-RRT, which optimizes motion planning performance in dynamic environments by introducing auxiliary metrics with larger connection distances. It also employs bidirectional search sampling strategies to reduce search time and proposes a reconnection method based on roots and goals to shorten path length. Yao et al.[32] paper proposes NG-RRT*, a neural-guided path planning algorithm for flexible needle puncture, using 3D U-Net for entry point selection and neural-guided sampling to enhance efficiency. It also introduces an arc trajectory search for smooth paths. Simulations demonstrate improved performance and clinical feasibility.

The positional relationship between the starting point and the goal point affects the sampling region's restriction, which can be optimized by limiting the tree structure's sampling area. For example, Informed-RRT* [33] and Informed RRT*-Connect [34] guide sampling and connection using restricted areas, utilizing heuristic information to accelerate the path search process. N. Li et al.[35] proposed the ABi-RRT algorithm, which employs an adaptive sampling strategy and adjustable step size to enhance search efficiency and integrates a greedy path pruning algorithm and cubic B-spline curves to optimize path smoothness and length in complex 3D environments, achieving a 100% success rate and significantly reduced running time. Y. Su et al.[36] proposed a dynamic path planning method combining improved RRT* and DWA, using APF for initial path generation, adaptive step-size for optimization, and enhanced DWA for obstacle avoidance in dynamic environments. Simulations and experiments show it generates high-quality paths and effectively navigates unknown obstacles. D. Uwacu et al.[37] proposed HAS-RRT, a hierarchical RRT-based motion planning strategy guided by a workspace skeleton, achieving up to 91% runtime reduction and a 30% smaller tree than competitors while maintaining competitive path costs. Inspired by Informed-RRT*, a novel search sampling method based on a adaptive sampling area, i.e., ASD-RRT*, is proposed in this article to further reduces the search time and the path length.

3. Proposed ASD-RRT*

The ASD-RRT* algorithm aims to improve the inefficiencies and tortuous paths characteristic of the classical RRT* algorithm. By introducing adaptive sampling regions and an obstacle recurrence superposition strategy, the algorithm achieves iterative global path planning, effectively reducing the performance loss associated with the undifferentiated treatment of the search space in RRT*. Additional, this algorithm employs a smoothing method that satisfies vehicle dynamic constraints to enhance path smoothness, thereby further improving the quality of the generated paths.

3.1. Algorithm Framework

Algorithm 1 describes the details of ASD-RRT*. The search tree in RRT is defined as $\mathcal{T} = \{n_i \mid i = 1, 2, \cdots, N\}$, where $n_i = [x_i, y_i, t_i, c_i, dx_i]$ represents the information of the ith node, containing node coordinates $[x_i, y_i]$, current time t_i, current node cost c_i, and the parent node number dx_i. P_{start} and P_{goal} represent the planning start and target points, respectively. δ is the adaptive sampling factor, *thr* is the threshold value used when simplifying the map with obstacles. map_s is the simplified raster map, S is the sampling area, $Path_{init}$ and $Path_s$ are the initial bootstrap path and final output path, respectively. Finally, a dynamic smoothing method with dynamics constraints is employed to enhance the quality of the path solution.

Next, the procedures on which the algorithm depends are described. The *InitializeTree*(P_{start}) function initializes the search tree $\mathcal{T}$ with P_{start} as the root node. *SimplifyMap*(*thr*) simplifies the obstacles based on the *thr* and returns the obstacle information *Obs* and the simplified map map_s. *InitialGuidingPath*(map_s, P_{start}, P_{goal}) generates an initial guiding path using graph search concepts. *SamplingArea*($Path_{init}$, δ) creates an adaptive sampling region based on $Path_{init}$ and δ. *ObstacleRecurrencePlan*($\mathcal{T}$, map_s, S, ξ) iteratively overlays obstacles based on the recurrence scale factor ξ and plans the path. *HighCurvatureProcessing*($Path_s$) returns the final $Path_s$ after smoothing.

Algorithm 1 ASD-RRT*

```
Input: $P_{start}, P_{goal}, \delta, thr, map, \xi$
Output: $Path_s$
1: $\mathcal{T} \leftarrow InitializeTree(P_{start})$
2: $Obs, map_s \leftarrow SimplifyMap(thr)$
3: $Path_{init} \leftarrow InitialGuidingPath(map_s, P_{start}, P_{goal})$
4: $S \leftarrow AdaptiveSamplingRegion(Path_{init}, \delta)$
5: $Path_s \leftarrow ObstacleRecurrencePlan(\mathcal{T}, Obs, S, \xi)$
6: $Path_s \leftarrow HighCurvatureProcessing(Path_s)$
7: return $Path_s$
```

3.2. Adaptive Sampling Area

The Rapidly-exploring Random Tree (RRT) algorithm explores the entire map space through continuous sampling and expansion. When the RRT algorithm expands a new vertex, it calculates the distance between that point and the target. If the distance is less than a predetermined threshold, the algorithm has found the target point and stops the search. Otherwise, the search continues. However, this uniform sampling across the entire state space struggles to effectively cover critical regions, leading to repeated examination of certain areas, resulting in wasted computational resources and reduced convergence speed.

Figure 1 illustrates the relationship between map size and sampling efficiency. The RRT and RRT* algorithms perform uniform sampling across the entire map space, causing the time taken by the algorithms to grow almost linearly with increasing map size. After finding an initial path, Informed-RRT* restricts the sampling area to an ellipse, defined by the heuristic space of the currently known path. While the time taken by Informed-RRT* also increases with map size, it is significantly less than that of the RRT and RRT* algorithms.

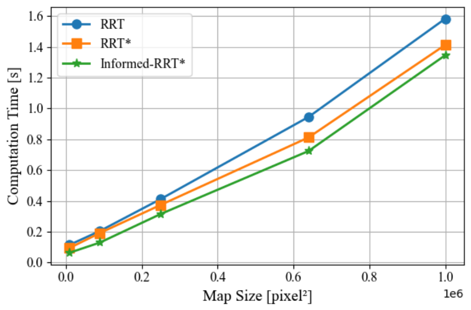

Fig. 1. Relationship between map size and RRT sampling efficiency

To address this issue, this paper proposes an adaptive sampling strategy that introduces initial feasible solutions and sampling regions to improve algorithm efficiency. To minimize the impact of obstacles on sampling efficiency, we simplify the global raster map and introduce an initial guided path based on this simplified map, thereby determining the sampling region for the RRT* algorithm.

