drones

MDPI

Article

A 3D Path Planning Algorithm for UAVs Based on an Improved Artificial Potential Field and Bidirectional RRT*

Yijun Huang [1], Hao Li [1], Yi Dai [1], Gehao Lu [1,*] and Minglei Duan [2,*]

[1] School of Information Science and Engineering, Yunnan University, Kunming 650500, China; huangyijun@stu.ynu.edu.cn (Y.H.); lihao707@ynu.edu.cn (H.L.); daiyi@mail.ynu.edu.cn (Y.D.)
[2] Yunnan Communications Investment & Construction Group Co., Ltd., Yunnan University, Kunming 650500, China
* Correspondence: glu@ynu.edu.cn (G.L.); 105617@yciccloud.com (M.D.)

Abstract: Efficient and effective path planning can significantly enhance the task execution capabilities of UAVs in complex environments. This paper proposes an improved sampling-based path planning algorithm, Bi-APF-RRT*, which integrates an Artificial Potential Field (APF) method with a newly introduced repulsive coefficient and incorporates dynamic step size adjustments. To further improve path planning performance, the algorithm introduces strategies such as dynamic goal biasing, target switching, and region-based adaptive sampling probability. The improved Bi-APF-RRT* algorithm effectively controls sampling direction and spatial distribution during the path search process, avoiding local optima and significantly improving the success rate and quality of path planning. To validate the performance of the algorithm, this paper conducts a comparative analysis of Bi-APF-RRT* against traditional RRT* in multiple simulation experiments. Quantitative results demonstrate that Bi-APF-RRT* achieves a 59.6% reduction in average computational time (from 5.97 s to 2.41 s), a 20.6% shorter path length (from 691.56 to 549.21), and a lower average path angle (reduced from 33.28° to 29.53°), while maintaining a 100% success rate compared to 95% for RRT*. Additionally, Bi-APF-RRT* reduces the average number of nodes in the search tree by 45.8% (from 381.17 to 206.5), showcasing stronger obstacle avoidance capabilities, faster convergence, and smoother path generation in complex 3D environments. The results highlight the algorithm's robust adaptability and reliability in UAV path planning.

Keywords: UAV 3D path planning; improved artificial potential field; bidirectional RRT*; dynamic goal biasing; sampling probability strategy

Citation: Huang, Y.; Li, H.; Dai, Y.; Lu, G.; Duan, M. A 3D Path Planning Algorithm for UAVs Based on an Improved Artificial Potential Field and Bidirectional RRT*. *Drones* **2024**, *8*, 760. https://doi.org/10.3390/drones8120760

Academic Editors: Delia Elena Spridon, Razvan Udroiu and Adrian Marius Deaconu

Received: 11 November 2024
Revised: 6 December 2024
Accepted: 13 December 2024
Published: 16 December 2024

1. Introduction

In recent years, with the rapid advancement of technology, UAV (Unmanned Aerial Vehicle) technology has made significant progress [1], and its application range has become increasingly extensive, including target recognition and tracking, traffic monitoring, crop protection, emergency rescue, and various military purposes [2]. Among these applications, one of the key challenges in UAV technology is how to ensure effective obstacle avoidance and precise path planning during mission execution [3]. To address this issue, researchers have proposed various path planning algorithms. Early path planning algorithms are generally classified into local planning and global planning methods [4]. Local planning focuses on real-time obstacle avoidance and path adjustment during UAV flight, while global planning emphasizes the overall design and optimization of the mission path [5]. However, as UAV application scenarios become increasingly complex, single local or global planning methods are often insufficient. Local planning lacks global information, while global planning struggles to adapt to rapid changes in dynamic environments [6]. Thus, the combination of global and local planning methods has gradually become the mainstream trend in UAV path planning. By leveraging the advantages of both, it is possible to achieve flexible local obstacle avoidance while ensuring a globally optimal path, which greatly

enhances the efficiency and success rate of UAV path planning and improves robustness and safety during flight [7].

After years of development, current UAV path planning algorithms can be mainly divided into four categories: graph search, optimization, learning-based, and sampling-based methods [8]. Graph search-based algorithms (such as Dijkstra and A*) typically model the environment as a grid or graph structure and use search algorithms to find the optimal path from the starting point to the target point [9]. These algorithms perform well in low-dimensional and static environments but have high computational costs in high-dimensional or dynamic environments, making it difficult to respond in real-time to complex environmental changes [10]. Moreover, graph search algorithms often require global environmental information, which is difficult to achieve in unknown or partially known environments, limiting their application scope. Optimization-based algorithms search for an optimal path by defining an objective function, suitable for path planning in continuous spaces [11]. Examples include artificial potential field methods, gradient descent methods, as well as intelligent optimization algorithms such as Particle Swarm Optimization (PSO), Ant Colony Optimization (ACO), and Genetic Algorithms (GAs) [12]. These algorithms can generate smooth paths, but in complex environments, they are prone to become trapped in local optima, making it challenging to guarantee global path validity and safety [13]. Additionally, their adaptability to dynamic environments is limited, resulting in insufficient real-time obstacle avoidance capabilities [14]. Learning-based algorithms include machine learning and deep learning methods, such as Deep Q-Network (DQN) and policy gradient methods [15]. These algorithms mainly focus on path planning in unknown or dynamic environments and have strong adaptability to changes in the environment [16]. However, the training process for these algorithms often requires extensive time and computational resources [17]. When facing environmental changes, the models may need frequent updates or retraining, posing challenges for real-time UAV path planning.

In contrast, sampling-based algorithms (e.g., Rapidly exploring Random Trees (RRT) and Probabilistic Roadmaps (PRM)) have the advantage of not requiring complete environmental information and can quickly generate paths [18]. RRT constructs a tree-like path through random sampling, effectively handling high-dimensional, dynamic, and complex environments, making it particularly suitable for obstacle avoidance and path planning in unknown or partially known environments for UAVs [19]. Therefore, sampling-based algorithms are more suitable for UAV path planning in terms of balancing real-time performance and adaptability to complex environments. However, due to the random sampling nature of RRT, the quality of the paths found is often suboptimal, lacking global optimality. Moreover, in dense or dynamic obstacle environments, the local obstacle avoidance capability of RRT is relatively weak, making it challenging to adjust the path flexibly [20]. To address these issues, researchers have proposed improved approaches combining RRT with the Artificial Potential Field (APF) method [21]. APF treats the target point as an attractive source and obstacles as repulsive sources, guiding the path points to automatically avoid obstacles and move towards the target point under the influence of these forces [22]. This effectively compensates for the shortcomings of RRT: the attractive and repulsive forces of APF guide the path generation, improving the smoothness of the path [23]. Additionally, APF provides effective guidance for local obstacle avoidance, helping the path planning avoid getting trapped in obstacle areas [24]. The combination of RRT and APF retains the global path planning capability of RRT while incorporating the local obstacle avoidance advantages of APF, significantly enhancing the reliability and adaptability of UAV path planning [25].

To improve the sampling efficiency of the RRT algorithm, researchers have proposed the enhanced RRT* algorithm. Subsequently, various RRT* variants have been introduced [26]. These include RRT*Smart, which accelerates the convergence of RRT* through intelligent sampling; Quick-RRT*, which optimizes paths by trimming; F-RRT*, which further reduces path costs by selecting new parent nodes; and Informed-RRT*, proposed

by Gammell et al., which improves sampling efficiency by refining the sampling space [27]. Additionally, there are several variants that incorporate target-bias strategies, such as Heuristic-guided RRT* (hRRT*), Potential-field-based RRT* (P-RRT*), Improved Potential-field RRT* (PF-RRT*), and Bidirectional APF-RRT* [28]. Among these, P-RRT* is a representative variant that iteratively guides random sampling states using attractive potential fields, increasing the probability of reaching the target while avoiding collisions. However, these RRT* variants have certain limitations [29]. Firstly, while RRT*Smart, Quick-RRT*, and F-RRT* have made improvements in accelerating convergence and optimizing paths, they may still fall into local optima in complex obstacle environments, with limited obstacle avoidance performance. Informed-RRT* improves efficiency by refining the sampling space, but its obstacle avoidance capabilities in densely cluttered areas are insufficient.

Heuristic-guided RRT* (hRRT*), Potential-field-based RRT* (P-RRT*), and PF-RRT* algorithms incorporate target bias and artificial potential field strategies [30]. However, existing APF integration methods face limitations in handling complex environments, dynamic obstacles, and global optimization. These methods are prone to becoming trapped in local minima, leading to path planning failures or inefficiencies, and they often exhibit oscillations in dense obstacle regions, resulting in unsmooth paths [31]. Additionally, the paths generated by these methods are typically not smooth enough and come with high computational complexity. The algorithms are sensitive to initial positions, which may cause convergence problems, and they often lack global optimization capabilities, leading to suboptimal planned paths [32]. To address these challenges, this paper proposes Bi-APF-RRT*, an improved algorithm integrating an enhanced APF. The proposed algorithm combines the advantages of APF while introducing a series of optimizations to overcome its limitations. By introducing dynamic repulsive force coefficients, the algorithm mitigates the problem of traditional APF being prone to local optima in complex environments, enabling UAVs to adapt more flexibly to environmental changes [33]. Furthermore, the algorithm leverages the bidirectional structure of RRT*, optimizing the path search strategy by conducting simultaneous searches from the start and end points, which significantly enhances search efficiency and reduces planning time. In addition, Bi-APF-RRT* incorporates strategies such as target switching, dynamic step size, and regional sampling probability to further improve adaptability and flexibility [34]. These enhancements effectively prevent the generation of overly lengthy or unsmooth paths, ensuring a more efficient, reliable, and robust path planning process.

In Bi-APF-RRT*, the attractive and repulsive forces of APF are dynamically adjusted to guide the generation of sampling points, making path planning more efficient and precise. Compared to existing APF integration methods, the Bi-APF-RRT* algorithm significantly enhances obstacle avoidance capabilities in dynamic environments and effectively avoids common local optimum issues in path planning, enabling smoother, safer, and more efficient path planning. These optimizations make the Bi-APF-RRT* algorithm more adaptable and practical for 3D path planning on mobile platforms such as UAVs. Additionally, addressing the limitation of previous RRT* variants that primarily conducted simulations in 2D environments, this paper utilizes a 3D random simulation environment to validate the superior performance of the improved algorithm.

The main contributions of this paper are as follows:

1. An improved traditional APF: Dynamic adjustment of repulsive coefficients, introduction of obstacle density factors, and an exponential decay factor make the algorithm more flexible in obstacle avoidance and generate smoother paths in complex environments.
2. Introduction of dynamic target bias: Dynamic adjustment of target sampling probability based on the distance to the target and the number of iterations increases the likelihood of reaching the target, avoiding local optima and improving planning efficiency and convergence speed.

3. Dynamic step size strategy: Adjusting the step size based on the density of surrounding obstacles enhances obstacle avoidance in cluttered areas and speeds up exploration in open spaces, improving efficiency and adaptability.
4. Target switching strategy: Random switching of target points with a certain probability increases the exploration space and flexibility, avoiding local optima and increasing the chance of finding the global optimal path.
5. Regional sampling probability strategy: Adjusting the sampling probability based on the distance to the start and target points concentrates sampling near potential paths, reducing invalid samples, accelerating convergence, and improving planning efficiency.
6. Three-dimensional random environment simulations: Experiments conducted in a 3D random environment demonstrate the actual performance and adaptability of the proposed algorithm.

2. Related Work

*2.1. RRT**

The Rapidly exploring Random Tree (RRT) is a path planning algorithm based on random sampling that can quickly generate feasible paths in complex environments [35]. It constructs nodes through random sampling and incrementally expands a tree structure towards the target region to find a path [36]. The process of RRT path planning is illustrated in Figure 1. The goal of the RRT algorithm is to quickly find a feasible path; however, the generated path is typically not optimal. To address this issue, RRT* was introduced as an improved version of RRT. RRT* optimizes the generated path by reconnecting new nodes with existing nodes, gradually approaching the optimal solution [37]. Compared to RRT, RRT* has a slightly higher time complexity but significantly improves the quality of the resulting path.

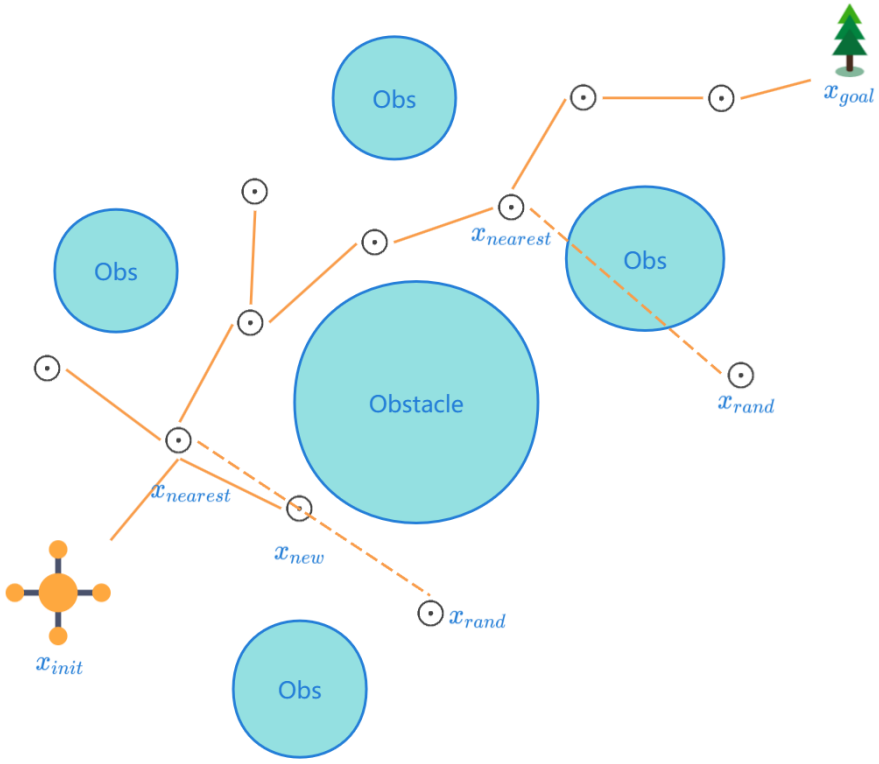

Figure 1. Illustration of the Rapidly exploring Random Tree (RRT) Path Planning Process.

