

Article

HHPSO: A Heuristic Hybrid Particle Swarm Optimization Path Planner for Quadcopters

Jiabin Lou, Rong Ding * and Wenjun Wu

State Key Laboratory of Software Development Environment, School of Artificial Intelligence, Beihang University (BUAA), Beijing 100191, China; loujiabin@buaa.edu.cn (J.L.); wwj09315@buaa.edu.cn (W.W.)

* Correspondence: dingr@buaa.edu.cn

Abstract: Path planning for quadcopters has been proven to be one kind of NP-hard problem with huge search space and tiny feasible solution range. Metaheuristic algorithms are widely used in such types of problems for their flexibility and effectiveness. Nevertheless, most of them cannot meet the needs in terms of efficiency and suffer from the limitations of premature convergence and local minima. This paper proposes a novel algorithm named Heuristic Hybrid Particle Swarm Optimization (HHPSO) to address the path planning problem. On the heuristic side, we use the control points of cubic b-splines as variables instead of waypoints and establish some heuristic rules during algorithm initialization to generate higher-quality particles. On the hybrid side, we introduce an iteration-varying penalty term to shrink the search range gradually, a Cauchy mutation operator to improve the exploration ability, and an injection operator to prevent population homogenization. Numerical simulations, physical model-based simulations, and a real-world experiment demonstrate the proposed algorithm's superiority, effectiveness and robustness.

Keywords: particle swarm optimization; path planning; motion capture; unmanned aerial vehicles (UAVs); aerial systems; applications



Citation: Lou, J.; Ding, R.; Wu, W. HHPSO: A Heuristic Hybrid Particle Swarm Optimization Path Planner for Quadcopters. *Drones* **2024**, *8*, 221. <https://doi.org/10.3390/drones8060221>

Academic Editors: Jihong Zhu, Heng Shi, Zheng Chen and Minchi Kuang

Received: 19 April 2024

Revised: 22 May 2024

Accepted: 22 May 2024

Published: 28 May 2024



Copyright: © 2024 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

1. Introduction

Path planning is the cornerstone of unmanned aerial systems, enabling Unmanned Aerial Vehicles (UAVs) to handle complex scenarios [1]. For any given scenario, there are generally three elements: task, individual, and environment, which together comprise the path planning optimization problem. The planned path should be optimal within specific criteria associated with the task. For example, in time-sensitive tasks such as air delivery and military transport, the core principle is to minimize the distance between the drone access locations, thereby reducing the time and fuel cost [2]. However, for search and rescue, patrol, and other search-related tasks, the performance metric is usually to maximize the area coverage within a fixed amount of time [3]. Meanwhile, the planned path should be able to safely address environmental threats and smoothly respond to individual maneuver properties. In general, solving this optimization problem is not a complicated task. However, as a task becomes more urgent, the environment becomes more complex, or the UAV becomes less manoeuvrable, it remains a challenge to calculate feasible optimal paths to avoid all threats within an acceptable time [4].

Researchers have proposed a series of approaches to solve the UAV path planning problem. Previously, the path planning problem was generally equivalent to the shortest path problem, and deterministic search algorithms were widely adopted, for instance, the Dijkstra algorithm [5], the Voronoi diagram method [6] and the A* algorithm [7,8]. However, as researchers began to consider the specific demands of different scenarios, the optimal path has become associated with the average altitude, fuel consumption, environmental threats and so on, making it clear that the problem has become increasingly complex. We know that the path planning problem is an NP-hard problem, which is

difficult to solve with deterministic algorithms when the scale of the problem becomes large. Therefore, researchers have slowly shifted from using deterministic to using non-deterministic algorithms.

Metaheuristics are one kind of non-deterministic algorithm that can provide a sufficiently good solution to an optimization problem with limited computation capacity. They are nature-inspired, population-based, and generation-iterated, facilitating powerful global searching and rapid convergence capability. In recent years, metaheuristic algorithms have been increasingly favored for UAV path planning because of their ability to explore low-dimensional manifolds in high-dimensional space [9]. For example, ref. [10] proposed an initial population enhancement method in a Genetic Algorithm (GA), which speeds up the convergence process. Ref. [11] proposed a spherical vector-based Particle Swarm Optimization (PSO) to solve the problem within complicated environments subjected to multiple threats. In addition, other meta-heuristic algorithms such as Differential Evolution (DE) [12–14], Ant Colony Optimization (ACO) [15,16], and Wolf Pack Search (WPS) [17] have been extensively studied in recent years.