The simplification of the raster map involves several steps [38]. First, the occupancy information in the raster map is classified. Then, the obstacle information in the connected raster cells is segmented to ensure that each obstacle region can be individually identified. Finally, obstacles are sorted based on the size of the occupancy data, and obstacles smaller than a predefined simplification threshold are removed. As shown in Figure 2(a), the numbers represent the proportion of the obstacle area to the total map area. Figure 2(b) shows the map after simplifying obstacles that occupy less than 5% of the area.

The process of generating the initial guided path is presented in Algorithm 2. The priority queue Q_f_{open} stores nodes to be expanded, while the closed set S_f_{closed} keeps track of nodes that have already been expanded. The start node T_{start}, based on P_{start}, is initialized with its cost and other relevant information before being inserted into Q_f_{open} to initiate the path search process (Lines 1-3).

Algorithm 2 InitialGuidingPath

```
Input: $map_s$, $P_{start}$, $P_{goal}$
Output: $Path_{init}$
 1: $Q\_f_{open} \leftarrow \emptyset$; $S\_f_{closed} \leftarrow \emptyset$
 2: $T_{start} \leftarrow CreateNode(P_{start})$
 3: $InitializePriorityQueue(Q\_f_{open}, T_{start})$
 4: while not $IsEmpty(Q\_f_{open})$ do
 5:     $x_{current} \leftarrow Dequeue(Q\_f_{open})$
 6:     if $x_{current}$ $IsGoal(P_{goal})$ then
 7:         $Path_{init} \leftarrow ReconstructPath(x_{current})$
 8:         return $Path_{init}$
 9:     end if
10:     $RemoveFromPriorityQueue(Q\_f_{open}, x_{current})$
11:     $InsertIntoSet(S\_f_{closed}, x_{current})$
12:     for $neigh$ in $GenerateNeighbor(x_{current}, map_s)$ do
13:         if $neigh$ $IsIn(S\_f_{closed})$ then
14:             continue
15:         end if
16:         $cost \leftarrow CalculateCost(x_{current}, neigh)$
17:         if $neigh$ $IsIn(Q\_f_{open})$ and $cost \geq neigh.cost$ then
18:             continue
19:         end if
20:         $neigh.cost \leftarrow cost$
21:         $neigh.parent \leftarrow x_{current}$
22:         $Enqueue(Q\_f_{open}, neigh)$
23:     end for
24: end while
25: return $\emptyset$
```

During the loop, while Q_f_{open} is not empty, the node with the lowest cost is dequeued as the current node $x_{current}$. If $x_{current}$ matches the goal point P_{goal}, the goal is considered reached, and the path is constructed by backtracking through the parent information recorded in each node (Lines 5-9). $x_{current}$ is added to S_f_{closed} to mark it as processed and removed from Q_f_{open} to avoid reconsideration (Lines 10-11). For each neighbor of $x_{current}$, if it is already in S_f_{closed}, it is skipped. Otherwise, the cost from the start point to this neighbor is calculated (Lines 12-16). If the neighbor is already in Q_f_{open} and its cost is not lower than the known cost, it is not updated to avoid path degradation (Lines 17-19). Otherwise, its cost and parent information are updated, and it is enqueued into Q_f_{open} (Lines 20-22).

The initial guided path provides a foundation for path planning in real road environments. To ensure that the planned path remains within the lanes, an appropriate expansion

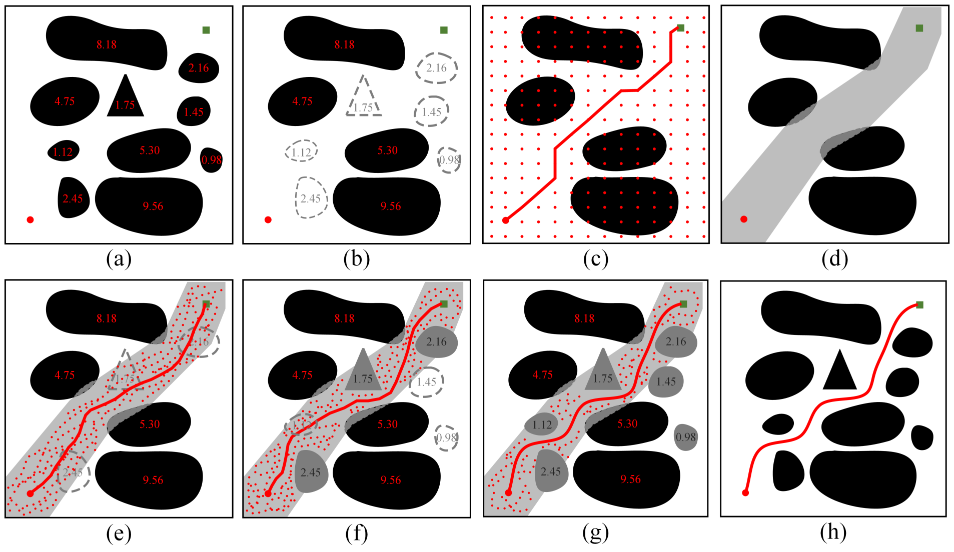

Fig. 2. Algorithm workflow

distance is selected to determine the sampling region. In a structured road environment, the expansion distance can be adjusted based on lane information, ensuring it falls within the drivable area for the autonomous vehicle. The formula for the expansion distance is as follows:

$$S = \frac{E}{\delta} \tag{1}$$

where S is the path expansion distance, E is the edge length in the environment map, and δ is the initial sampling area expansion factor.