The core idea of the RRT algorithm is to iteratively expand the tree through multiple iterations. First, a point x_{rand} is randomly sampled from the search space with a uniform probability distribution, which serves as a direction for expanding the search tree. After

obtaining the coordinates of the sampled point x_{rand}, the algorithm computes the nearest point x_{nearest} among all vertices in the search tree. Then, using a guiding function `Steer()`, the algorithm moves x_{nearest} towards x_{rand} by a given step size, resulting in a new node x_{new}. After obtaining the new node x_{new}, the path from x_{nearest} to x_{new} is checked for obstacles. If there are no obstacles along this path, the node x_{new} is added to the search tree as a new node. This process repeats iteratively until a path to the target is found or the maximum iteration limit of the algorithm is reached.

The RRT* algorithm is an optimal path planning algorithm, and its pseudocode is shown in Algorithm 1. In RRT*, after a new node x_{new} is generated, the algorithm searches within a certain radius (r_{near}) around the new node for other nodes and selects the node with the lowest path connection cost as the parent node ("choose parent"). This step reduces the cost of reaching the new node, gradually optimizing the path [38]. Furthermore, after the new node x_{new} is connected to the tree, RRT* checks all nodes in the vicinity of x_{new} to determine if connecting these neighboring nodes through x_{new} would reduce the overall path cost. If it does, the algorithm reconnects these neighbor nodes to the new node, further optimizing the path. This process, known as "rewiring," ensures that more nodes can be reached with minimal cost. By continuously adjusting the connections between parent nodes and neighboring nodes, RRT* reduces redundant nodes and unnecessary bends in the path, resulting in a smoother, shorter path with lower overall path cost. The RRT algorithm does not include path optimization steps, and the quality of the generated path depends heavily on random sampling, so the path may not be optimal. In contrast, the quality of the RRT* path improves as the number of iterations increases, and the path gradually converges towards the global optimal solution. Therefore, with a sufficient number of iterations, RRT* can yield a near-optimal path [39].

Algorithm 1: RRT* Algorithm

Input : $\mathcal{X}_{\text{free}}$, $\mathbf{x}_{\text{init}}$, $\mathbf{x}_{\text{goal}}$, $\mathcal{X}_{\text{obs}}$, Δs (Step Size)
Output: A path $\mathcal{T}$ from $\mathbf{x}_{\text{init}}$ to $\mathbf{x}_{\text{goal}}$

$\mathcal{T}$.init();
for $i \leftarrow 1$ **to** n **do**
  $\mathbf{x}_{\text{rand}} \leftarrow \text{Sample}(\mathcal{X}_{\text{free}})$;
  $\mathbf{x}_{\text{nearest}} \leftarrow \text{Nearest}(\mathbf{x}_{\text{rand}}, \mathcal{T})$;
  $\mathbf{x}_{\text{new}} \leftarrow \text{Steer}(\mathbf{x}_{\text{nearest}}, \mathbf{x}_{\text{rand}}, \Delta s)$;
  if $\mathit{CollisionFree}(\mathcal{X}_{\mathit{free}}, \mathbf{x}_{\mathit{nearest}}, \mathbf{x}_{\mathit{new}}, \mathcal{X}_{\mathit{obs}})$ **then**
    $\mathcal{X}_{\text{near}} \leftarrow \text{Near}(\mathbf{x}_{\text{new}}, \mathcal{T}, r_{\text{near}})$;
    $\mathbf{x}_{\text{min}} \leftarrow \mathbf{x}_{\text{nearest}}$;
    foreach $\mathbf{x} \in \mathcal{X}_{\mathit{near}}$ **do**
      if $\mathit{CollisionFree}(\mathcal{X}_{\mathit{free}}, \mathbf{x}, \mathbf{x}_{\mathit{new}}, \mathcal{X}_{\mathit{obs}})$ ***and*** $\mathit{Cost}(\mathbf{x}) + \mathit{Cost}(\mathit{Edge}(\mathbf{x}, \mathbf{x}_{\mathit{new}})) < \mathit{Cost}(\mathbf{x}_{\mathit{min}}) + \mathit{Cost}(\mathit{Edge}(\mathbf{x}_{\mathit{min}}, \mathbf{x}_{\mathit{new}}))$ **then**
        $\mathbf{x}_{\text{min}} \leftarrow \mathbf{x}$;
    $\mathcal{T}$.addNode($\mathbf{x}_{\text{new}}$);
    $\mathcal{T}$.addEdge($\mathbf{x}_{\text{min}}, \mathbf{x}_{\text{new}}$);
    foreach $\mathbf{x} \in \mathcal{X}_{\mathit{near}}$ **do**
      if $\mathit{CollisionFree}(\mathcal{X}_{\mathit{free}}, \mathbf{x}_{\mathit{new}}, \mathbf{x}, \mathcal{X}_{\mathit{obs}})$ ***and*** $\mathit{Cost}(\mathbf{x}_{\mathit{new}}) + \mathit{Cost}(\mathit{Edge}(\mathbf{x}_{\mathit{new}}, \mathbf{x})) < \mathit{Cost}(\mathbf{x})$ **then**
        Update $\mathcal{T}$: Rewire $\mathbf{x}$ to $\mathbf{x}_{\text{new}}$;
    if $\mathbf{x}_{\mathit{new}} = \mathbf{x}_{\mathit{goal}}$ **then**
      return Success;

In Algorithm 1 (pseudocode), the explanations for the related functions and parameters are as follows:

- $\mathcal{X}_{\text{free}}$: The search space, representing the environment where the algorithm explores paths. This space may include obstacles or other constraints.
- x_{init}: The initial point, which is the starting location of the path. The algorithm begins expanding from this point.
- x_{goal}: The goal point, representing the target destination of the path. The algorithm aims to find a path from x_{init} to x_{goal}.
- $\mathcal{T}$: The tree structure, composed of nodes and edges, used to represent the path explored by the RRT* algorithm. The tree grows from x_{init} and gradually expands towards x_{goal}.
- $\mathcal{X}_{\text{obs}}$: The obstacle space, a set of obstacles that must be avoided by the algorithm during path planning.
- x_{rand}: A random sample point, a point randomly generated within the search space $\mathcal{X}_{\text{free}}$, used to guide the direction of tree expansion.
- x_{nearest}: The nearest node, which is the node in the current tree $\mathcal{T}$ that is closest to x_{rand}. It serves as the starting point for expansion.
- x_{new}: The new node, generated by extending x_{nearest} towards x_{rand} by a given step size Δs. This node is added to the tree if the path is collision-free.
- $\mathcal{X}_{\text{near}}$: The set of nearby nodes, a collection of nodes within a radius r_{near} around x_{new}.
- x_{min}: The optimal parent node, which is the node in $\mathcal{X}_{\text{near}}$ with the minimum path cost. It is chosen as the parent node for x_{new}.
- r_{near}: The neighborhood radius, used to define the range of the nearby node set $\mathcal{X}_{\text{near}}$. It typically decreases as the number of sampled points increases.
- Δs: The step size, which limits the maximum distance between x_{nearest} and x_{new} during expansion.
- Sample($\mathcal{X}_{\text{free}}$): The sampling function, which generates a random sample point x_{rand} within the search space $\mathcal{X}_{\text{free}}$.
- Nearest($x_{\text{rand}}, \mathcal{T}$): The nearest node search function, which finds the node x_{nearest} in the tree $\mathcal{T}$ that is closest to x_{rand}.
- Steer($x_{\text{nearest}}, x_{\text{rand}}, \Delta s$): The steering function, which extends from x_{nearest} towards x_{rand}, generating a new node x_{new} within a distance not exceeding Δs.
- CollisionFree($\mathcal{X}_{\text{free}}, x_{\text{nearest}}, x_{\text{new}}$): The collision detection function, which checks whether the path between x_{nearest} and x_{new} is free of obstacles.
- Near($x_{\text{new}}, \mathcal{T}, r_{\text{near}}$): The nearby node search function, which finds all nodes in the tree $\mathcal{T}$ that are within a distance less than r_{near} from x_{new}.
- ChooseParent($\mathcal{X}_{\text{near}}, x_{\text{new}}, x_{\text{nearest}}$): The parent node selection function, which chooses the node in $\mathcal{X}_{\text{near}}$ with the minimum cost as the parent of x_{new}.
- addNode(x_{new}): The node addition function, which adds the new node x_{new} to the tree $\mathcal{T}$.
- addEdge($x_{\text{min}}, x_{\text{new}}$): The edge addition function, which adds an edge between x_{min} and x_{new} in the tree $\mathcal{T}$.
- Rewire($\mathcal{T}, x_{\text{new}}, x$): The rewire function, which attempts to reconnect a neighboring node x to x_{new} in order to further optimize the path.
- Cost(x): The path cost function, which calculates the cumulative cost from the initial point x_{init} to the node x, used for selecting the optimal path.
- Success(): Indicates that a path has been found, and the algorithm has succeeded.

*2.2. Bi-RRT**

The Bi-RRT* algorithm combines the advantages of bidirectional RRT and RRT*, utilizing the bidirectional growth of two trees and path optimization to allow faster convergence towards a near-optimal solution [40]. The pseudocode for the Bi-RRT* algorithm is shown in Algorithm 2. In the Bi-RRT* algorithm, the root nodes of two trees are first initialized at the starting point and the goal point, respectively. A random sample point x_{rand} is

then generated in the search space. When extending in the direction of x_{rand}, one of the two trees—the start tree or the goal tree—is selected as the **active tree** for expansion, while the other tree is designated as the **passive tree**. In the active tree, the nearest node to x_{rand} is identified as x_{nearest}. Using the steering function Steer(), the algorithm extends from x_{nearest} towards x_{rand} by a step size, resulting in a new node x_{new}. Next, the algorithm checks if the path from x_{nearest} to x_{new} is collision-free (i.e., it does not intersect with any obstacles $\mathcal{X}_{\text{obs}}$). If the path is collision-free, the new node x_{new} is added to the active tree. After adding x_{new} to the tree, the active tree performs the "Choose Parent" and "Rewire" operations to optimize the path structure.

After the above steps, once the active tree successfully expands to include x_{new}, the algorithm attempts to connect x_{new} to the passive tree. In the passive tree, the nearest node to x_{new} is identified as x_{connect}. If the path from x_{new} to x_{connect} is collision-free, the algorithm attempts to connect the two trees, forming a candidate path λ_{sol} from the start point to the goal point. The cost of this path is denoted as c_{sol}. If $c_{\text{sol}} < c_{\text{best}}$, the optimal path is updated to λ_{sol}, and the best cost is updated to c_{sol}, indicating that the current best path from the start point to the goal point has been found.

Algorithm 2: Bi-RRT* Algorithm

Input : $\mathbf{x}_{\text{init}}$, $\mathbf{x}_{\text{goal}}$, $\mathcal{X}_{\text{obs}}$, $\mathcal{X}$ (Space), Δs (Step Size)
Output: A path $\mathcal{T}$ from $\mathbf{x}_{\text{init}}$ to $\mathbf{x}_{\text{goal}}$

$\mathcal{V}_1 \leftarrow \{\mathbf{x}_{\text{init}}\}$, $\mathcal{V}_2 \leftarrow \{\mathbf{x}_{\text{goal}}\}$;
$\mathcal{E}_1 \leftarrow \emptyset$, $\mathcal{E}_2 \leftarrow \emptyset$;
$\mathcal{T}_1 \leftarrow (\mathcal{V}_1, \mathcal{E}_1)$, $\mathcal{T}_2 \leftarrow (\mathcal{V}_2, \mathcal{E}_2)$;
$C_{\text{best}} \leftarrow \infty$;
for *iter* $\leftarrow 1$ **to** *MaxIter* **do**
  $\mathbf{x}_{\text{rand}} \leftarrow \text{Sample}(\mathcal{X})$;
  $\mathbf{x}_{\text{nearest}} \leftarrow \text{Nearest}(\mathbf{x}_{\text{rand}}, \mathcal{T}_1)$;
  $\mathbf{x}_{\text{new}} \leftarrow \text{Steer}(\mathbf{x}_{\text{nearest}}, \mathbf{x}_{\text{rand}}, \Delta s)$;
  if *CollisionFree*$(\mathcal{X}, \mathbf{x}_{nearest}, \mathbf{x}_{new}, \mathcal{X}_{obs})$ **then**
    $\mathcal{X}_{\text{near}} \leftarrow \text{Near}(\mathbf{x}_{\text{new}}, \mathcal{T}_1, r_{\text{near}})$;
    $\mathbf{x}_{\text{min}} \leftarrow \mathbf{x}_{\text{nearest}}$;
    foreach $\mathbf{x} \in \mathcal{X}_{near}$ **do**
      if *CollisionFree*$(\mathcal{X}, \mathbf{x}, \mathbf{x}_{new}, \mathcal{X}_{obs})$ ***and***
       $Cost(\mathbf{x}) + Cost(Edge(\mathbf{x}, \mathbf{x}_{new})) < Cost(\mathbf{x}_{min}) + Cost(Edge(\mathbf{x}_{min}, \mathbf{x}_{new}))$
       then
        $\mathbf{x}_{\text{min}} \leftarrow \mathbf{x}$;
    $\mathcal{T}_1$.addNode($\mathbf{x}_{\text{new}}$);
    $\mathcal{T}_1$.addEdge($\mathbf{x}_{\text{min}}, \mathbf{x}_{\text{new}}$);
    foreach $\mathbf{x} \in \mathcal{X}_{near}$ **do**
      if *CollisionFree*$(\mathcal{X}, \mathbf{x}_{new}, \mathbf{x}, \mathcal{X}_{obs})$ ***and***
       $Cost(\mathbf{x}_{new}) + Cost(Edge(\mathbf{x}_{new}, \mathbf{x})) < Cost(\mathbf{x})$ **then**
        Update $\mathcal{T}_1$: Rewire $\mathbf{x}$ to $\mathbf{x}_{\text{new}}$;
    $\mathbf{x}_{\text{connect}} \leftarrow \text{Nearest}(\mathcal{T}_2, \mathbf{x}_{\text{new}})$;
    $(\mathbf{x}_{\text{sol}}, C_{\text{sol}}) \leftarrow \text{ConnectGraphs}(\mathcal{T}_2, \mathbf{x}_{\text{connect}}, \mathbf{x}_{\text{new}})$;
    if $C_{sol} < C_{best}$ **then**
      $C_{\text{best}} \leftarrow C_{\text{sol}}$;
      $\mathcal{T}_{\text{best}} \leftarrow \mathcal{T}_1 \cup \mathcal{T}_2$;
    Swap($\mathcal{T}_1$, $\mathcal{T}_2$);
return Success;