Among these algorithms, PSO is much simpler to implement while maintaining excellent efficiency, effectiveness and scalability, and thus has been successfully applied in many drone fields. Ref. [18] presented a comprehensively improved Particle Swarm Optimization (CIPSO) algorithm with the chaos-based logistic map initialization and mutation strategy to solve this problem in war scenarios. Ref. [19] proposed a multiobjective particle swarm optimization algorithm with multimode collaboration based on reinforcement learning (MCMOPSO-RL) algorithm to find the optimal path and handle threats simultaneously. Ref. [20] proposed the SHOPSO algorithm, which combines the Selfish Swarm Optimizer (SHO) and the PSO, to accomplish a given combat mission at a meager cost. Nevertheless, the scenarios used in these studies are relatively simple: the number of threats was small, and the terrain was small-scale or flat, making these algorithms impractical for complex real scenes. The main reason is that PSO, as a general optimizer, does not analyze the built-in physics for specific problems. Although it inherently provides approaches for automatically abstracting features from iterations, the iterations require sufficient time to play a part. Therefore, with the limited execution time, PSO always suffers from premature convergence limitations, hindering its promotion in complex, high-dimensional, and noisy bounded scenarios.

Compared to other scenarios, the drone scenarios often require path planning algorithms that can respond quickly and ensure safety [21]. On the one hand, the drone tasks are often urgent, resulting in limited planning time for the algorithm. On the other hand, a drone scenario has many threats and complex terrain, which makes the algorithm easily to fall into local optimum. For the former problem, we established heuristic rules during particle initialization to prevent invalid searching and inspire the powerful search efficiency of the algorithm. For the latter problem, we hybridized a Cauchy mutation operator, an injection operator and a penalty function to enhance the exploration capabilities of the algorithm. The new algorithm, called Heuristic Hybrid Particle Swarm Optimization (HHP SO), strikes a good balance between exploration and exploitation and significantly improves the convergence, robustness and constraint-handling ability. Numerical simulations, Unreal Engine 4 (UE4) [22] simulations and a real-drone experiment confirmed the results.

The remainder of this paper is organized as follows: Section 2 proposes the problem scenarios and optimization model. The heuristic rules and hybrid operators are introduced in Section 3. The experimental results are provided in Section 4. Section 5 concludes this article.

2. Problem Statement

The quadcopters path planning problem can be treated as a multi-objective constrained optimization problem. In this section, scenario representation and the optimization model are discussed.

2.1. Scenario Representation

In a drone scenario, two elements must be considered for the path planning task. One is the terrain, which imposes physical constraints on drones. The other is threats, which constitute dangers that drones may encounter in their missions.

2.1.1. Terrain

The terrain of the drone scenario is totally open, implying that we can discretize the broad planning space into a surface. On this basis, the terrain can be generated using several perlin noises layers [23] with diverse frequencies and amplitudes. This terrain ensures that the height z_{ij} is unique and continuous over the entire plane (x_i, y_j) . To save computing resources, we sample the entire plane at specific intervals to obtain a point cloud map of the terrain, as shown in Figure 1a. However, such a discretized representation of terrain cannot constrain all points on the plane, so we used triangle interpolation among every three nearest points, thus confirming the unique mapping of (x_i, y_j) , denoted as $z_{ij} = \text{Map}(x_i, y_j)$, as shown in Figure 1b.

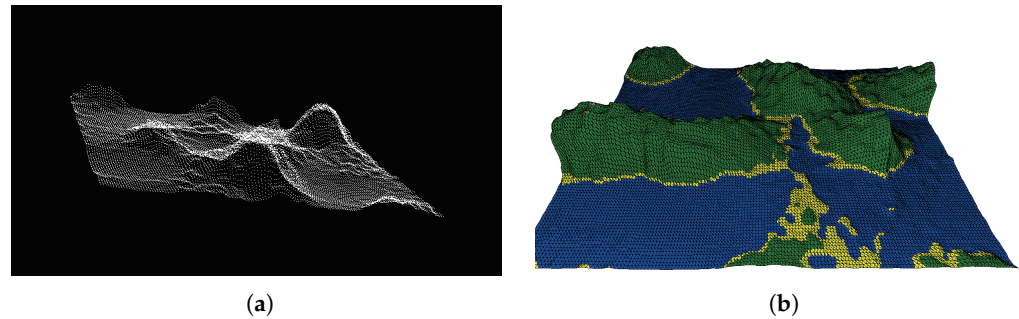


Figure 1. Terrain representation. (a) Terrain representation with a point cloud. (b) Terrain representation with triangle interpolation.