Algorithm 3 provides the adaptive sampling region process. Initially, the input path is split into segments, and the initial path is assigned to S (Line 1). Normal vectors and translated segment lists for outward and inward translations are initialized (Line 2). The evaluateRegion(S) function is used to assess the generated sampling region to ensure that a feasible solution can be obtained within the region. The while loop continues until the sampling region is adequately evaluated (Line 3). Within the loop, each line segment

Algorithm 3 AdaptiveSamplingRegion

```
Input: Path_init, $\delta$
Output: $S$
1: $Line_{seg} \leftarrow SplitSeg(Path_{init})$, $S \leftarrow Path_{init}$;
2: $NroVectors \leftarrow \emptyset$, $Trans_{out} \leftarrow \emptyset$, $Trans_{in} \leftarrow \emptyset$;
3: while not $evaluateRegion(S)$ do
4:     for $i = 0$ to $Strlen(Line_{seg}) - 1$ do
5:         $p_1 \leftarrow Line_{seg}[i], p_2 \leftarrow Line_{seg}[i+1]$;
6:         $L \leftarrow \sqrt{(p_1.x - p_2.x)^2 + (p_1.y - p_2.y)^2}$;
7:         $n_x \leftarrow -\frac{(p_1.y - p_2.y)}{L}, n_y \leftarrow \frac{(p_1.x - p_2.x)}{L}$;
8:         $NroVectors.append((n_x, n_y))$;
9:         $x_{out1} \leftarrow p_1.x + \delta \cdot n_x$, $y_{out1} \leftarrow p_1.y + \delta \cdot n_y$;
10:        $x_{out2} \leftarrow p_2.x + \delta \cdot n_x$, $y_{out2} \leftarrow p_2.y + \delta \cdot n_y$;
11:        $Trans_{out}.append((x_{out1}, y_{out1}), (x_{out2}, y_{out2}))$;
12:        $x_{in1} \leftarrow p_1.x - \delta \cdot n_x$, $y_{in1} \leftarrow p_1.y - \delta \cdot n_y$;
13:        $x_{in2} \leftarrow p_2.x - \delta \cdot n_x$, $y_{in2} \leftarrow p_2.y - \delta \cdot n_y$;
14:        $Trans_{in}.append((x_{in1}, y_{in1}), (x_{in2}, y_{in2}))$;
15:    end for
16: end while
17: $Trans_{out}.pop()$, $Trans_{in}$.pop();
18: $S \leftarrow Trans_{out} + reverse(Trans_{in})$;
19: return $S$
```

is processed to calculate the normal vectors and translated segments. For each segment, the current point is dequeued, and the segment endpoints are identified (Lines 4-6). The segment length and normal vector components are computed (Lines 7-8), and the outward and inward translated points are calculated and appended to their respective lists (Lines 9-16). The end of the loop is reached after processing all segments (Line 17). Once the loop condition is satisfied, the last points from the translated segments are removed to avoid duplication (Line 17), and the final sampling region is constructed by combining the outward segments and the reversed inward segments (Line 18). The function returns the constructed sampling region (Line 19).

Figure 2 (a)-(d) illustrates the generation process of the adaptive sampling region. Figure 2 (a) shows the original map, Figure 2 (b) the simplified map, Figure 2 (c) the initial path from the red starting point to the green endpoint, and Figure 2 (d) the sampling region expanded by the sampling factor. The distance between the initial path and the gray boundary is S. To ensure that the width of the sampling region is sufficiently large and to maintain search efficiency, δ is generally set to 4, though this value can be adjusted to modify the search range and time.

3.3. Obstacle Recurrence Stacking Strategy

When dealing with environments with multiple obstacles, the RRT algorithm often generates excessively long paths due to obstacle avoidance. Additionally, the local minimum problem in such environments poses a significant challenge. The adaptive sampling region obtained from the simplified map greatly reduces the sampling space, enabling the

RRT* algorithm to quickly generate a planned path. However, there is a probability that the planned path within the sampling region may intersect with simplified obstacles.

After completing the path planning within the sampling region using RRT*, this paper introduces an obstacle superposition and reproduction strategy. This strategy involves gradually superimposing and reproducing obstacles on the simplified map. An obstacle superposition and reproduction scale factor, ξ, is introduced to control the batches of obstacle superposition and reproduction, ensuring the algorithm adapts to obstacles of different sizes. The maximum ξ value equals the number of simplified obstacles, representing the scenario where obstacles are reproduced one by one. Typically, ξ is set to 2 or 3, dividing the obstacles into two or three parts based on their area for superposition and reproduction, thereby enhancing the algorithm's ability to handle different types of obstacles.

Algorithm 3 summarizes the obstacle superposition and reproduction process. Initially, obstacles are sorted by area, and the map_s is assigned to the current map $Curr_{map}$ (Lines 1-3). The $obsRecurrence(Curr_{map}, obs)$ function is responsible for gradually overlaying and reproducing obstacles onto the simplified map, with path re-planning occurring after each overlay. This process continues until all obstacles are reproduced on the map. In each iteration, the current batch of obstacles is obtained based on ξ (Line 5), added to the current map (Lines 6-8), and the path is re-planned using the updated map (Line 9). The batch counter (batch) is updated after each batch (Line 10) to track the reproduced obstacle batches. Once all obstacles are reproduced, the function returns the final planned path (Line 12).

Algorithm 4 ObstacleRecurrencePlanning

```
Input: $T, map_s, S, \xi$
Output: $Path_s$
1: $Path_s \leftarrow \emptyset; batch \leftarrow 0$
2: $Sorted\_obs \leftarrow$ SortObstacles(map)
3: $Curr\_map \leftarrow map_s$
4: while $batch < \xi$ do
5:     $Obs\_batch \leftarrow$ getBatch($Sorted\_obs, batch$)
6:     for $obs$ in $Obs\_batch$ do
7:         $Curr\_map \leftarrow$ obsRecurrence($Curr\_map, obs$)
8:     end for
9:     $Path_s \leftarrow$ ReplanPath($T, Curr\_map, S$)
10:    $batch \leftarrow batch + 1$
11: end while
12: return $Path_s$
```

The process of successive obstacle superposition and reproduction is illustrated in Figure 2 (e)-(h), with ξ set to 2. Figure 2 (e) shows the initial planning result of RRT* within the sampling region, Figure 2 (f) shows the superposition and reproduction result of obstacles with an area ratio between 1.5% and 5%, Figure 2 (g) shows the result for obstacles with an area ratio below 1.5%, and Figure 2 (h) shows the final planning result of the algorithm. Additionally, when adding new nodes based on the ASD-RRT* algorithm with adaptive sampling, a new node selection method is introduced. The new random nodes must satisfy Equation (2) to be added to the random search tree. This node selection method ensures that the resulting path has better connectivity, improving search efficiency.

$$C_{new} \leq C_{path} - C_{goal} \tag{2}$$

where C_{new} is the current cost of the new node P_{new}, C_{path} is the total cost of the current optimal path, and C_{goal} is the Euclidean distance from the new node to the destination.

3.4. Path Optimization Strategy

Due to the inherent randomness of the RRT* algorithm, the generated paths often exhibit excessive curvature and lack smoothness, leading to instability in the motion of autonomous vehicles and resulting in mechanical wear and tear. To address these issues, this paper introduces a high-curvature dense sampling point method and a dynamic smoothing control point routing strategy that satisfy vehicle dynamics constraints. These methods are used for post-processing the generated paths to create continuous and smooth paths, thereby improving their quality.