In Algorithm 2 (pseudocode), the explanations for the related functions and parameters are as follows:

- $\mathcal{V}_1$: The set of nodes for the first tree $\mathcal{T}_1$, initialized with the starting point x_{init}.
- $\mathcal{V}_2$: The set of nodes for the second tree $\mathcal{T}_2$, initialized with the goal point x_{goal}.
- $\mathcal{E}_1$: The set of edges for the first tree $\mathcal{T}_1$, initially empty.
- $\mathcal{E}_2$: The set of edges for the second tree $\mathcal{T}_2$, initially empty.
- c_{best}: The cost of the current best path. It is initially set to infinity, allowing updates when a better path is found.
- iter: The current iteration count of the algorithm.
- MaxIter: The maximum number of iterations, used to limit the runtime of the algorithm.
- x_{connect}: The nearest node in the second tree $\mathcal{T}_2$ to the new node x_{new}. It is used to attempt a connection between $\mathcal{T}_1$ and $\mathcal{T}_2$.
- $(c_{\text{sol}}, \lambda_{\text{sol}}) \leftarrow \text{ConnectGraphs}(\mathcal{T}_2, x_{\text{connect}}, x_{\text{new}})$: Attempts to connect trees $\mathcal{T}_1$ and $\mathcal{T}_2$, generating a new path λ_{sol} and calculating its cost c_{sol}.
- c_{sol}: The cost of the new path generated by connecting x_{new} and x_{connect}.
- λ_{sol}: The new path formed by connecting x_{new} and x_{connect}.
- $\text{Cost}(\text{Edge}(x_{\text{new}}, x))$: Calculates the cost of the edge from x_{min} to x_{new}. This is used to determine if the new path cost is better when selecting the optimal parent node.
- $\text{ConnectGraphs}(\mathcal{T}_2, x_{\text{connect}}, x_{\text{new}})$: Attempts to connect trees $\mathcal{T}_1$ and $\mathcal{T}_2$ through nodes x_{connect} and x_{new}, forming a complete path and enabling the bidirectional RRT* connection to find a feasible path from the start to the goal point.
- $\text{Swap}(\mathcal{T}_1, \mathcal{T}_2)$: Swaps the roles of trees $\mathcal{T}_1$ and $\mathcal{T}_2$. This alternating expansion of both trees improves the efficiency of path searching in bidirectional exploration.

*2.3. APF-RRT**

The APF-RRT* (Artificial Potential Field-RRT*) algorithm combines the strengths of the Artificial Potential Field (APF) and RRT*, as shown in Algorithm 3. By introducing potential field methods into the RRT* framework, the algorithm enhances the efficiency of path planning and improves path quality [41]. The APF-RRT* algorithm builds upon the traditional RRT* algorithm by incorporating attractive and repulsive forces to guide the growth of the tree, resulting in more efficient and smoother path planning while accelerating the planning process.

In the APF-RRT* algorithm, the tree is first initialized with the root node at the starting point [42]. During each iteration, a random sample point x_{rand} is drawn from the search space. The sample point is then adjusted using the attractive force, which pulls it towards the goal, and the repulsive force, which pushes it away from obstacles, resulting in an adjusted sample point x_{apf}. The algorithm then finds the nearest node in the tree to this adjusted point, denoted as x_{nearest}, and generates a new node x_{new} in this direction. If the path from x_{nearest} to x_{new} is collision-free, the algorithm proceeds to the "Choose Parent" step, where it selects the optimal parent node from the neighborhood of x_{new}. It also performs the "Rewire" operation, attempting to reconnect nearby nodes to further optimize the path structure. If the new node reaches the goal region, the algorithm outputs the current tree as the solution path.

In Algorithm 3 (pseudocode), the explanations for the related functions and parameters are as follows:

- K_{att}: The attractive force coefficient, used to control the magnitude of the attraction exerted by the goal point on the sample point.
- K_{rep}: The repulsive force coefficient, used to control the magnitude of the repulsion exerted by obstacles on the sample point.
- InfluenceRadius: The radius of influence for the repulsive force. Only obstacles within this radius affect the sample point.
- $\mathbf{F}_{\text{att}}$: The attractive force vector, generated by the attraction exerted by the goal point x_{goal} on the sample point x_{rand}.

- AttractiveForce($x_{\text{nearest}}, x_{\text{goal}}, K_{\text{att}}$): The function for calculating the attractive force, based on the current node x_{nearest}, the goal point x_{goal}, and the attractive force coefficient K_{att}.
- $\mathbf{F}_{\text{rep}}$: The repulsive force vector, generated by the repulsion exerted by obstacles on the sample point x_{rand}.
- RepulsiveForce($x_{\text{nearest}}, \mathcal{X}_{\text{obs}}, K_{\text{rep}}$, InfluenceRadius): The function for calculating the repulsive force, based on the current node x_{nearest}, the set of obstacles $\mathcal{X}_{\text{obs}}$, the repulsive force coefficient K_{rep}, and the influence radius InfluenceRadius.
- $\mathbf{F}_{\text{total}}$: The total force vector, which is the vector sum of the attractive force $\mathbf{F}_{\text{att}}$ and the repulsive force $\mathbf{F}_{\text{rep}}$. This vector controls the adjusted direction of the sample point.
- x_{apf}: The adjusted sample point after applying the artificial potential field, obtained by adding the total force $\mathbf{F}_{\text{total}}$ to the original sample point x_{rand}.

Algorithm 3: APF-RRT* Algorithm

Input : $\mathbf{x}_{\text{init}}$, $\mathbf{x}_{\text{goal}}$, $\mathcal{X}_{\text{obs}}$, $\mathcal{X}$ (Space), Δs (Step Size), K_{att}, K_{rep}, r_{inf} (Influence Radius)

Output: A path $\mathcal{T}$ from $\mathbf{x}_{\text{init}}$ to $\mathbf{x}_{\text{goal}}$

$\mathcal{T}$.init($\mathbf{x}_{\text{init}}$);
for *iter* $\leftarrow$ 1 **to** *MaxIter* **do**
  $\mathbf{x}_{\text{rand}} \leftarrow \text{Sample}(\mathcal{X})$;
  $\mathbf{F}_{\text{att}} \leftarrow \text{AttractiveForce}(\mathbf{x}_{\text{nearest}}, \mathbf{x}_{\text{goal}}, K_{\text{att}})$;
  $\mathbf{F}_{\text{rep}} \leftarrow \text{RepulsiveForce}(\mathbf{x}_{\text{nearest}}, \mathcal{X}_{\text{obs}}, K_{\text{rep}}, r_{\text{inf}})$;
  $\mathbf{F}_{\text{total}} \leftarrow \mathbf{F}_{\text{att}} + \mathbf{F}_{\text{rep}}$;
  $\mathbf{x}_{\text{apf}} \leftarrow \mathbf{x}_{\text{rand}} + \mathbf{F}_{\text{total}}$;
  $\mathbf{x}_{\text{nearest}} \leftarrow \text{Nearest}(\mathbf{x}_{\text{apf}}, \mathcal{T})$;
  $\mathbf{x}_{\text{new}} \leftarrow \text{Steer}(\mathbf{x}_{\text{nearest}}, \mathbf{x}_{\text{apf}}, \Delta s)$;
  if *CollisionFree*$(\mathcal{X}, \mathbf{x}_{nearest}, \mathbf{x}_{new}, \mathcal{X}_{obs})$ **then**
    $\mathcal{X}_{\text{near}} \leftarrow \text{Near}(\mathbf{x}_{\text{new}}, \mathcal{T}, r_{\text{near}})$;
    $\mathbf{x}_{\text{min}} \leftarrow \mathbf{x}_{\text{nearest}}$;
    foreach $\mathbf{x} \in \mathcal{X}_{near}$ **do**
      if *CollisionFree*$(\mathcal{X}, \mathbf{x}, \mathbf{x}_{new}, \mathcal{X}_{obs})$ ***and*** $Cost(\mathbf{x}) + Cost(Edge(\mathbf{x}, \mathbf{x}_{new})) < Cost(\mathbf{x}_{min}) + Cost(Edge(\mathbf{x}_{min}, \mathbf{x}_{new}))$ **then**
        $\mathbf{x}_{\text{min}} \leftarrow \mathbf{x}$;
    $\mathcal{T}$.addNode($\mathbf{x}_{\text{new}}$);
    $\mathcal{T}$.addEdge($\mathbf{x}_{\text{min}}, \mathbf{x}_{\text{new}}$);
    foreach $\mathbf{x} \in \mathcal{X}_{near}$ **do**
      if *CollisionFree*$(\mathcal{X}, \mathbf{x}_{new}, \mathbf{x}, \mathcal{X}_{obs})$ ***and*** $Cost(\mathbf{x}_{new}) + Cost(Edge(\mathbf{x}_{new}, \mathbf{x})) < Cost(\mathbf{x})$ **then**
        Update $\mathcal{T}$: Rewire $\mathbf{x}$ to $\mathbf{x}_{\text{new}}$;
    if $\mathbf{x}_{new} = \mathbf{x}_{goal}$ **then**
      return Success;

*2.4. Informed-RRT**

The Informed-RRT* (Informed Rapidly exploring Random Tree Star) algorithm is an improved version of RRT* that accelerates convergence and enhances path planning efficiency by constraining the sampling space based on the known path [43]. Unlike RRT*, Informed-RRT* dynamically reduces the sampling area after finding an initial feasible path, focusing on regions with lower path costs to gradually optimize the path quality [44]. The pseudocode for Informed-RRT* is shown in Algorithm 4. In the Informed-RRT* algorithm,

the tree is first initialized with the root node at the starting point, and the cost of the optimal path is set to infinity. During each iteration, if a feasible path already exists (i.e., the cost of the found path is less than infinity), an ellipsoid S is generated, with the start point and goal point as its foci and the semi-major axis equal to the current best path cost [45]. The sampling is then restricted to this ellipsoid to focus on more promising regions. Otherwise, sampling is performed over the entire space [46]. The algorithm then finds the nearest node in the tree to the sampled point, denoted as x_{nearest}, and extends towards the sampled point by a step size to generate a new node x_{new}. If the path from x_{nearest} to x_{new} is collision-free, the algorithm proceeds with the "Choose Parent" operation to connect the new node to the optimal parent node, followed by the "Rewire" operation to attempt reconnection with neighboring nodes for further path optimization [47]. If the new node reaches the goal region, the optimal path is updated, and the algorithm completes.