2.1.2. Threats

In general, quadcopters should remain concealed and secure when performing tasks in a drone scenario. We assume that the enemy will use radar and missiles to detect and attack drones. In addition, there are some No-fly Zones (NFZs) in the scene that drones cannot approach.

The probability of radar and missile affecting flight safety can be calculated by (1) and (2) [24].

$$P_R = \begin{cases} \frac{1}{1 + \zeta_2 (d^4 / RCS)^{\zeta_1}} & \text{if } d \leq R_R \\ 0 & \text{otherwise} \end{cases} \quad (1)$$

$$P_M = \begin{cases} R_M^4 / (R_M^4 + d^4), & \text{if } d \leq R_M \\ 0, & \text{otherwise.} \end{cases} \quad (2)$$

where R_R and R_M denote the maximum influence distance of the radar and missile, respectively, d is the distance between the drone's position and the missile and radar deployment center, and ζ_1 and ζ_2 depend on the radar used. RCS denotes the radar cross-section, which can be calculated according to the drone's position and velocity [25].

For the NFZs, we need to determine whether the waypoints are within their range, so we defined the following equation:

$$\text{InNFZs}(w_i) = \begin{cases} 1, & \text{if } w_i \text{ falls in any NFZs} \\ 0, & \text{otherwise} \end{cases} \quad (3)$$

where $w_i (i = 1, 2, \dots, n)$ are waypoints.

2.2. Optimization Model

Mathematically, the path planning problem can be modeled as a Multi-Objective Constraint Satisfaction Problem, which comprises three components, i.e., variables, objectives, and constraints.

2.2.1. Variables

Typically, a path planning problem uses waypoints as planning variables. Suppose we start from start point $S : (x_S, y_S, z_S)^T$, and go to target point $T : (x_T, y_T, z_T)^T$. The path planning problem can be depicted as finding a series of waypoints $W = \{w_1, w_2, \dots, w_{n-1}\}$ through which the UAV can reach its destination successfully. However, the outputs obtained in the three-dimensional configuration space cannot guarantee differential flat control. For example, these paths may contain sharp turns that challenge the kinematics and dynamics of the drone.

Several methods have been proposed to generate a smoothing path from the control points $P = \{p_0, p_1, \dots, p_m\}$. In some previous research, the Dubins curve was used to smooth the path [26]. A Dubins curve uses a series of arcs and straight line segments to form the motion path of the drone, as shown in the Figure 2a. This method is unsuitable for parameterization, because it may generate many arcs without curvature continuity. Another method used in recent research is the Rauch-Tung-Striebel (RTS) smoother, which consists of two stages, Kalman forward filtering and RTS backward smoothing [27], as depicted in Figure 2b. At a higher computational cost, RTS smoother achieves relatively low tracking errors between generated paths and control points. However, it is unnecessary in the proposed algorithm because the UAV does not require flying directly through the control points. A Bezier curve [28] and b-spline curve [29] are two of the most well-known path smoothing methods, as shown in Figure 2c,d. Of these two curves, the latter evolves from the former and inherits all the advantages, including geometrical invariance, convexity-preserving, and affine invariance. Compared to the Bezier curve, the b-spline curve overcomes the disadvantage that moving one control point affects the entire curve and does not increase the degree of the polynomial no matter how many control points are added [30].

The b-spline function used in this paper can be defined as:

$$w_i = \sum_{j=0}^m p_j B_{j,k} \left(\frac{i}{n+1} \right) \quad (4)$$

where $w_i (i = 1, 2, \dots, n)$ are waypoints, $p_j (j = 0, 1, \dots, m)$ denote control points, $B_{j,k}()$ are the k -order normalized b-spline basic functions defined by the de Boor–Cox recursion formula as follows:

$$\begin{cases} B_{j,0}(t) = \begin{cases} 1, & \text{if } u_j \leq t \leq u_{j+1} \\ 0, & \text{otherwise} \end{cases} \\ B_{j,k}(t) = \frac{t-u_j}{u_{j+k}-u_j} B_{j,k-1}(t) + \frac{u_{j+k+1}-t}{u_{j+k+1}-u_{j+1}} B_{j+1,k-1}(t) \\ \text{define } 0/0 = 0 \end{cases} \quad (5)$$

where $t \in [0, 1]$ and $U = \{u_0, u_1, \dots, u_{k+m}\}$ is a non-decreasing sequence of parameters called the knot vector.