A key parameter is the vehicle's maximum lateral acceleration $a_{\text{lat,max}}$, which is typically determined by the vehicle's design and safety considerations. At a specific speed, the vehicle's maximum curvature can be calculated using the following equation:

$$a_{\text{lat,max}} = v^2 k_{\text{max}} \tag{3}$$

where v is the vehicle speed and k_{max} is the maximum curvature. To satisfy the vehicle dynamics constraints and ensure driving safety, the maximum curvature of the trajectory planning path must not exceed the vehicle's maximum curvature, as given by the following equation:

$$k \leq k_{\text{max}} \tag{4}$$

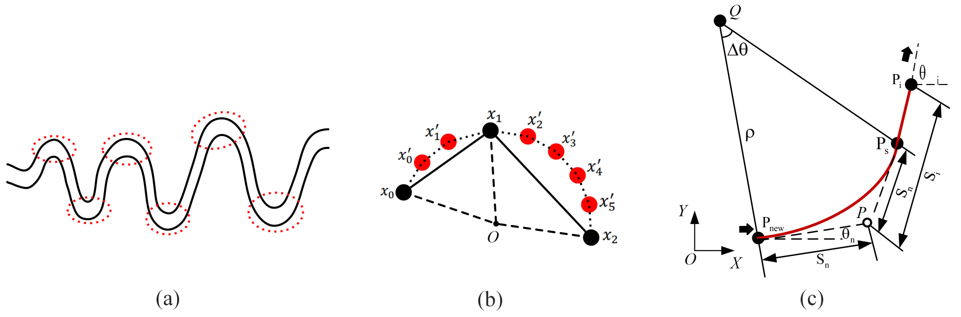

Fig. 3. Path optimization strategy (a) High curvature road; (b) Increasing the density of sampling points; (c) Rewiring strategy for control nodes

Algorithm 5 HighCurvatureProcessing

```
Input: $Path_s$
Output: $Path_s$
1: $a_{lat,max}, v \leftarrow VehicleDynamicsMode()$
2: $k_{max} \leftarrow \frac{v^2}{a_{lat,max}}$
3: for $i = 0$ to $(strlen(Path_s) - 3)$ do
4:     $P_1 \leftarrow Path_s[i]$, $P_2 \leftarrow Path_s[i+1]$, $P_3 \leftarrow Path_s[i+2]$
5:     $f_1' \leftarrow SubtractVectors(P_1, P_2)$
6:     $f_2'' \leftarrow SubtractVectors(P_2, P_3)$
7:     $k \leftarrow \frac{\|f_2''\|}{(1+\|f_1'\|^2)^{3/2}}$
8:     if $k_{max} \leq k$ then
9:         $L \leftarrow Distance(P_1, P_2, P_3)$
10:        $\Delta s \leftarrow \frac{1}{k_{max}}$
11:        $N \leftarrow \lceil \frac{L}{\Delta s} \rceil$
12:        $\Delta t_{in} \leftarrow \frac{1}{N}$
13:        for $j = 1$ to $N$ do
14:            $t \leftarrow j \times \Delta t_{in}$
15:            $P_{den} \leftarrow CubicInterpolate(P_1, P_2, P_3, t)$
16:            $Path_s \leftarrow InsertPointPath(Path_s, P_{den}, i+1)$
17:        end for
18:    end if
19: end for
20: return $Path_s$
```

As shown in Figure 3 (a) and (b), when the path curvature is excessively high, the method of increasing the density of sampling points is employed to smooth the path. The curvature calculation of the discrete points satisfies the following equation:

$$k = \frac{|x'y'' - x''y'|}{(x'^2 + y'^2)^{3/2}} \tag{5}$$

where x' and y' are the first derivatives of the sampling point coordinates x and y, respectively, and x'' and y'' are the second derivatives of the sampling point coordinates. In regions of high curvature, the spacing Δs between densely sampled points needs to be sufficiently small to ensure path smoothness, satisfying the following equation:

$$\Delta s \leq \frac{1}{k_{\text{thread}}} \tag{6}$$

The number of densely sampled points N can be calculated using the following formula:

$$N = \lceil \frac{L}{\Delta s} \rceil \tag{7}$$

where L is the length of the path to be densely sampled.

The densification process for high-curvature segments is presented in Algorithm 5. First, the vehicle dynamics model determines the current speed and maximum lateral acceleration of the vehicle, which are used to calculate the maximum allowable curvature

(Lines 1-2). The algorithm then iterates through the path points, estimating the curvature for each segment formed by three consecutive points (Lines 3-7). If the curvature exceeds the maximum allowable value, the necessary spacing Δs, the number of densification points N, and the interval step Δt_{in} are calculated (Lines 8-12). New sampling points are then generated on the path segment using cubic interpolation and inserted into the densified path (Lines 13-17). The function *CubicInterpolate*(P_1, P_2, P_3, t) generates the interpolation point P_{den} using cubic interpolation, and *InsertPointPath*($Path_s$, P_{den}, $i+1$) inserts it into the specified position of the $Path_s$. Finally, the densified $Path_s$ is returned (Line 20). This process ensures effective densification of sampling points in high-curvature areas while maintaining path smoothness.

Figure 3 (c) illustrates the re-wiring strategy of ASD-RRT* for the densely sampled points after refinement, where P_{new} and P_i represent the newly generated state nodes by the cost-based RRT* and the existing state vertices in the random tree T, respectively. ASD-RRT* introduces a novel rewiring method that incorporates the motion dynamics constraints of the autonomous vehicle in addition to the geometric approach. The rewiring between state nodes P_i and P_{new} must satisfy the following conditions: the extension lines of P_i and P_{new} intersect at point P, and the distance between P_i and P (denoted by S_i) is greater than the distance between P_{new} and P (denoted by S_n). The point P_s is defined as the point between P and P_i, and the distance between $\overline{P \text{ and } P_s}$ is equal to S_n. P_s helps to generate a trajectory between P_{new} and P_i via the arc $\overline{P_{\text{new}}P_s}$ and the line $\overline{P_sP_i}$. The point O represents the center of the arc and ρ represents the curvature. P_s can be considered a virtual node with the same orientation as the node P_i. The angle subtended by the arc is equal to the change in direction from P_{new} to P_i, denoted by $\Delta\theta$. The rewiring process is also divided into an arc phase and a straight line phase.