Algorithm 4: Informed-RRT* Algorithm

Input : $\mathbf{x}_{\text{init}}$, $\mathbf{x}_{\text{goal}}$, $\mathcal{X}_{\text{obs}}$, $\mathcal{X}$ (Space), Δs (Step Size)
Output: A path $\mathcal{T}$ from $\mathbf{x}_{\text{init}}$ to $\mathbf{x}_{\text{goal}}$

$\mathcal{T}$.init($\mathbf{x}_{\text{init}}$);
$C_{\text{best}} \leftarrow \infty$;
for *iter* $\leftarrow 1$ **to** *MaxIter* **do**
  if $C_{best} < \infty$ **then**
    $\mathcal{S} \leftarrow$ GenerateEllipsoid($\mathbf{x}_{\text{init}}, \mathbf{x}_{\text{goal}}, C_{\text{best}}$);
    $\mathbf{x}_{\text{rand}} \leftarrow$ SampleInformed($\mathcal{S}$);
  else
    $\mathbf{x}_{\text{rand}} \leftarrow$ Sample($\mathcal{X}$);
  $\mathbf{x}_{\text{nearest}} \leftarrow$ Nearest($\mathbf{x}_{\text{rand}}, \mathcal{T}$);
  $\mathbf{x}_{\text{new}} \leftarrow$ Steer($\mathbf{x}_{\text{nearest}}, \mathbf{x}_{\text{rand}}, \Delta s$);
  if *CollisionFree*($\mathcal{X}, \mathbf{x}_{nearest}, \mathbf{x}_{new}, \mathcal{X}_{obs}$) **then**
    $\mathcal{X}_{\text{near}} \leftarrow$ Near($\mathbf{x}_{\text{new}}, \mathcal{T}, r_{\text{near}}$);
    $\mathbf{x}_{\text{min}} \leftarrow \mathbf{x}_{\text{nearest}}$;
    foreach $\mathbf{x} \in \mathcal{X}_{near}$ **do**
      if *CollisionFree*($\mathcal{X}, \mathbf{x}, \mathbf{x}_{new}, \mathcal{X}_{obs}$) ***and*** $Cost(\mathbf{x}) + Cost(Edge(\mathbf{x}, \mathbf{x}_{new})) < Cost(\mathbf{x}_{min}) + Cost(Edge(\mathbf{x}_{min}, \mathbf{x}_{new}))$ **then**
        $\mathbf{x}_{\text{min}} \leftarrow \mathbf{x}$;
    $\mathcal{T}$.addNode($\mathbf{x}_{\text{new}}$);
    $\mathcal{T}$.addEdge($\mathbf{x}_{\text{min}}, \mathbf{x}_{\text{new}}$);
    foreach $\mathbf{x} \in \mathcal{X}_{near}$ **do**
      if *CollisionFree*($\mathcal{X}, \mathbf{x}_{new}, \mathbf{x}, \mathcal{X}_{obs}$) ***and*** $Cost(\mathbf{x}_{new}) + Cost(Edge(\mathbf{x}_{new}, \mathbf{x})) < Cost(\mathbf{x})$ **then**
        Update $\mathcal{T}$: Rewire $\mathbf{x}$ to $\mathbf{x}_{\text{new}}$;
    if $\mathbf{x}_{new} = \mathbf{x}_{goal}$ **then**
      $C_{\text{best}} \leftarrow$ Cost($\mathbf{x}_{\text{goal}}$);
return Success;

In Algorithm 4 (pseudocode), the explanations for the related functions are as follows:

- GenerateEllipsoid($x_{\text{init}}, x_{\text{goal}}, c_{\text{best}}$): This function generates an ellipsoid based on the current start point x_{init}, goal point x_{goal}, and the current best path cost c_{best}. The ellipsoid constrains the sampling space, focusing the sampling on regions that are more likely to contain the optimal path.

- SampleInformed(S): This function generates a random sample point x_{rand} within the ellipsoid S, improving the efficiency of the sampling process by limiting it to the most promising area.

2.5. Three-Dimensional Environment Setup

Unmanned aerial vehicles (UAVs) often face complex environments during flight. To better evaluate the performance of path planning algorithms, a randomly generated 3D simulation environment can effectively mimic real-world scenarios [48]. The 3D environment in this paper is described using the exponential function in Equation (1):

$$Z(x,y) = \sum_{i=1}^{n} h_i \exp\left(-\left(\frac{x - x_{ic}}{x_{si}}\right)^2 - \left(\frac{y - y_{ic}}{y_{si}}\right)^2\right) \tag{1}$$

where:

- $Z(x,y)$: The terrain height at the coordinates (x,y).
- n: The number of hills in the environment.
- h_i: The height of the i-th hill.
- (x_{ic}, y_{ic}): The center coordinates of the i-th hill.
- x_{si}, y_{si}: The standard deviations in the x and y directions, controlling the spread of the hill.

The height contribution of each hill is described by a 2D Gaussian distribution function. By adjusting the parameters h_i, (x_{ic}, y_{ic}), x_{si}, and y_{si}, hills of different heights, positions, and spread extents can be generated. The final terrain height at any point (x,y) is the cumulative sum of the height contributions from all the hills, forming a complex 3D surface. The number of hills n, the center positions (x_{ic}, y_{ic}), the heights h_i, and the spread parameters x_{si} and y_{si} are randomly generated, allowing different 3D terrains to be created in each run. An example of such a 3D terrain is shown in Figure 2, which is used for path planning and testing.

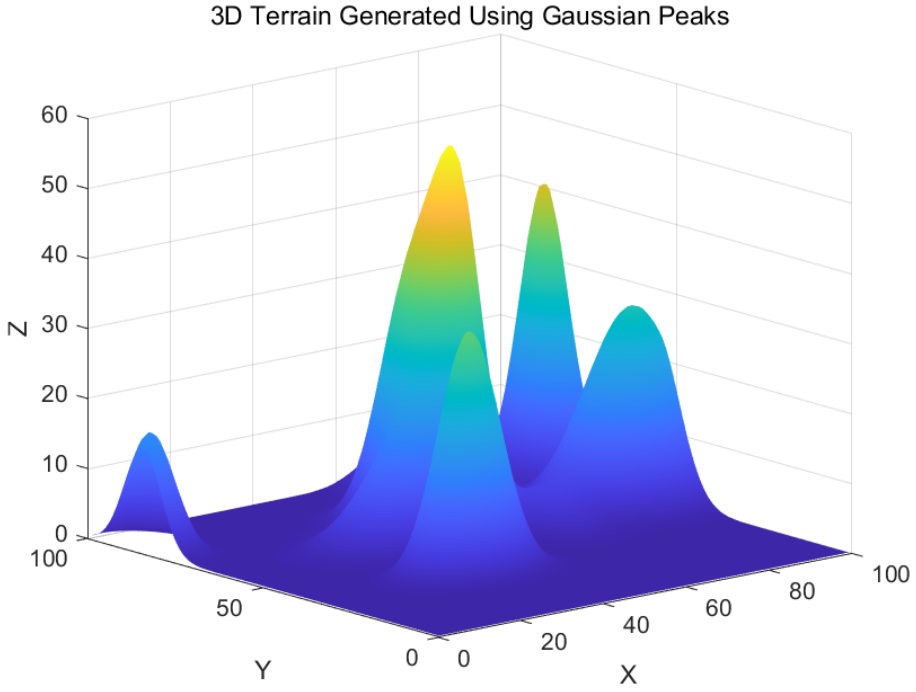

Figure 2. Randomly Generated Mountain Peaks in 3D.

3. Research Methods

3.1. Improved Artificial Potential Field (APF)

The traditional Artificial Potential Field (APF) method is a classical path planning approach widely used for obstacle avoidance and target tracking in mobile devices such as robots and UAVs [49]. The core idea of APF is to simulate a virtual "force field" using an attractive-repulsive force model: the target point generates an attractive force that guides the robot toward the goal, while obstacles produce a repulsive force that pushes the robot away to avoid collisions. However, the traditional APF method has limitations in complex environments; it tends to become trapped in local minima, exhibits oscillation in areas with dense obstacles, and struggles to converge near the goal. To address these issues, this paper proposes an improved APF method. First, the repulsive force coefficient is dynamically adjusted, allowing the repulsive strength to vary based on the complexity of the environment, thereby avoiding excessive repulsion. Second, an obstacle density awareness mechanism is introduced to enhance the algorithm's responsiveness in areas with dense obstacles. Finally, a distance attenuation mechanism is incorporated to reduce the impact of repulsive forces when far from obstacles, ensuring a smoother and more stable path. With these enhancements, the proposed algorithm demonstrates better adaptability and convergence in complex scenarios, resulting in higher-quality path planning outcomes.

The improved Artificial Potential Field (APF) algorithm proposed in this paper dynamically adjusts the repulsive force coefficient, making path planning more flexible in different environments. When approaching the target, the algorithm reduces the influence of the repulsive force, mitigating oscillation caused by excessive repulsion, allowing the robot to more easily reach the goal area. Conversely, when the robot is far from the target or in regions with dense obstacles, the repulsive force strength is increased to enhance obstacle avoidance, ensuring safety.

The pseudocode of the algorithm is shown in Algorithm 5. The key functions are defined as follows:

- AttractionForce $(x_{\text{current}}, x_{\text{goal}}, K_{\text{att}})$: Computes the attractive force towards the goal.
- RepulsionDirection $(x_{\text{current}}, x_{\text{obs}})$: Computes the direction of the repulsive force from obstacles.
- CalculateObstacleDensity $(x_{\text{current}}, x_{\text{obstacle}})$: Calculates the density of obstacles around the current position.

The main procedure of the algorithm is as follows: First, the attractive and repulsive forces acting on the robot are calculated. Then, the repulsive force coefficient is dynamically adjusted based on the distance to the target and the obstacle density around the current position. The final resultant force is determined by combining the attractive force and the adjusted repulsive force, ensuring that the robot can safely and efficiently reach the target in complex environments. By dynamically adjusting the repulsive force, the algorithm adapts to changes in the environment, avoids becoming trapped in local minima, and generates a smoother, more optimized path. This significantly enhances the adaptability and performance of the APF algorithm in complex scenarios.

In the improved Artificial Potential Field (APF) algorithm, the magnitude and direction of the repulsive force are calculated by segmenting and summing the influences of obstacles. This method calculates the total repulsive force by integrating the effects from multiple obstacles, combining forces from different directions into a single resultant force, which allows the robot to better avoid obstacles. The total repulsive force $\mathbf{F}_{\text{rep}}$ is the cumulative sum of all obstacle influences, as shown in Equation (2), where N represents the number of obstacles, and M_i is the influence coefficient of the i-th obstacle.

$$\mathbf{F}_{\text{rep}} = \sum_{i=1}^{N} M_i \sum_{j=1}^{2} \left(F_{\text{rep1}} + F_{\text{rep2}} \right) \cdot \hat{e}_{\text{rep}} \tag{2}$$

The components F_{rep1} and F_{rep2} represent two parts of the repulsive force, calculated by Equations (3) and (4), respectively. In these equations:

- K_r: The dynamic repulsive force coefficient, used to control the magnitude of the repulsive force.
- d_{obs}: The Euclidean distance between the current point and the obstacle.
- $R_{\text{influence}}$: The influence radius of the repulsive force; beyond this radius, the repulsive force becomes zero.
- d_{goal}: The distance between the current point and the goal.
- n: The linear control factor of the repulsive force, used to adjust the growth rate of the repulsive force.

$$\mathrm{F}_{\text{rep1}} = K_r \left(\frac{1}{d_{\text{obs}}} - \frac{1}{R_{\text{influence}}} \right) \left(\frac{d_{\text{goal}}^n}{d_{\text{obs}}^2} \right) \tag{3}$$

$$\mathrm{F}_{\text{rep2}} = \frac{n}{2} \cdot K_r \left(\frac{1}{d_{\text{obs}}} - \frac{1}{R_{\text{influence}}} \right) \left(\frac{d_{\text{goal}}^{n-1}}{d_{\text{obs}}^2} \right) \tag{4}$$

Algorithm 5: Improved APF Algorithm

Input : $\mathbf{x}_{\text{current}}$, $\mathbf{x}_{\text{goal}}$, $\mathcal{X}_{\text{obs}}$, K_{att}, K_{rep}, r_{inf} (Influence Radius)
Output: Total force $\mathbf{F}_{\text{total}}$

// Calculate Attraction Force
$\mathbf{F}_{\text{att}} \leftarrow \text{AttractionForce}(\mathbf{x}_{\text{current}}, \mathbf{x}_{\text{goal}}, K_{\text{att}})$;
// Calculate Repulsion Force
$\mathbf{F}_{\text{rep}} \leftarrow 0$;
foreach $\mathbf{x}_{obs} \in \mathcal{X}_{obs}$ **do**
  $d_{\text{obs}} \leftarrow \text{Distance}(\mathbf{x}_{\text{current}}, \mathbf{x}_{\text{obs}})$;
  if $d_{obs} < r_{inf}$ **then**
    $\mathbf{F}_{\text{rep1}} \leftarrow K_{\text{rep}} \times \left(\frac{1}{d_{\text{obs}}} - \frac{1}{r_{\text{inf}}} \right) \times \left(\frac{\text{Distance}(\mathbf{x}_{\text{current}}, \mathbf{x}_{\text{goal}})^n}{d_{\text{obs}}^2} \right)$;
    $\mathbf{F}_{\text{rep2}} \leftarrow \frac{n}{2} \times K_{\text{rep}} \times \left(\frac{1}{d_{\text{obs}}} - \frac{1}{r_{\text{inf}}} \right) \times \left(\frac{\text{Distance}(\mathbf{x}_{\text{current}}, \mathbf{x}_{\text{goal}})^{n-1}}{d_{\text{obs}}^2} \right)$;
    $\mathbf{F}_{\text{rep}} \leftarrow \mathbf{F}_{\text{rep}} + (\mathbf{F}_{\text{rep1}} + \mathbf{F}_{\text{rep2}}) \times \text{RepulsionDirection}(\mathbf{x}_{\text{current}}, \mathbf{x}_{\text{obs}})$;
// Dynamic Adjustment of Repulsion Coefficient
$\rho_{\text{density}} \leftarrow \text{CalculateObstacleDensity}(\mathbf{x}_{\text{current}}, \mathcal{X}_{\text{obs}})$;
if $Distance(\mathbf{x}_{current}, \mathbf{x}_{goal}) < r_{goal}$ **then**
  $K_{\text{rep\_adjusted}} \leftarrow K_{\text{rep}} \times (1 - \beta \times \rho_{\text{density}}) \times \text{decayFactor}$;
else
  $K_{\text{rep\_adjusted}} \leftarrow K_{\text{rep}} \times (1 + \alpha \times \rho_{\text{density}}) \times \text{decayFactor}$;
// Calculate Total Force
$\mathbf{F}_{\text{total}} \leftarrow \mathbf{F}_{\text{att}} + K_{\text{rep\_adjusted}} \times \mathbf{F}_{\text{rep}}$;
return $\mathbf{F}_{\text{total}}$;

We introduce a dynamically adjusted repulsive force coefficient K_r in the two components of the repulsive force to allow the robot to respond to different repulsive effects at various positions. When the robot approaches the goal, the repulsive force is reduced, as described in Equation (5). When the robot is far from the goal, the repulsive force is increased, as described in Equation (6). In these equations:

- K_r: The initial repulsive force coefficient.
- β: The weakening factor for obstacle density within the goal region, used to reduce the strength of the repulsive force.
- α: The strengthening factor for obstacle density outside the goal region, used to increase the strength of the repulsive force.