B-spline enjoys C^{k-1} continuous property and the derivative of a b-spline is still a b-spline curve with order $k - 1$. Therefore, we choose the cubic b-spline curve with C^2 continuity (this guarantees that the quadcopters will not be commanded to change their propeller speed sharply) to convert the control points P into waypoints W . Concretely, we fixed the p_0 at the starting point S , the p_m at the target point T and defined all remaining intervening control points as decision variables:

$$\xi = \{p_1, p_2, \dots, p_{m-1}\} \quad (6)$$

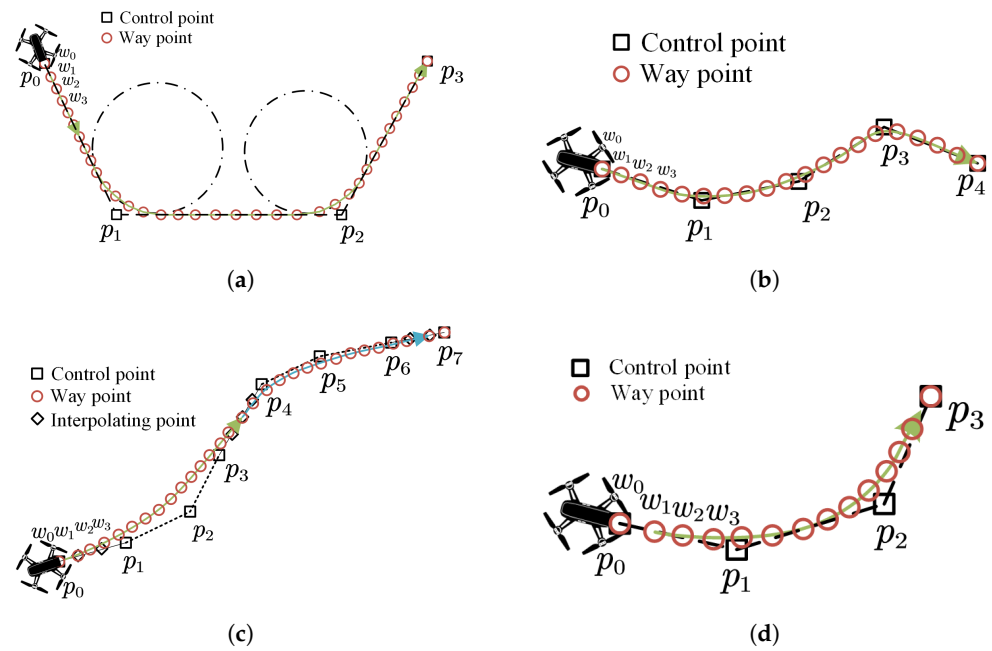


Figure 2. Smooth methods. (a) Tangent circle curve. (b) RTS Smoother. (c) Six-order Bezier Curve. (d) Cubic b-spline Curve

2.2.2. Objectives

In order to obtain a short, safe and smooth trajectory, five objectives, i.e., length cost, flight altitude, radar detection, missile attack and turning angle are considered. Generally, these objectives are somewhat contradictory so we treat them into a weighted function (7).

$$F = \omega_1 f_1 + \omega_2 f_2 + \omega_3 f_3 + \omega_4 f_4 + \omega_5 f_5 \tag{7}$$

where $\omega_1, \omega_2, \omega_3, \omega_4, \omega_5$ are weights that sum to 1, and f_1, f_2, f_3, f_4, f_5 denote different objectives.

(1) Length Cost

Traditionally, the goal of a planner is to find the shortest path conforming to constraints, and the normalized approximate length of a path is defined as (8).

$$f_1 = \frac{\sum_{i=1}^n \sqrt{\|w_i - w_{i-1}\|}}{\sqrt{\|w_n - w_0\|}} \tag{8}$$

where $w_i = [x_i, y_i, z_i]$ denotes the coordinate of the i^{th} waypoint.

(2) Flight Altitude

A lower altitude is desired for the sake of using ground effect to avoid radars and saving fuel. The mean flight altitude of a path is denoted by (9).

$$f_2 = \sum_{i=1}^{n-1} FA_i \quad \text{with} \tag{9}$$

$$FA_i = \begin{cases} 0, & \text{if } z_i \leq Map(x_i, y_i) \\ (z_i - Map(x_i, y_i)) / n, & \text{otherwise} \end{cases}$$

where $Map(x_i, y_i)$ represent the terrain height at (x_i, y_i) .

(3) Radar Detection

The probability of the quadcopters being detected by radars can be calculated as follows:

$$f_3 = \sum_{i=1}^{n-1} \sum_{j=1}^R P_{Rij} \quad (10)$$

where R denotes the number of radars in the scenario, P_R can be obtained according to (1).