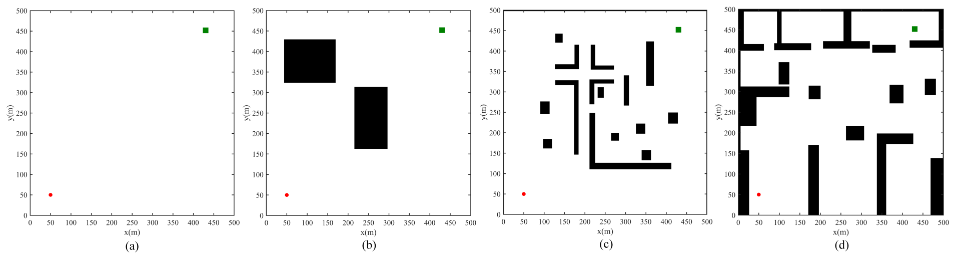

Fig. 4. Environments for the simulations: (a) Black; (b) Simple; (c) Narrow; (d) Closed Trap

4. Simulation and Experiment

To verify the performance of the ASD-RRT* algorithm, this study conducted a comparative analysis of the RRT, RRT*, Informed-RRT*, and ASD-RRT* algorithms across the four scenarios illustrated in Figure 4. The grid map size was 500 × 500 pixels with a resolution of 1 pixel. All algorithms used the red point with coordinates (50, 48) as the starting point and the green rectangle with coordinates (430, 450) as the global target area. The red curves represented the final planned paths. The tree's expansion step was set to 5 pixels, and the pruning radius was set to 10 pixels. The maximum sampling value was set to 3000, if the number of samples exceeded this value, the path planning was considered a failure.

Algorithm performance was evaluated by execution time, final path length, and the average number of iterations. Due to the random nature of the algorithms, the average of 100 simulation experiments was used for evaluation. The simulation platform was MATLAB R2022a, and all algorithms used the min function to find neighboring nodes when expanding the tree nodes. The collision detection program employed the incremental sampling method.

4.1. Simple Environments

The experimental scenario was divided into two parts: one part was the case where there were no obstacles between the start and end points, and the other part was the case where there were very few obstacles, corresponding to maps (a) and (b) in Figure 4. The average path length and the difference from the optimal path were employed to evaluate the algorithm's performance. Since there are no simplifiable obstacles in the maps, the threshold for simplified grid maps in the ASD-RRT* algorithm could be set to any value, which was set to 0 in this instance.

Black Environment Figure 5 illustrates the planning scenarios of the four algorithms in a simple environment. This scenario primarily evaluates the performance loss due to additional computational costs. In an empty scene, the RRT algorithm finds a path by randomly sampling the state space and incrementally building a tree that connects the free space. Due to its randomness, the generated path contains many unnecessary twists. RRT* enhances RRT by optimizing the path and reconnecting surrounding nodes to reduce its twistiness. Informed-RRT* further optimizes path quality by focusing heuristic sampling in more optimal areas. ASD-RRT*, by introducing adaptive sampling regions, effectively reduces invalid samples, resulting in a more direct and smoother path.

Table 2 presents the performance of the algorithms in scenario (a). All four algorithms achieved a planning success rate of 1.00. However, the path planned by ASD-RRT* is nearly optimal, being 7.97% shorter than that of Informed-RRT*, 11.1% shorter than RRT*, and 21.18% shorter than RRT.

Simple Environment Figure 6 presents the planning results of the four algorithms in scenario (b). This scenario primarily evaluates the impact of non-simplifiable obstacles on ASD-RRT*. The RRT algorithm does not consider the global optimality of the path when expanding nodes, resulting in longer and less smooth paths when encountering obstacles. RRT* improves path length to some extent through re-connection operations when expanding nodes in the presence of obstacles. Both Informed-RRT* and ASD-RRT* reduce ineffective random sampling to varying degrees, significantly improving sampling speed and path smoothness.

As shown in Table 3, the success rate of RRT in this scenario is only 0.98, while RRT*, Informed-RRT*, and ASD-RRT* all achieve a success rate of 1.00. The average path length planned by ASD-RRT* is reduced by 21.62% compared to RRT, by 10.13% compared to RRT*, and by 7.61% compared to Informed-RRT*. This demonstrates ASD-RRT*'s ability to generate more effective global paths, resulting in superior path quality.

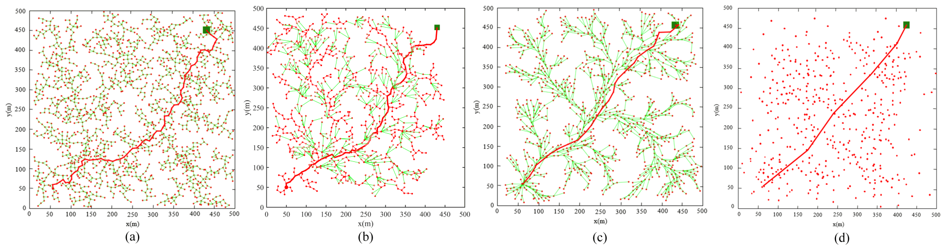

Fig. 5. Performance comparison of four algorithms in environment (a). (a) RRT; (b) RRT*; (c) Informed-RRT*; (d) ASD-RRT*

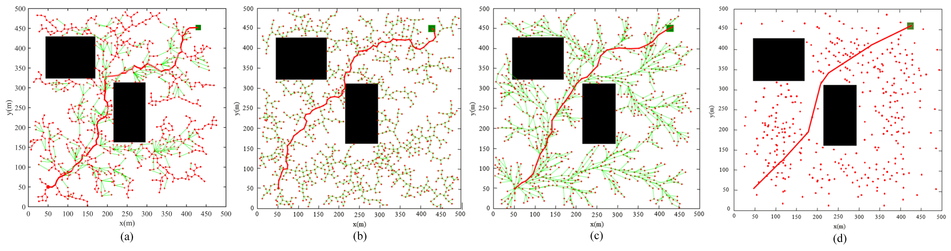

Fig. 6. Performance comparison of four algorithms in environment (a). (a) RRT; (b) RRT*; (c) Informed-RRT*; (d) ASD-RRT*

Table 1. Results for Planning in the Simple Environment (a)

Algorithm	Avg. Length of Path	Success Rate
RRT	718.75	1.00
RRT*	637.24	1.00
Informed-RRT*	615.52	1.00
ASD-RRT*	566.49	1.00

Table 2. Results for Planning in the Simple Environment (b)

Algorithm	Avg. Length of Path	Success Rate
RRT	748.23	0.98
RRT*	652.62	1.00
Informed-RRT*	634.82	1.00
ASD-RRT*	586.48	1.00

4.2. Complex Environments

This section validates the effectiveness of global path planning algorithms in complex scenarios. The experimental setup includes maps with multiple obstacles and narrow passages, corresponding to maps (c) and (d) in Figure 4. The performance of the algorithms is evaluated based on the average number of iterations, average computation time, average length of the final planned path, and planning success rate. In these experiments, the scenes contain numerous obstacles, and the threshold for simplifying the grid map in the ASD-RRT* algorithm is set to 100, while all other parameters remain unchanged.