- ρ_{density}: The density of obstacles at the current position.

$$\mathrm{K}_r^{\text{adjusted}} = K_r\left(1 - \beta \cdot \rho_{\text{density}}\right) \cdot \text{decayFactor} \tag{5}$$

$$\mathrm{K}_r^{\text{adjusted}} = K_r\left(1 + \alpha \cdot \rho_{\text{density}}\right) \cdot \text{decayFactor} \tag{6}$$

The **decayFactor** represents a factor for the exponential decay of the repulsive force. The purpose of introducing the decay factor is to gradually reduce the repulsive force coefficient as the iteration count increases. This helps prevent excessive force interference in later stages of the algorithm, thereby reducing oscillations. The calculation of the decay factor is shown in Equation (7), where **iter** represents the current iteration number.

$$\text{decayFactor} = e^{-0.01 \cdot \text{iter}} \tag{7}$$

The unit vector $\hat{e}_{\text{rep}}$ indicates the direction from the obstacle to the current position, which is also the direction of the repulsive force. The calculation of $\hat{e}_{\text{rep}}$ is given by Equation (8):

- $\vec{P}_{\text{current}}$: The position vector of the current location.
- $\vec{P}_{\text{obs}}$: The position vector of the obstacle.

The term $\|\vec{P}_{\text{current}} - \vec{P}_{\text{obs}}\|$ represents the distance between the current position and the obstacle.

$$\hat{e}_{\text{rep}} = \frac{\vec{P}_{\text{current}} - \vec{P}_{\text{obs}}}{\|\vec{P}_{\text{current}} - \vec{P}_{\text{obs}}\|} \tag{8}$$

3.2. Introduction of Dynamic Goal Biasing

In classical RRT and RRT* algorithms, the sample points (random samples) are typically distributed randomly throughout the entire search space [50]. However, purely random sampling has a significant drawback: when the target region is far away or the environment is complex, random sampling can lead to low algorithm efficiency. A large number of samples may be required to successfully connect to the target, resulting in increased computation time and resource consumption. To address this issue, this paper introduces a **Dynamic Goal Biasing** strategy.

This strategy dynamically adjusts the sampling point generation process during path planning, making the samples more likely to be distributed closer to the target. The core idea is to introduce two control coefficients, α_1 and β_1, which dynamically adjust the goal sampling rate based on the distance between the current position and the target, as well as the number of iterations [51]. Specifically, when the distance to the target is large, the algorithm favors random exploration; as the robot gets closer to the target, the algorithm biases the samples toward the goal, increasing the probability of connecting to the target. The pseudocode is shown in Algorithm 6. By introducing dynamic goal biasing, the algorithm can more effectively guide the sample points toward the target, reducing excessive exploration in irrelevant areas and significantly improving the efficiency and convergence speed of path planning.

The dynamic goal biasing formula can generally be adjusted based on the distance and iteration count, as shown in Equation (9):

$$\mathrm{P}_{\text{goal}} = P_{\text{initial}} \cdot e^{-\alpha_1 \cdot d} \cdot e^{-\beta_1 \cdot k} \tag{9}$$

where

- P_{goal}: The probability of sampling directly towards the goal in the current iteration.
- P_{initial}: The initial goal sampling probability, set as a constant.
- α_1: The coefficient controlling the decrease in goal sampling probability based on the distance d.

- β_1: The coefficient controlling the decrease in goal sampling probability based on the iteration count k.
- d: The Euclidean distance between the current sample point and the goal point.
- k: The current iteration count.

As the node x_{new} gets closer to the goal, the distance d decreases, making $e^{-\alpha_1 \cdot d}$ approach 1, thereby increasing the probability of sampling towards the goal. With an increasing iteration count k, the term $e^{-\beta_1 \cdot k}$ gradually decreases, reducing the goal sampling probability and adding more randomness to the sampling process, helping to avoid local minima.

Algorithm 6: Improved Dynamic Goal Bias

Input : $\mathbf{x}_{\text{current}}$, $\mathbf{x}_{\text{goal}}$, P_{initial}, α_1, β_1, MaxIter
Output: Adjusted target sampling probability P_{goal}

Initialize Goal Bias Parameters:;
$P_{\text{goal}} \leftarrow P_{\text{initial}}$ `// Initial target sampling probability`
for *iter* $\leftarrow 1$ **to** *MaxIter* **do**
  $d \leftarrow \text{Distance}(\mathbf{x}_{\text{current}}, \mathbf{x}_{\text{goal}})$ `// Calculate distance to the goal`
  $P_{\text{goal}} \leftarrow P_{\text{initial}} \times \exp(-\alpha_1 \times d) \times \exp(-\beta_1 \times \text{iter})$ `// Adjust sampling probability`
  if $rand() < P_{goal}$ **then**
    $\mathbf{x}_{\text{rand}} \leftarrow \mathbf{x}_{\text{goal}}$ `// Bias towards the goal point`
  else
    $\mathbf{x}_{\text{rand}} \leftarrow \text{Sample}(\mathcal{X})$ `// Sample a random point in space`

3.3. Dynamic Step Size

In classical RRT and RRT* algorithms, the step size is usually fixed. However, a fixed step size has significant limitations in path planning, especially in complex environments. Specifically, a fixed step size lacks flexibility and is difficult to adapt to environmental changes. In open areas, a smaller step size results in lower exploration efficiency; in cluttered environments, a larger step size may cause insufficient path resolution, making it difficult to avoid obstacles and leading to issues such as unsmooth paths and excessive resource consumption.

Therefore, in dynamic or cluttered environments, using a fixed step size compromises both efficiency and precision, making it unsuitable for optimized path planning. To address this issue, the traditional fixed step size is replaced with a dynamic step size, as shown in the pseudocode. The pseudocode for this approach is shown in Algorithm 7. In the design of the dynamic step size, an attenuation factor is introduced, which allows the step size to adapt to changes in the environment. When the obstacle density increases, the step size automatically decreases to ensure better precision; when the environment is open, the step size increases to accelerate exploration and improve path planning efficiency.

This adaptive mechanism enables the algorithm to balance exploration efficiency and path precision in different environments, improving the overall quality of path planning. The formula for the dynamic step size is given in Equation (10):

$$\text{StepSize} = \text{StepSize}_{\max} \cdot e^{-\lambda \cdot N_{\text{obs}}} \tag{10}$$

where

- $\text{StepSize}_{\max}$: The maximum step size.
- λ: The attenuation factor.
- N_{obs}: The current number of nearby obstacles.

Algorithm 7: Dynamic Step Size

Input : $\Delta s_{\max}$, λ, $\mathbf{x}_{\text{current}}$, $\mathbf{x}_{\text{goal}}$, $\mathcal{X}_{\text{obs}}$
Output: $\mathbf{x}_{\text{new}}$

```
// Calculate the number of obstacles near the current position
```
$N_{\text{obs}} \leftarrow \text{CalculateNearbyObstacles}(\mathbf{x}_{\text{current}}, \mathcal{X}_{\text{obs}})$;
```
// Adjust the step size dynamically based on the number of nearby
    obstacles
```
$\Delta s \leftarrow \Delta s_{\max} \times \exp(-\lambda \times N_{\text{obs}})$;
```
// Move towards the new position
```
$\mathbf{x}_{\text{new}} \leftarrow \text{MoveTowards}(\mathbf{x}_{\text{current}}, \mathbf{x}_{\text{goal}}, \Delta s)$;
return $\mathbf{x}_{\text{new}}$;

3.4. Target Switching Strategy

In the traditional Bi-RRT* algorithm, the use of a fixed target point for guidance often leads to local optima in complex obstacle environments. This issue is particularly prominent in densely cluttered regions, where the tree expansion can be hindered, resulting in slower exploration and reduced path planning efficiency. To address this problem, a **target switching strategy** is introduced in this paper.

The core idea of this strategy is to randomly switch the target point with a certain probability P_{switch} during each iteration, by introducing a temporary random target point. The pseudocode for this approach is shown in Algorithm 8. By randomly switching the target point, the algorithm avoids repeatedly expanding near a single target point, thus increasing exploration diversity and preventing the algorithm from getting stuck in local optima or infinite loops. While the introduction of a temporary target point adds randomness to the search process, the algorithm still maintains a global guidance toward the final goal, ensuring that the search process ultimately converges towards the endpoint. In summary, the target switching strategy not only improves the efficiency of path planning but also enhances the algorithm's ability to escape local regions and find the global optimal path.

Algorithm 8: Dynamic Target Switching

Input : P_{switch}, $\mathbf{x}_{\text{goal}}$, $\mathcal{X}$ (Space)
Output: $\mathbf{x}_{\text{target}}$

if $rand() < P_{switch}$ **then**
    $\mathbf{x}_{\text{target}} \leftarrow \text{SelectRandomTarget}(\mathcal{X})$ `// Randomly select a new target point`
else
    $\mathbf{x}_{\text{target}} \leftarrow \mathbf{x}_{\text{goal}}$ `// Keep the original goal point`
return $\mathbf{x}_{\text{target}}$;

3.5. Different Region Sampling Probability Strategy

In the traditional Bi-RRT* algorithm, the sample points are distributed randomly, which often results in many samples being spread around the start point and the line connecting the start and end points, far from the ideal path. This issue is especially prevalent in complex environments, where the randomly distributed samples tend to concentrate in areas that do not require exploration. This leads to a large number of ineffective samples, increasing the computational overhead and reducing the efficiency of path planning. To address this issue, this paper introduces a region-based adaptive sampling probability strategy inspired by Thompson sampling. Specifically, the sampling probability is adjusted based on the Euclidean distance between the current sample and

the line connecting the start and end points. The sampling probability P_{sample} is higher for samples that are closer to this line, as shown in Equation (11):

$$\mathrm{P}_{\text{sample}} = e^{-\frac{d_{\text{line}}^2}{2\sigma^2}} \tag{11}$$

where:

- d_{line}: The distance between the sample point and the line connecting the start and goal points. The smaller d_{line}, the higher the probability P_{sample}.
- σ: The standard deviation controlling the spread of the sampling distribution.

With this strategy, the samples are concentrated near potential path regions, avoiding excessive exploration in irrelevant areas and thus significantly improving the efficiency of the path planning algorithm. The pseudocode for this approach is shown in Algorithm 9.

Algorithm 9: Region-Based Sampling Probability

Input : $\mathbf{x}_{\text{init}}$, $\mathbf{x}_{\text{goal}}$, $\mathcal{X}$ (Space), σ
Output: Sampled point $\mathbf{x}_{\text{sample}}$

function RegionBasedSampling($\mathbf{x}_{\text{init}}, \mathbf{x}_{\text{goal}}, \mathcal{X}, \sigma$);
while ***True*** **do**
    // Randomly sample a point in the search space
    $\mathbf{x}_{\text{rand}} \leftarrow \text{Sample}(\mathcal{X})$;
    // Calculate the distance from the sample point to the line connecting start and goal
    $d_{\text{line}} \leftarrow \text{DistanceToLine}(\mathbf{x}_{\text{rand}}, \mathbf{x}_{\text{init}}, \mathbf{x}_{\text{goal}})$;
    // Calculate the sampling probability P_{sample} based on Gaussian distribution
    $P_{\text{sample}} \leftarrow \exp\left(-\frac{d_{\text{line}}^2}{2\times\sigma^2}\right)$;
    // Select the current sample point with probability P_{sample}
    if $rand() < P_{sample}$ **then**
        return $\mathbf{x}_{\text{rand}}$;

function DistanceToLine($\mathbf{x}, \mathbf{x}_{\text{start}}, \mathbf{x}_{\text{end}}$);
// Calculate the shortest distance from a point to a line segment using projection
$\mathbf{v}_{\text{line}} \leftarrow \mathbf{x}_{\text{end}} - \mathbf{x}_{\text{start}}$;
$\mathbf{v}_{\text{point}} \leftarrow \mathbf{x} - \mathbf{x}_{\text{start}}$;
$\text{proj} \leftarrow \frac{\text{dot}(\mathbf{v}_{\text{point}}, \mathbf{v}_{\text{line}})}{\text{dot}(\mathbf{v}_{\text{line}}, \mathbf{v}_{\text{line}})}$;
$\text{proj} \leftarrow \max(0, \min(1, \text{proj}))$ // Clamp within [0, 1] range
$\mathbf{x}_{\text{closest}} \leftarrow \mathbf{x}_{\text{start}} + \text{proj} \times \mathbf{v}_{\text{line}}$;
return $\text{Distance}(\mathbf{x}, \mathbf{x}_{\text{closest}})$;

3.6. The Structure of the Improved Bi-APF-RRT Algorithm*

As shown in Figure 3, the improved Bi-APF-RRT* algorithm begins by initializing two trees and setting parameters before entering the main loop. In each iteration, a random temporary target point is introduced with a certain probability using the target switching strategy, and the sampling point is adjusted through dynamic goal biasing. Subsequently, regional probability sampling and the Artificial Potential Field (APF) are applied to optimize the sampling point, producing an adjusted sampling point for further expansion. During the expansion phase, a dynamic step size strategy is employed to adjust the expansion range based on environmental complexity, enabling bidirectional tree growth. If the expansion path is collision-free, the new node is added to the tree, and the algorithm

checks whether the two trees are successfully connected. If the connection is successful, the path is output; otherwise, the iterations continue until the termination condition is met or the optimal path is found.

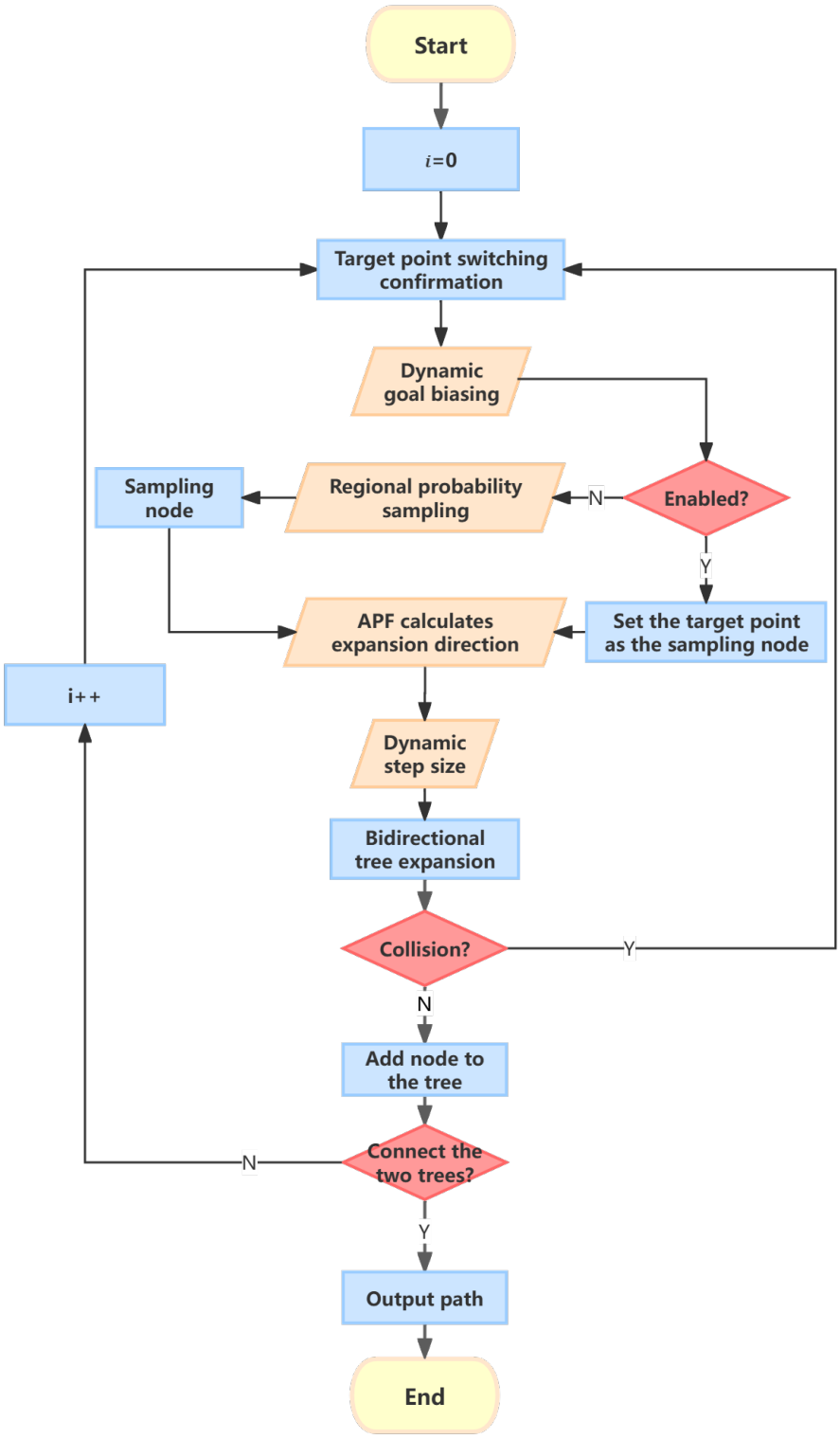

Figure 3. Flowchart of the Improved Bi-APF-RRT* Algorithm.

4. Experimental

4.1. Experimental Setup

To evaluate the performance of our improved algorithm in 3D path planning, all simulation experiments were conducted using MATLAB on a computer equipped with a single Nvidia GeForce RTX 4090 GPU (24GB VRAM), an Intel Core i9-14900K 24-core processor, and 64GB of RAM.

4.2. Simulation Comparison Experiments

4.2.1. Algorithm Comparison Experiments

When selecting algorithms for comparative experiments, RRT* and its variants were chosen due to their wide application and representativeness in the field of path planning. These algorithms are effective in addressing path planning problems in high-dimensional, dynamic, or partially known environments. Moreover, they improve computational efficiency and path quality through intelligent sampling and path optimization, making them suitable for meaningful comparisons with the proposed Bi-APF-RRT* algorithm. Additionally, while the RRT* series algorithms excel in global path optimization, they are prone to getting trapped in local minima in complex obstacle environments. Therefore, choosing these algorithms helps evaluate the advantages of Bi-APF-RRT* in avoiding local minima, enhancing path smoothness, and improving obstacle avoidance capabilities. Finally, considering that RRT* and its variants are primarily focused on 2D environments, this paper conducts experiments in 3D environments to further highlight the strengths of the Bi-APF-RRT* algorithm, particularly its adaptability and real-time performance in complex dynamic environments.

To eliminate the impact of random factors, we conducted six sets of experiments to compare the performance of the improved Bi-APF-RRT algorithm against the baseline algorithms RRT, Bi-RRT*, and Informed-RRT*. The performance metrics include: **Time** (runtime), **Path Length**, **Nodes** (number of generated nodes), **Path Angle** (average turning angle of the path), and **Success Rate** (the success rate of reaching the goal within the set maximum iterations). To ensure consistent experimental results, we standardized the following variable settings: the maximum iteration limit (MaxIter) was set to 1000, the start point was $(50, 50, 5)$, and the goal point was $(450, 450, 10)$. Additionally, the maximum step size and the average fixed step size of the baseline algorithms were set to 10, and the search radius was set to half of this value. Under these conditions, the experimental setup can reliably estimate the computational performance of the algorithms while minimizing the impact of abnormal variables.

As shown in Figures 4 and 5, the visualized results of one of the experiment groups clearly indicate that, compared with Informed-RRT* and Bi-RRT*, the improved Bi-APF-RRT algorithm generates fewer samples while maintaining a comparable search space, similar to the baseline algorithms. However, unlike RRT*, the improved Bi-APF-RRT algorithm's samples tend to concentrate more along the path direction towards the goal point. This directional bias reduces the number of nodes required for searching and improves the efficiency of path expansion. The results show that the improved Bi-APF-RRT and RRT* algorithms yield shorter path lengths compared to Informed-RRT* and Bi-RRT*. Additionally, it can be observed that, when navigating through cluttered environments, the improved Bi-APF-RRT algorithm's paths are smoother and avoid areas with dense obstacles. This is attributed to the dynamic response system introduced by the improved APF method, which allows the Bi-APF-RRT algorithm to effectively adapt to complex environments, ultimately producing safer and smoother paths.

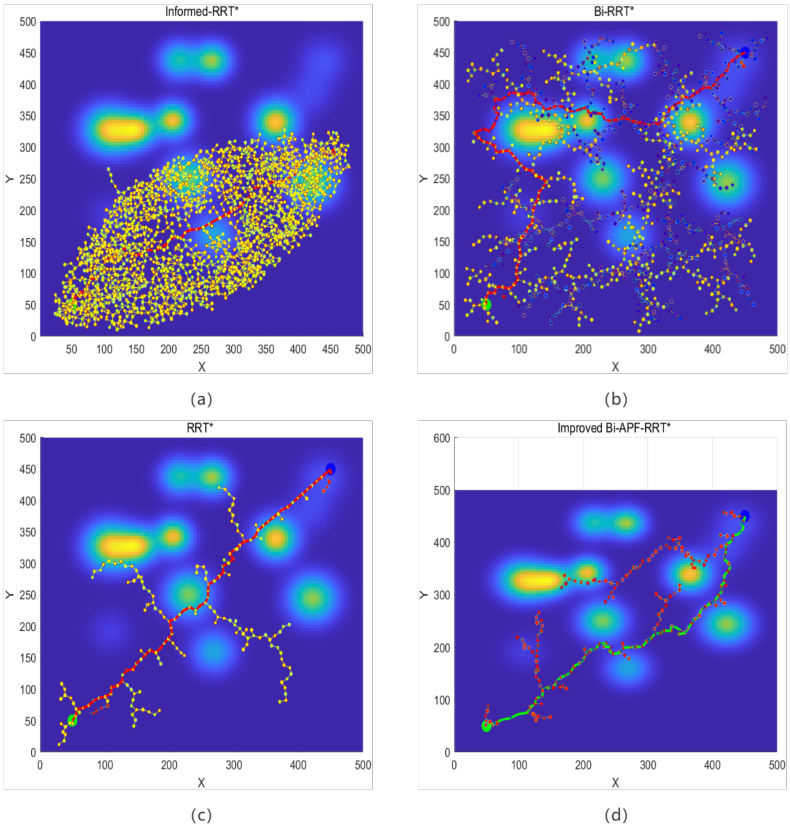

Figure 4. Comparison of Different RRT Variants for Path Planning. Figure (**a**) shows the visualization result of Informed-RRT*, Figure (**b**) shows the visualization result of Bi-RRT*, Figure (**c**) shows the visualization result of RRT*, and Figure (**d**) shows the visualization result of Improved Bi-APF-RRT*. In Figures (**a**–**c**), yellow nodes represent tree nodes and red represents the final path. In Figure (**d**), red nodes represent tree nodes and green represents the final path.

Figures 6 and 7 present detailed experimental data for each group, reflecting the average performance of various path planning algorithms during the experiments. From the data in these figures, it can be observed that the improved Bi-APF-RRT* algorithm clearly outperforms Informed-RRT*, Bi-RRT*, and RRT* in terms of path generation time, achieving an average planning time of 2.41 s, which is 92.3% faster than Informed-RRT* (31.19 s), 51.5% faster than Bi-RRT* (4.97 s), and 59.6% faster than RRT* (5.97 s), indicating a significant improvement in algorithm efficiency. At the same time, the average path length generated by Bi-APF-RRT* is 549.21, which is 26.3% shorter than Bi-RRT* (745.56) and 20.6% shorter than RRT* (691.56). While the path length is slightly shorter than Informed-RRT* (604.44), the latter suffers from a much higher computational cost and lower efficiency. Specifically, Informed-RRT* requires 1962 nodes on average to complete the search, compared to only 206.5 nodes for Bi-APF-RRT*, representing an 89.5% reduction in nodes, which further highlights the computational efficiency of the proposed algorithm. Moreover, the average path angle of Bi-APF-RRT* is 29.53 degrees, indicating smoother paths compared to Informed-RRT* (39.32 degrees), Bi-RRT* (35.19 degrees), and RRT* (33.28 degrees). Additionally, Bi-APF-RRT* achieves a 100% success rate, outperforming Informed-RRT* (92%), Bi-RRT* (96%), and RRT* (95%), demonstrating its robustness and reliability. Overall, while Informed-RRT* generates slightly shorter paths due to its larger sampling space, its much higher computational cost (31.19 s on average) and inefficiency make it unsuitable for real-time planning scenarios such as UAV applications, where fast and efficient path planning is critical.

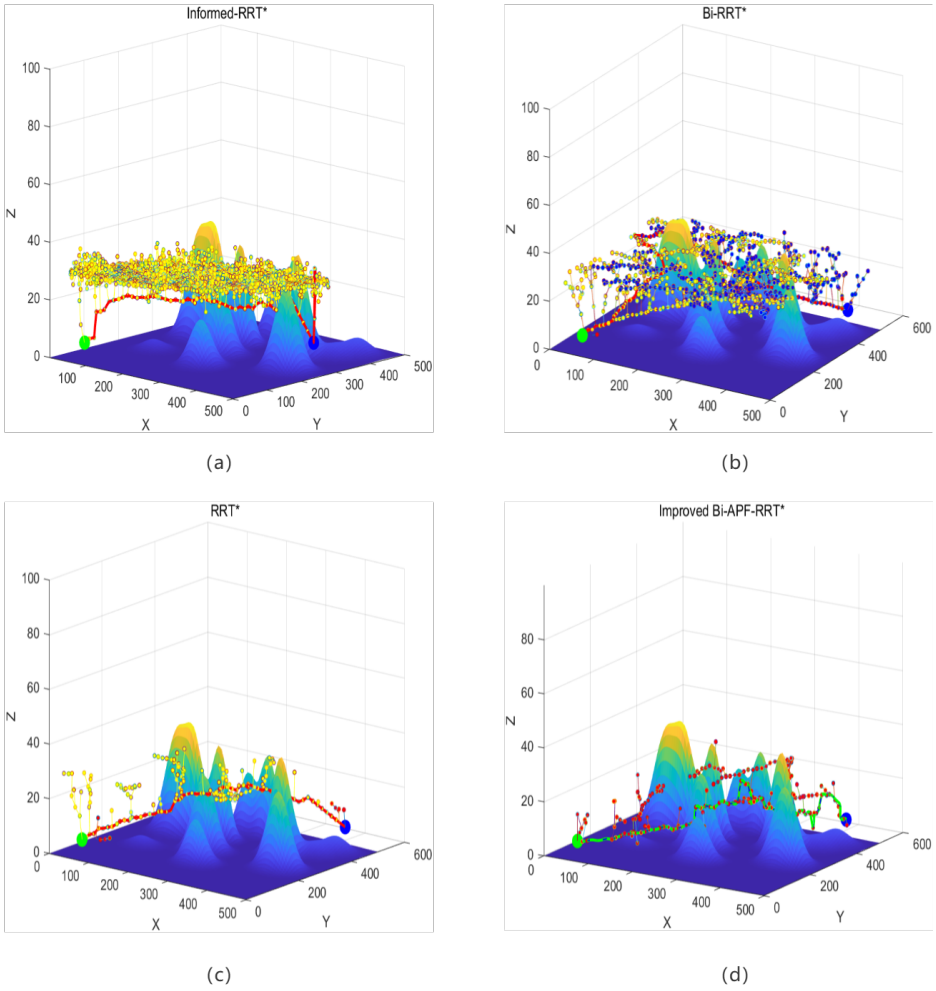

Figure 5. Three-Dimensional Terrain Path Planning Comparison of RRT Variants. Figure (**a**) shows the visualization result of Informed-RRT*, Figure (**b**) shows the visualization result of Bi-RRT*, Figure (**c**) shows the visualization result of RRT*, and Figure (**d**) shows the visualization result of Improved Bi-APF-RRT*. In Figures (**a**–**c**), yellow nodes represent tree nodes and red represents the final path. In Figure (**d**), red nodes represent tree nodes and green represents the final path.