Narrow Environment The overlay reproduction process of the ASD-RRT* algorithm in environment (c) is illustrated in Figure 7. In the experiments, the obstacle overlay reproduction scale factor is set to 2, meaning that the obstacles are reproduced in two batches based on their area. Obstacles with an area between 50 and 100 are reproduced in the first batch, while those with an area less than 50 are reproduced in the second batch. By restricting the sampling area, invalid sampling is effectively reduced. The staged overlay reproduction of obstacles allows the algorithm to gradually adjust the planned path to avoid obstacles and navigate through narrow passages, thus better adapting to environments with multiple obstacles and narrow paths.

The simulation results of the four algorithms in map (c) are shown in Figure 8. This scenario primarily evaluates the algorithms' ability to navigate through narrow passages with multiple obstacles. The paths planned by RRT and ASD-RRT* are roughly similar, but the RRT path exhibits significantly more sharp turns and is more convoluted, whereas the ASD-RRT* path is noticeably smoother. The paths planned by RRT* and Informed-RRT* are also similar to each other but are significantly longer compared to the path generated by the ASD-RRT* algorithm.

Table 4 presents the performance of the algorithms in the narrow environment (c). The success rates for RRT and Informed-RRT* were only 0.88 and 0.95, respectively, while both RRT* and ASD-RRT* achieved a success rate of 1.00. The average number of iterations for RRT was 3.9 times that of ASD-RRT*, while the average number of iterations for RRT* and Informed-RRT* were close to each other, being 3.4 and 3.1 times that of ASD-RRT*, respectively. In terms of average time spent, RRT was 3.8 times longer than ASD-RRT*, while RRT* and Informed-RRT* were 2.4 and 2.0 times longer than ASD-RRT*, respectively. This demonstrates the high efficiency of the ASD-RRT* algorithm. With regard to the average path length of the four algorithms, ASD-RRT* was reduced by 25.10% compared to RRT, and by 18.70% and 12.46% compared to RRT* and Informed-RRT*, respectively.

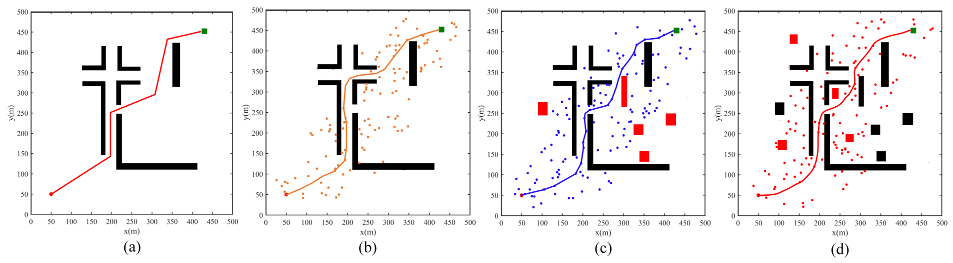

Fig. 7. The planning process of the ASD-RRT* algorithm in map (c). (a) The initial guiding path on the simplified map; (b) The RRT* initial planned path; (c) The result of the first overlay reproduction of large obstacles; (d) The result of the second overlay reproduction of small obstacles

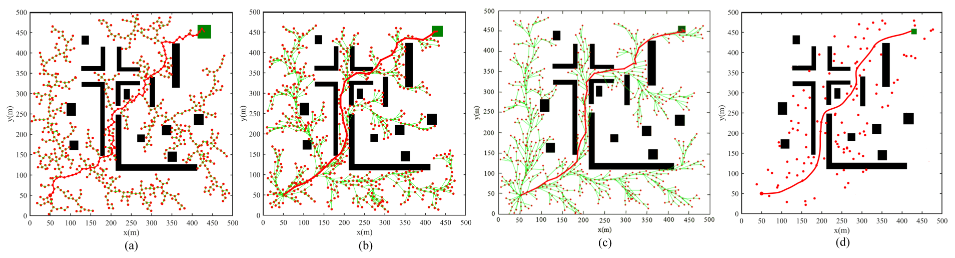

Fig. 8. Performance comparison of four algorithms in environment (a). (a) RRT; (b) RRT*; (c) Informed-RRT*; (d) ASD-RRT*

Table 3. Results for Planning in the Complex Environment (c)

Algorithm	Avg. Iteration	Avg. Time(s)	Avg. Length of Path	Success Rate
RRT	895	0.428	805.82	0.88
RRT*	794	0.274	742.38	1.00
Informed-RRT*	724	0.225	689.45	0.94
ASD-RRT*	232	0.113	603.53	1.00

Closed Environment With Multiple Obstacles The simulation results for the four algorithms in map (d) are depicted in Figure 9. This scenario primarily evaluates the performance of the algorithms in a complex environment with multiple trap-like obstacles. The structures of the four trees are generally similar, but the trajectories generated by RRT and RRT* exhibit more tortuosity and contain more inflection points compared to those produced by Informed-RRT* and ASD-RRT*. In this trap environment, the paths planned by the other three algorithms are notably more convoluted with unnecessary detours. Leveraging the initial guiding path, ASD-RRT* efficiently plans a feasible trajectory in such challenging environments.

The violin plot in Figure 10 illustrates the performance of the four algorithms in map (d). The average number of iterations for RRT was 4.2 times higher than for ASD-RRT*, RRT* was 3.7 times higher, and Informed-RRT* was 3.4 times higher. Regarding average computation time, RRT took three times longer than ASD-RRT*, and RRT* and Informed-RRT* were 2 and 1.8 times longer, respectively. In terms of average path length, ASD-RRT* was 25.56% shorter than RRT, and 22.67% and 18.43% shorter than RRT* and Informed-RRT*, respectively.