Additionally, the improved Bi-APF-RRT* algorithm incorporates a new force parameter setting. In environments with dense obstacles, the algorithm actively avoids obstacles by maintaining a safe distance from them, rather than purely pursuing the shortest path. This approach enhances the safety and robustness of the planned paths. Meanwhile, with the dynamic adjustments of the step size, dynamic goal biasing, and region-based sampling probability strategy, the improved Bi-APF-RRT* algorithm shows a significant reduction in the number of sample nodes and the path angle metric compared to Informed-RRT*, Bi-RRT*, and RRT*, indicating further optimization in path smoothness and efficiency. In summary, the experimental results provide strong evidence that the improved Bi-APF-RRT* algorithm demonstrates significant advantages in search efficiency and path planning capability, especially in complex 3D environments, where it exhibits strong adaptability and robustness.

4.2.2. Parameter Comparison Experiments

To thoroughly evaluate the optimization performance of the improved Bi-APF-RRT* algorithm and identify the optimal parameter combinations, we conducted a single-factor experimental analysis of the key parameters. Table 1 presents the detailed experimental data. In the experiments, we observed that the force parameters and the dynamic step size

adjustment of the Bi-APF-RRT* algorithm had a significant impact on performance. Therefore, we focused on comparing different combinations of force parameters and dynamic step sizes.

Table 1. Comparison of Experimental Results with Different Step Size and Force Parameters.

$\mathbf{StepSize_{max}}$	α	β	Time (s)	Path Length	Nodes in Tree	Avg Path Angle
10	0.1	0.1	2.95	630.77	278	24.98
10	0.3	0.3	2.73	627.55	239	25.57
10	0.5	0.5	2.16	584.47	217	28.59
10	0.7	0.7	3.49	672.48	281	29.00
10	0.9	0.9	3.18	697.92	379	33.87
15	0.1	0.1	2.86	587.25	141	29.81
15	0.3	0.3	2.07	517.24	120	31.87
15	0.5	0.5	3.79	606.79	151	38.83
15	0.7	0.7	4.85	674.72	216	40.15
15	0.9	0.9	3.89	678.73	259	37.34
20	0.1	0.1	2.71	581.49	135	34.01
20	0.3	0.3	3.77	609.82	210	40.91
20	0.5	0.5	4.20	673.37	238	42.92
20	0.7	0.7	4.63	632.60	248	47.94
20	0.9	0.9	4.96	631.95	270	48.59

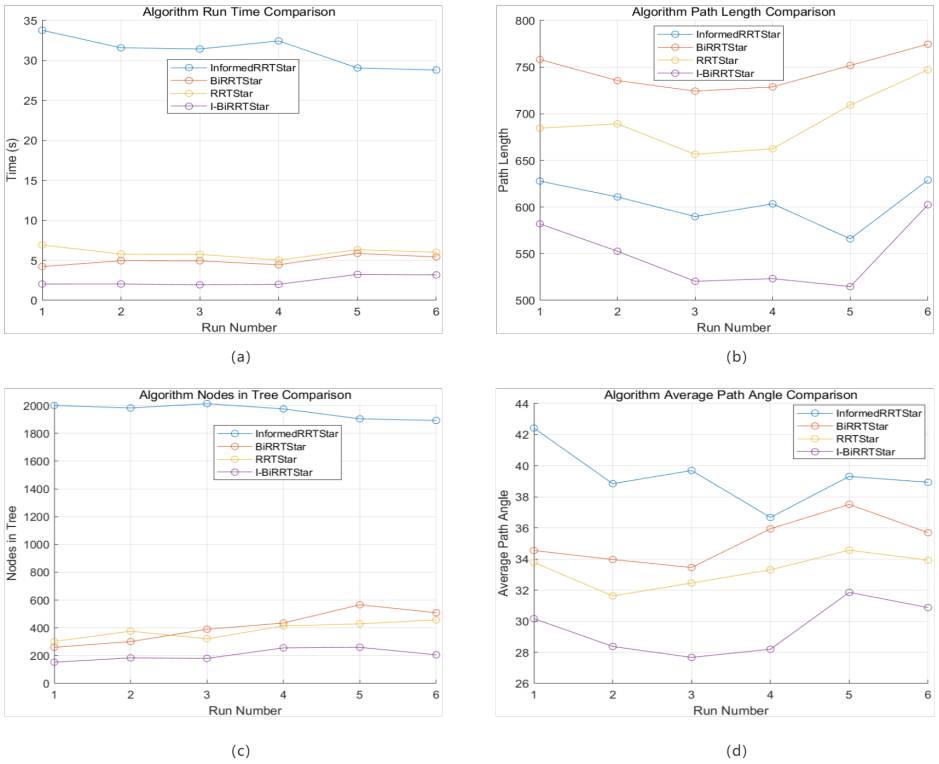

Figure 6. Evaluation of Informed-RRT, Bi-RRT, RRT*, and Improved Bi-APF-RRT* on Key Metrics. Figure (**a**) shows the comparison of execution times between our algorithm and the comparative algorithms. Figure (**b**) shows the comparison of generated path lengths between our algorithm and the comparative algorithms. Figure (**c**) shows the comparison of the total number of nodes generated during the execution process of our algorithm and the comparative algorithms. Figure (**d**) shows the comparison of the average path angle during the execution process of our algorithm and the comparative algorithms.

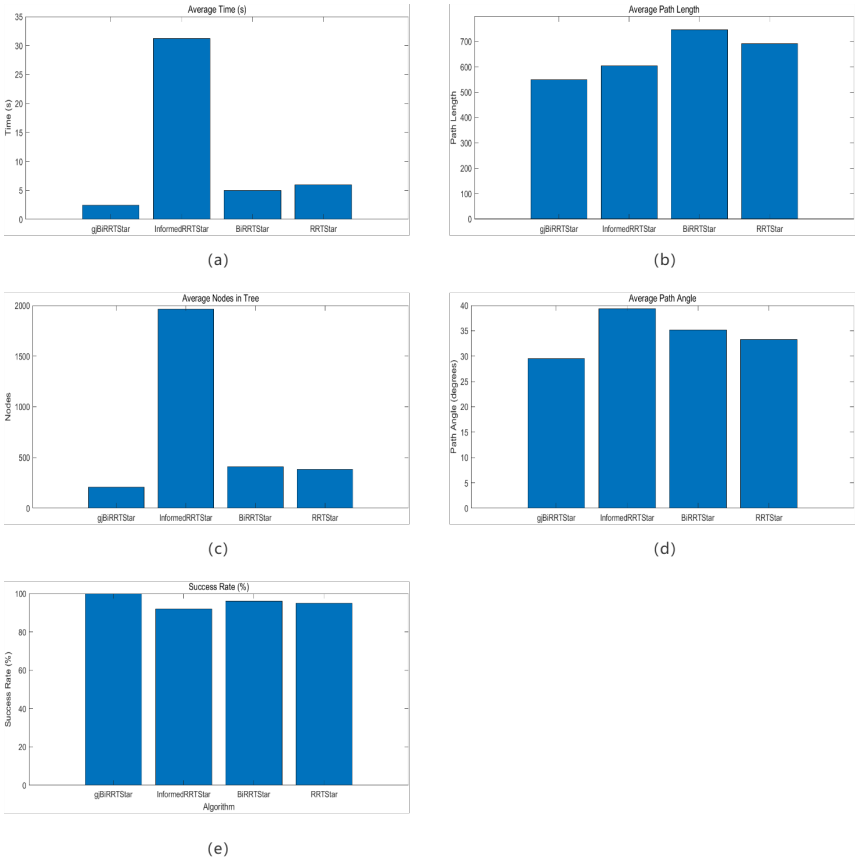

Figure 7. Average Performance Metrics of Informed-RRT, Bi-RRT, RRT*, and Improved Bi-APF-RRT*. Figure (**a**) shows the comparison of average execution times between our algorithm and the comparative algorithms. Figure (**b**) shows the comparison of average path lengths generated by our algorithm and the comparative algorithms. Figure (**c**) shows the comparison of the average number of nodes generated during the execution process of our algorithm and the comparative algorithms. Figure (**d**) shows the comparison of the total average path angles during the execution process of our algorithm and the comparative algorithms. Figure (**e**) shows the comparison of execution success rates between our algorithm and the comparative algorithms.

From the data in the table, it can be seen that in the 3D environment settings used in this paper, under the same conditions, an increase in the maximum step size $\text{StepSize}_{\max}$ leads to a noticeable decrease in the number of sample nodes, indicating improved search efficiency. However, as the step size increases, the average path turning angle (average path angle) also tends to increase, possibly due to a greater inclination towards straight-line expansion, resulting in sharper turns. Further analysis shows that when the maximum step size is set to 15 and the force parameters $\alpha = 0.3$, $\beta = 0.3$, the algorithm achieves a balance between fewer sample nodes, shorter path length, and a smaller average turning angle. When the step size is set to 15, and the force parameters $\alpha = 0.3$, $\beta = 0.3$, the algorithm exhibits the best performance, with a high success rate and reduced path oscillations. On the other hand, when the step size is increased to 20 and the force parameters are adjusted to $\alpha = 0.1$, $\beta = 0.1$, the algorithm still achieves good performance, avoiding excessive force that could lead to path oscillations.

In summary, the best parameter combination for the improved Bi-APF-RRT* algorithm, as verified in the 3D simulation experiments, is a maximum step size $\text{StepSize}_{\max}$ of 15, and force parameters $\alpha = 0.3$ and $\beta = 0.3$. This configuration enables the algorithm to dynamically adjust both the step size and force parameters across different environments,

producing smooth and optimized paths while reducing the number of sample nodes, enhancing search efficiency, and achieving reliable path planning performance.

5. Conclusions

In this paper, we propose an improved sampling-based path planning algorithm, Bi-APF-RRT*, which is based on the guided sampling approach of RRT*. The proposed algorithm integrates an improved Artificial Potential Field (APF) method with a newly introduced repulsive coefficient, while also utilizing a dynamic step size. Additionally, it incorporates three strategies: dynamic goal biasing, target switching, and region-based adaptive sampling probability.

In multiple sets of simulation experiments, the improved Bi-APF-RRT* algorithm demonstrated significant performance enhancements. Experimental results show that the proposed algorithm effectively controls the sampling space and direction during the initial path generation phase, making the search process more efficient and stable. It avoids getting stuck in local optima, significantly improving both the success rate and the quality of the path planning. Moreover, the results also confirm the algorithm's reliability in 3D path planning scenarios. Compared with the traditional RRT* and its variants, the improved Bi-APF-RRT* algorithm not only exhibits superior obstacle avoidance capabilities and faster convergence in complex environments, but also generates smoother and more reasonable paths, validating the advantages and practicality of this method in real-world applications.

Despite the outstanding performance of the improved Bi-APF-RRT* algorithm in the 3D path planning simulations for UAVs, there is still room for further improvements. Specifically, the generated paths are often discrete after the planning phase, which may lead to instability or increased energy consumption if directly applied to UAV flights. Therefore, future research will focus on how to further smooth the planned paths to better align with the motion characteristics and flight requirements of UAVs. This would not only enhance the flight efficiency but also improve the feasibility and safety of the paths, especially in complex and dynamic environments. Moreover, incorporating path smoothing techniques such as Bézier curves, spline interpolation, or optimal control methods could further enhance the application performance of the improved Bi-APF-RRT* algorithm.

Author Contributions: Conceptualization, Y.H.; methodology, Y.H.; software, Y.H.; investigation, Y.H.; writing—original draft preparation, Y.H.; data curation, H.L.; validation, H.L.; formal analysis, Y.D.; resources, Y.D.; writing—review and editing, G.L.; visualization, G.L.; supervision, M.D. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: The original data presented in the study are included in the article. Any further inquiries can be directed to the corresponding author.

Conflicts of Interest: Author Minglei Duan was employed by the company Yunnan Communications Investment & Construction Group Co., Ltd. The remaining authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

References

1. Yang, J.; Guo, Z.; Liu, J.; Liu, S. Research on APF-Bi-RRT Algorithm of Adaptive Step Strategy for Robot Path Planning. *Arab. J. Sci. Eng.* **2024**, 1–14. [CrossRef]
2. Xu, X.; Zhang, F.; Zhao, Y. Unmanned Aerial Vehicle Path-Planning Method Based on Improved P-RRT* Algorithm. *Electronics* **2023**, *12*, 4576. [CrossRef]
3. Fan, J.; Chen, X.; Liang, X. UAV trajectory planning based on bi-directional APF-RRT* algorithm with goal-biased. *Expert Syst. Appl.* **2023**, *213*, 119137. [CrossRef]
4. Zhao, W.; Tan, A.; Ren, C. An Innovative Path Planning Algorithm for Complex Obstacle Environments with Adaptive Obstacle Density Adjustment: AODA-PF-RRT*. *Electronics* **2024**, *13*, 4047. [CrossRef]
5. Tian, Z.; Li, L.; Wang, Y.; Wang, X.; Zan, P. An Environment Adaptation Algorithm Based on RRT for Manipulator Path Planning. In Proceedings of the 2023 42nd Chinese Control Conference (CCC), Tianjin, China, 24–26 July 2023 ; pp. 2821–2825. [CrossRef]

6. Chen, J.; Jiang, Y.; Pan, H.; Yang, M. Path Planning in Complex Environments Using Attention-Based Deep Deterministic Policy Gradient. *Electronics* **2024**, *13*, 3746. [CrossRef]
7. Dai, J.; Li, D.; Zhao, J.; Li, Y. Autonomous Navigation of Robots Based on the Improved Informed-RRT* Algorithm and DWA. *J. Robot.* **2022**, *2022*, 3477265. [CrossRef]
8. Liu, T. D.; Chen, X.; He, M.; Fu, X.P.; Wu, X.M.; Shao, G.F. Improved Artificial Potential Field based Parallel RRT Star for Fast Path Planning. In Proceedings of the 2021 China Automation Congress (CAC), Beijing, China, 22–24 October 2021; pp. 5801–5806. [CrossRef]
9. Jones, M.; Djahel, S.; Welsh, K. Path-Planning for Unmanned Aerial Vehicles with Environment Complexity Considerations: A Survey. *ACM Comput. Surv.* **2023**, *55*, 1–39. [CrossRef]
10. Zhou, Y.; Zhang, E.; Guo, Y.; Li, H. Lifting Path Planning of Mobile Cranes Based on an Improved RRT Algorithm. *Adv. Eng. Informatics* **2021**, *50*, 101376. [CrossRef]
11. Xu, J.; Song, K.; Dong, H.; Yan, Y. A Batch Informed Sampling-Based Algorithm for Fast Anytime Asymptotically-Optimal Motion Planning in Cluttered Environments. *Expert Syst. Appl.* **2020**, *144*, 113124. [CrossRef]
12. Yafei, L.; Anping, W.; Qingyang, C.; Yujie, W. An Improved UAV Path Planning Method Based on RRT-APF Hybrid Strategy. In Proceedings of the 2020 5th International Conference on Automation, Control and Robotics Engineering (CACRE), Dalian, China, 19–20 September 2020; pp. 81–86.
13. Liao, B.; Wan, F.; Hua, R.; Zhu, S.; Qing, X. F-RRT*: An Improved Path Planning Algorithm with Improved Initial Solution and Convergence Rate. *Expert Syst. Appl.* **2021**, *184*, 115457. [CrossRef]
14. Li, Y.; Wei, W.; Gao, Y.; Wang, D.; Fan, Z. PQ-RRT*: An Improved Path Planning Algorithm for Mobile Robots. *Expert Syst. Appl.* **2020**, *152*, 113425. [CrossRef]
15. Liu, Y.; Qi, J.; Wang, M.; Wu, C.; Sun, H. Path Planning for Large-Scale UAV Formation Based on Improved SA-APF Algorithm. In Proceedings of the 2022 41st Chinese Control Conference (CCC), Hefei, China, 25–27 July 2022; pp. 4472–4478.
16. Huang, T.; Huang, D.; Qin, N.; Li, Y. Path Planning and Control of a Quadrotor UAV Based on an Improved APF Using Parallel Search. *Int. J. Aerosp. Eng.* **2021**, *2021*, 5524841. [CrossRef]
17. Armstrong, D.; Jonasson, A. AM-RRT*: Informed Sampling-Based Planning with Assisting Metric. In Proceedings of the 2021 IEEE International Conference on Robotics and Automation (ICRA), Xi'an, China, 30 May–5 June 2021; pp. 10093–10099.
18. Wang, K.; Xu, J.; Song, K.; Yan, Y.; Peng, Y. Informed Anytime Bi-Directional Fast Marching Tree for Optimal Motion Planning in Complex Cluttered Environments. *Expert Syst. Appl.* **2023**, *215*, 119263. [CrossRef]
19. Wang, J.; Chi, W.; Shao, M.; Meng, M.Q.-H. Finding a High-Quality Initial Solution for the RRT Algorithms in 2D Environments. *Robotica* **2019**, *37*, 1677–1694. [CrossRef]
20. Li, Y.; Xu, Y.; Xue, X.; Liu, X.; Liu, X. Optimal Spraying Task Assignment Problem in Crop Protection with Multi-UAV Systems and Its Order Irrelevant Enumeration Solution. *Biosyst. Eng.* **2022**, *214*, 177–192. [CrossRef]
21. Zhou, Q.; Liu, G. UAV Path Planning Based on the Combination of A-Star Algorithm and RRT-Star Algorithm. In Proceedings of the 2022 IEEE International Conference on Unmanned Systems (ICUS), Guangzhou, China, 28–30 October 2022; pp. 146–151.
22. Pu, H.; Wan, X.; Song, T.; Schonfeld, P.; Peng, L. A 3D-RRT-star algorithm for optimizing constrained mountain railway alignments. *Eng. Appl. Artif. Intell.* **2024**, *130*, 107770. [CrossRef]
23. Yin, X.; Dong, W.; Wang, X.; Yu, Y.; Yao, D. Route planning of mobile robot based on improved RRT star and TEB algorithm. *Sci. Rep.* **2024**, *14*, 8942. [CrossRef] [PubMed]
24. Mohammed, H.; Romdhane, L.; Jaradat, M.A. RRT* N: An Efficient Approach to Path Planning in 3D for Static and Dynamic Environments. *Adv. Robot.* **2021**, *35*, 168–180. [CrossRef]
25. Chen, Z.; Zhang, X.; Wang, L.; Xia, Y. A Fast Path Planning Method Based on RRT Star Algorithm. In Proceedings of the 2023 3rd International Conference on Consumer Electronics and Computer Engineering (ICCECE), Guangzhou, China, 6–8 January 2023; pp. 258–262.
26. Wang, W.; Gao, H.; Yi, Q.; Zheng, K.; Gu, T. An Improved RRT* Path Planning Algorithm for Service Robot. In Proceedings of the 2020 IEEE 4th Information Technology, Networking, Electronic and Automation Control Conference (ITNEC), Chongqing, China, 12–14 June 2020; Volume 1, pp. 1824–1828.
27. Wang, W.; Gao, H.; Yi, Q.; Zheng, K.; Gu, T. UAV Path Planning Using Optimization Approaches: A Survey. *Arch. Comput. Methods Eng.* **2022**, *29*, 4233–4284.
28. Ahmad, A.; Qadeer, K.; Naquash, A.; Riaz, F.; Hasan, M.; Qyyum, M.A.; Lee, M. Particle Swarm-Assisted Artificial Neural Networks for Making Liquefied Natural Gas Processes Feasible Under Varying Feed Conditions. *Front. Energy Res.* **2022**, *10*, 917656. [CrossRef]
29. Liu, P.; Zhang, B. An Autonomous Quadrotor Exploration Combining Frontier and Sampling for Environments with Narrow Entrances. In Proceedings of the 2022 41st Chinese Control Conference (CCC), Hefei, China, 25–27 July 2022; pp. 3656–3661.
30. Fan, J.; Chen, X.; Wang, Y.; Chen, X. UAV Trajectory Planning in Cluttered Environments Based on PF-RRT* Algorithm with Goal-Biased Strategy. *Eng. Appl. Artif. Intell.* **2022**, *114*, 105182. [CrossRef]
31. Alzahrani, B.; Oubbati, O.S.; Barnawi, A.; Atiquzzaman, M.; Alghazzawi, D. UAV Assistance Paradigm: State-of-the-Art in Applications and Challenges. *J. Netw. Comput. Appl.* **2020**, *149*, 102706. [CrossRef]
32. Aggarwal, S.; Kumar, N. Path Planning Techniques for Unmanned Aerial Vehicles: A Review, Solutions, and Challenges. *Comput. Commun.* **2020**, *149*, 270–299. [CrossRef]

33. Puente-Castro, A.; Rivero, D.; Pazos, A.; Fern, ez-Blanco, E. A Review of Artificial Intelligence Applied to Path Planning in UAV Swarms. *Neural Comput. Appl.* **2022**, *34*, 153–170. [CrossRef]
34. Zhang, Z.W.; Jia, Y.W.; Su, Q.Q.; Chen, X.T.; Fu, B.P. ATS-RRT*: An improved RRT* algorithm based on alternative paths and triangular area sampling. *Adv. Robot.* **2023**, *37*, 605–620. [CrossRef]
35. Liang, Y.; Zhao, H. CCPF-RRT*: An improved path planning algorithm with consideration of congestion. *Expert Syst. Appl.* **2023**, *228*, 120403. [CrossRef]
36. Wang, D.; Zheng, S.; Ren, Y.; Du, D. Path Planning Based on the Improved RRT* Algorithm for the Mining Truck. *Comput. Mater. Contin.* **2022**, *71*. [CrossRef]
37. Xue, W.; Wang, B.; Huang, X.; Yang, B.; Wen, Z.; Zhang, H.; Li, S. Spacecraft attitude maneuver planning with multi–sensor pointing constraints using improved RRT–star algorithm. *Adv. Space Res.* **2023**, *72*, 1485–1495. [CrossRef]
38. Wang, H.; Li, G.; Hou, J.; Chen, L.; Hu, N. A path planning method for underground intelligent vehicles based on an improved RRT* algorithm. *Electronics* **2022**, *11*, 294. [CrossRef]
39. Ding, J.; Zhou, Y.; Huang, X.; Song, K.; Lu, S.; Wang, L. An improved RRT* algorithm for robot path planning based on path expansion heuristic sampling. *J. Comput. Sci.* **2023**, *67*, 101937. [CrossRef]
40. Wu, D.; Wei, L.; Wang, G.; Tian, L.; Dai, G. APF-IRRT*: An improved informed rapidly-exploring random trees-star algorithm by introducing artificial potential field method for mobile robot path planning. *Appl. Sci.* **2022**, *12*, 10905. [CrossRef]
41. Guo, J.; Xia, W.; Hu, X.; Ma, H. Feedback RRT* algorithm for UAV path planning in a hostile environment. *Comput. Ind. Eng.* **2022**, *174*, 108771. [CrossRef]
42. Suwoyo, H.; Adriansyah, A.; Andika, J.; Ubaidillah, A.; Zakaria, M.F. An Integrated RRT* SMART-A* Algorithm for solving the Global Path Planning Problem in a Static Environment. *IIUM Eng. J.* **2023**, *24*, 269–284. [CrossRef]
43. Zhang, J.; Li, X.; Liu, X.; Li, N.; Yang, K.; Zhu, H. Navigation Method Based on Improved Rapid Exploration Random Tree Star-Smart (RRT*-Smart) and Deep Reinforcement Learning. *J. Donghua Univ. (Engl. Ed.)* **2022**, *39* . [CrossRef]
44. Cao, Y.; Cheng, X.; Mu, J. Concentrated Coverage Path Planning Algorithm of UAV Formation for Aerial Photography. *IEEE Sens. J.* **2022**, *22*, 11098–11111. [CrossRef]
45. Zhang, W.; Li, J.; Yu, W.; Ding, P.; Wang, J.; Zhang, X. Algorithm for UAV Path Planning in High Obstacle Density Environments: RFA-Star. *Front. Plant Sci.* **2024**, *15*, 1391628. [CrossRef] [PubMed]
46. Wang, L.; Zhang, X.; Zheng, H.; Wang, C.; Gao, Q.; Zhang, T.; Li, Z.; Shao, J. A Butterfly Algorithm That Combines Chaos Mapping and Fused Particle Swarm Optimization for UAV Path Planning. *Drones* **2024**, *8*, 576. [CrossRef]
47. Wu, T.; Zhang, Z.; Jing, F.; Gao, M. A Dynamic Path Planning Method for UAVs Based on Improved Informed-RRT* Fused Dynamic Windows. *Drones* **2024**, *8*, 539. [CrossRef]
48. Abhishek, B.; Ranjit, S.; Shankar, T.; Eappen, G.; Sivasankar, P.; Rajesh, A. Hybrid PSO-HSA and PSO-GA Algorithm for 3D Path Planning in Autonomous UAVs. *Soc. Netw. Appl. Sci.* **2020**, *2*, 1805. [CrossRef]
49. Liu, J.; Yan, Y.; Yang, Y.; Li, J. An Improved Artificial Potential Field UAV Path Planning Algorithm Guided by RRT Under Environment-Aware Modeling: Theory and Simulation. *IEEE Access* **2024**, *12*, 12080–12097. [CrossRef]
50. Huang, C.; Tang, B.; Guo, Z.; Su, Q.; Gai, J. Agile-RRT*: A Faster and More Robust Path Planner with Enhanced Initial Solution and Convergence Rate in Complex Environments. *IEEE Access* **2024**, *12*, 58703–58714. [CrossRef]
51. Yang, F.; Fang, X.; Gao, F.; Zhou, X.; Li, H.; Jin, H.; Song, Y. Obstacle avoidance path planning for UAV based on improved RRT algorithm. *Discret. Dyn. Nat. Soc.* **2022**, *2022*, 4544499. [CrossRef]