In conjunction with the aforementioned experimental analyses, the ASD-RRT* algorithm enhances convergence speed and sampling efficiency. Post-processing of the path results in a smoother global trajectory. Even within complex scenarios featuring multiple obstacles and narrow passages, the algorithm rapidly generates a safe and smooth trajectory that complies with vehicle dynamics. This capability is facilitated by the initial solution, adaptive sampling area, and obstacle superposition reproduction strategy.

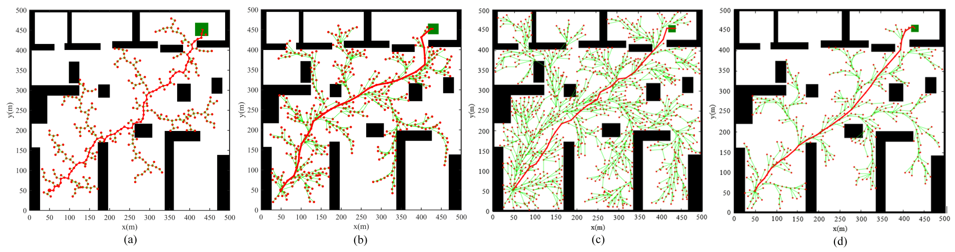

Fig. 9. Performance comparison of four algorithms in environment (a). (a) RRT; (b) RRT*; (c) Informed-RRT*; (d) ASD-RRT*

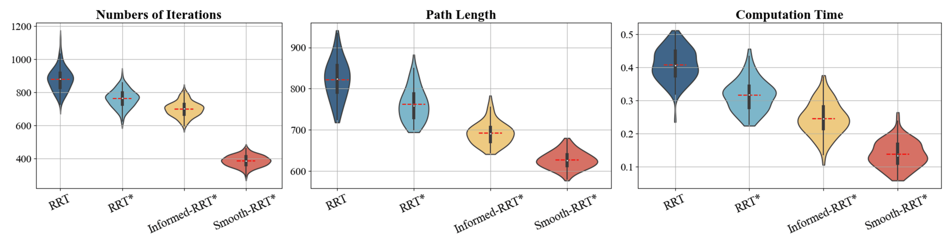

Fig. 10. Performance comparison of four algorithms in environment (a). (a) RRT; (b) RRT*; (c) Informed-RRT*; (d) ASD-RRT*

5. Conclusions

In this paper, we propose an adaptive sampling ASD-RRT* algorithm aimed at addressing challenges such as low search efficiency, slow convergence speed, and convoluted planning paths encountered in traditional RRT* algorithms. ASD-RRT* initiates by simplifying the global raster map, retaining only the larger obstacles. This simplified map guides the initial path generation process. Subsequently, the algorithm expands this initial path within an adaptive sampling area, employing adaptive sampling regions to enhance search efficiency. Concurrently, it iteratively plans around obstacles using the unsimplified global raster map to ensure the path's practicality. The algorithm further integrates a densification technique for sampling in scenarios characterized by high curvature, accompanied by a dynamic smoothing optimization process that complies with dynamic constraints. Such post-processing of the paths produced ensures their continuity and smoothness, thereby improving the efficiency of path tracking. Comprehensive experiments demonstrate that our method outperforms both RRT and RRT* in terms of convergence speed and path length. Even in narrow passages with multiple obstacles, the average number of iterations is only 25.9% of that required by RRT*, and the time spent is reduced by 58.76%. Compared to Informed-RRT*, which also employs a constrained sampling strategy, our method achieves a 56.4% reduction in average iterations, a 47.92% decrease in average computation time, and an 11.19% reduction in the average planned path length. Extensive multi-scenario experimental results validate the effectiveness of the proposed method, significantly enhancing motion planning search time and path quality.

Due to the inherent randomness of the RRT algorithm, the ASD-RRT* algorithm cannot guarantee identical trajectories under identical scenarios. Future research will concentrate on enhancing trajectory stability and aligning trajectories more closely with the global optimal path.

References

1. L. Chen and et al. Milestones in autonomous driving and intelligent vehicles: Survey of surveys. *IEEE Trans. Intell. Veh.*, 8(2):1046–1056, Feb. 2023.
2. Y. Ma, Z. Wang, H. Yang, and L. Yang. Artificial intelligence applications in the development of autonomous vehicles: a survey. *IEEE/CAA J. Autom. Sinica*, 7(2):315–329, March 2020.
3. F. Leon and M. Gavrilescu. A review of tracking and trajectory prediction methods for autonomous driving. *Mathematics*, 9(660), 2021.
4. Y. Huang, J. Du, Z. Yang, Z. Zhou, L. Zhang, and H. Chen. A survey on trajectory-prediction methods for autonomous driving. *IEEE Trans. Intell. Veh.*, 7(3):652–674, Sept. 2022.
5. L. Chen and et al. Milestones in autonomous driving and intelligent vehicles—part ii: Perception and planning. *IEEE Trans. Syst. Man Cybern. Syst.*, 53(10):6401–6415, Oct. 2023.
6. X. Li, Z. Sun, D. Cao, Z. He, and Q. Zhu. Real-time trajectory planning for autonomous urban driving: Framework, algorithms, and verifications. *IEEE/ASME Trans. Mechatronics*, 21(2):740–753, April 2016.
7. 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.*, 23(9):15729–15741, Sept. 2022.
8. F. Bounini, D. Gingras, H. Pollart, and D. Gruyer. Modified artificial potential field method for online path planning applications. In *Proc. IEEE Intell. Veh. Symp. (IV)*, pages 180–185, Los Angeles, CA, USA, 2017.
9. Y. Tian, X. Zhu, D. Meng, X. Wang, and B. Liang. An overall configuration planning method of continuum hyper-redundant manipulators based on improved artificial potential field method. *IEEE Robot. Autom. Lett.*, 6(3):4867–4874, July 2021.
10. W. Lim, S. Lee, M. Sunwoo, and K. Jo. Hierarchical trajectory planning of an autonomous car based on the integration of a sampling and an optimization method. *IEEE Trans. Intell. Transp. Syst.*, 19(2):613–626, Feb. 2018.
11. J.-N. Yuan, L. Yang, X.-F. Tang, and A.-W. Chen. Autonomous vehicle motion planning based on improved rrt* algorithm and trajectory optimization. *Acta Automatica Sinica*, 48(12):2941–2950, 2022.
12. L. Ma, J. Xue, K. Kawabata, J. Zhu, C. Ma, and N. Zheng. Efficient sampling-based motion planning for on-road autonomous driving. *IEEE Trans. Intell. Transp. Syst.*, 16(4):1961–1976, Aug. 2015.
13. A. H. Qureshi, Y. Miao, A. Simeonov, and M. C. Yip. Motion planning networks: Bridging the gap between learning-based and classical motion planners. *IEEE Trans. Robot.*, 37(1):48–66, Feb. 2021.
14. T. Kim, H. Lee, and W. Lee. Physics embedded neural network vehicle model and applications in risk-aware autonomous driving using latent features. In *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, pages 4182–4189, Kyoto, Japan, 2022.
15. F. Ye, S. Zhang, P. Wang, and C.-Y. Chan. A survey of deep reinforcement learning algorithms for motion planning and control of autonomous vehicles. In *Proc. IEEE Intell. Veh. Symp. (IV)*, pages 1073–1080, Nagoya, Japan, 2021.
16. Z. Qin, T.-W. Weng, and S. Gao. Quantifying safety of learning-based self-driving control using almost-barrier functions. In *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, pages 12903–12910, Kyoto, Japan, 2022.

17. L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. H. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Trans. Robot. Autom.*, 12(4):566–580, Aug. 1996.
18. S. M. LaValle. Rapidly-exploring random trees: A new tool for path planning. Technical Report TR 98-11, Iowa State University, 1998.
19. R. Bohlin and L. E. Kavraki. Path planning using lazy prm. In *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, volume 1, pages 521–528, San Francisco, CA, USA, 2000.
20. K. Belghith, F. Kabanza, L. Hartman, and R. Nkambou. Anytime dynamic path-planning with flexible probabilistic roadmaps. In *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, pages 2372–2377, Orlando, FL, USA, 2006.
21. S. Karaman and E. Frazzoli. Sampling-based algorithms for optimal motion planning. *Int. J. Robot. Res.*, 30(7):846–894, June 2011.
22. L. Janson, E. Schmerling, A. Clark, and M. Pavone. Fast marching tree: A fast marching sampling-based method for optimal motion planning in many dimensions. *Int. J. Robot. Res.*, 34(7):883–921, June 2015.
23. M. Fuad and S. Wahyuni. Path planning and smoothing in maze exploration using virtual mobile robot-based modified probabilistic road map. In *Proc. IEEE 8th Inf. Technol. Int. Seminar (ITIS)*, pages 15–20, Surabaya, Indonesia, 2022.
24. D. J. Webb and J. van den Berg. Kinodynamic rrt*: Asymptotically optimal motion planning for robots with linear dynamics. In *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, pages 5054–5061, Karlsruhe, Germany, 2013.
25. D. Armstrong and A. Jonasson. Am-rrt*: Informed sampling-based planning with assisting metric. In *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, pages 10093–10099, Xi'an, China, 2021.
26. Z. Sun, B. Xia, P. Xie, X. Li, and J. Wang. Namr-rrt: Neural adaptive motion planning for mobile robots in dynamic environments. *IEEE Transactions on Automation Science and Engineering*, 22:13087–13100, 2025.
27. S. Klemm and et al. Rrt*-connect: Faster, asymptotically optimal motion planning. In *Proc. IEEE Int. Conf. Robot. Biomimetics (ROBIO)*, pages 1670–1677, Zhuhai, China, 2015.
28. Ahmed Hussain Qureshi and Yasar Ayaz. Intelligent bidirectional rapidly-exploring random trees for optimal motion planning in complex cluttered environments. *Robotics and Autonomous Systems*, 68:1–11, 2015.
29. Han Ma, Fei Meng, Chengwei Ye, Jiankun Wang, and Max Q.-H. Meng. Bi-risk-rrt based efficient motion planning for autonomous ground vehicles. *IEEE Transactions on Intelligent Vehicles*, 7(3):722–733, 2022.
30. Jiankun Wang, Wenzheng Chi, Chenming Li, and Max Q.-H. Meng. Efficient robot motion planning using bidirectional-unidirectional rrt extend function. *IEEE Transactions on Automation Science and Engineering*, 19(3):1859–1868, 2022.
31. Ying Zhang, Heyong Wang, Maoliang Yin, Jiankun Wang, and Changchun Hua. Bi-am-rrt*: A fast and efficient sampling-based motion planning algorithm in dynamic environments. *IEEE Transactions on Intelligent Vehicles*, 9(1):1282–1293, 2024.
32. J. Yao, Z. Fu, Z. Fang, Z. Guo, and F. Jing. Neural-guided rrt*: Learning-based planning of entry point and puncture path for steerable bevel-tip needle insertion. *IEEE Robotics and Automation Letters*, 10(9):9016–9023, 2025.
33. Jonathan D. Gammell, Siddhartha S. Srinivasa, and Timothy D. Barfoot. Informed rrt*: Optimal sampling-based path planning focused via direct sampling of an admissible ellipsoidal heuristic. In *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2997–3004, 2014.
34. Z. Tu, W. Zhuang, Y. Leng, and C. Fu. Accelerated informed rrt*: Fast and asymptotically path planning method combined with rrt*-connect and apf. In *Intelligent Robotics and Applications*, pages 279–292, Singapore, 2023. Springer Nature Singapore.

35. N. Li and S. I. Han. Adaptive bi-directional rrt algorithm for three-dimensional path planning of unmanned aerial vehicles in complex environments. *IEEE Access*, 13:23748–23767, 2025.
36. Y. Su, J. Xin, and C. Sun. Dynamic path planning for mobile robots based on improved rrt* and dwa algorithms. *IEEE Transactions on Industrial Electronics*, pages 1–10, 2025.
37. D. Uwacu, A. Yammanuru, K. Nallamotu, V. Chalasani, M. Morales, and N. M. Amato. Has-rrt: Rrt-based motion planning using topological guidance. *IEEE Robotics and Automation Letters*, 10(6):6223–6230, 2025.
38. Constantin Wellhausen, Joachim Clemens, and Kerstin Schill. Efficient grid map data structures for autonomous driving in large-scale environments. In *2021 IEEE International Intelligent Transportation Systems Conference (ITSC)*, pages 2855–2862, 2021.

Wang Chao is an Associate Professor at the School of Computer Science, Beijing Information Science and Technology University, China. His main research interests are directed to unmanned systems, including object detection, navigation, and decision-making.

Wenbin Li is a Master's student at the School of Computer Science, Beijing Information Science and Technology University, China. His primary research focuses on path planning for unmanned driving.

Received: June 12, 2025; Accepted: October 30, 2025.