(4) Missile Attack

The probability of the quadcopters being attacked by missiles is expressed as follows:

$$f_4 = \sum_{i=1}^{n-1} \sum_{j=1}^M P_{Mij} \quad (11)$$

where M denotes the number of missiles in the scenario, P_M is calculated according to (2).

(5) Smoothness

This is designed to evaluate the smoothness of the planned path. Values closer to 0 indicate smoother paths, and 0 means a straight path [31].

$$f_5 = \frac{1}{n-1} \sum_{i=1}^{n-1} \theta_i \quad \text{with} \quad (12)$$

$$\theta_i = \arccos \left(\frac{(x_i - x_{i-1}, y_i - y_{i-1}) \cdot (x_{i+1} - x_i, y_{i+1} - y_i)^T}{\|(x_i - x_{i-1}, y_i - y_{i-1})\| \cdot \|(x_{i+1} - x_i, y_{i+1} - y_i)\|} \right)$$

To ensure the single-valuedness and continuity of the arccos function, that is, to ensure each value corresponds to a unique θ_i , we choose to restrict the function's range to $[0, \pi]$.

2.2.3. Constraints

It's well accepted that quadcopters should meet the following constraints to ensure safe and stable flight.

(1) Climbing/Gliding Angle

Since the maneuverability of quadcopters, the slope s_i should be restricted in the range of maximum climbing angle α_i and minimum gliding angle β_i [20]. This forms the constraint functions g_1 and g_2 :

$$g_1 = \max(s_i - \alpha_i) \leq 0 \quad (13)$$

$$g_2 = \max(\beta_i - s_i) \leq 0 \quad (14)$$

For i in $1, \dots, n-1$, where

$$\alpha_i = -1.5377 \times 10^{-10} z_i^2 - 2.6997 \times 10^{-5} z_i + 0.4211 \quad (15)$$

$$\beta_i = 2.5063 \times 10^{-9} z_i^2 - 6.3014 \times 10^{-6} z_i - 0.3257 \quad (16)$$

$$s_i = \frac{z_{i+1} - z_i}{\sqrt{(x_{i+1} - x_i)^2 + (y_{i+1} - y_i)^2}} \quad (17)$$

(2) Turning angle constraint

The turning angle θ_i at waypoint w_i can be calculated according to (12). Due to the maneuverability constraints of the quadcopters, the turning angle should not be greater than its upper bound, which can be written as

$$g_3 = \max(\theta_i - \theta_i^{max}) \leq 0 \quad (18)$$

(3) Minimum flight altitude

For safety reasons, quadcopters should be at a certain level with the ground, as described in (19).

$$g_3 = H_{\text{safe}} - \min(z_i - \text{Map}(x_i, y_i)) \leq 0 \quad (19)$$

where H_{safe} denotes the minimal safe flight height.

(4) Forbidden flying area

According to mission requirements, the quadcopters have to keep away from NFZs. We describe this as a hard constraint as follows:

$$h_1 = \sum_{i=1}^{n-1} \text{InNFZs}(w_i) = 0 \quad (20)$$

3. Approach

3.1. Standard Particle Swarm Optimization

The standard PSO algorithm was developed by Kennedy and Eberhart in 1995 based on social and cognitive behavior [32], and is widely used in engineering. It solves problems by generating candidate solutions (particles) and moving those particles through a search space based on their positions and velocities, as seen in (21) and (22).

$$V_k = wV_k + c_1 \cdot r_1 \cdot (pb_k - \zeta_k) + c_2 \cdot r_2 \cdot (gb - \zeta_k) \quad (21)$$

$$\zeta_k = \zeta_k + V_k \quad (22)$$

where the Equation (21) updates a new velocity for the k -th particle according to its previous velocity V_k , its current position ζ_k , its best historical position pb_k and the current global best position gb . And w is the inertia weight that determines the particle to maintain its original trend, r_1 and r_2 denote two random numbers, c_1 and c_2 are learning factors. The Equation (22) updates each particle's position based on its updated velocity from the former.

To further improve the algorithm's efficiency for solving path planning problems, we introduce the heuristic rules to guide the search and hybrid some operators to speed up the convergence.

3.2. Heuristic Rules

When a drone move from the start position to its goal, there are strong constraints inside a path for respecting the kinematic and the dynamic limits. Embedding these limits into initialization can provide informative priors, i.e., strong physics constraints and inductive biases to guide the search.

Thanks to the convexity-preserving property, constraining the derivatives of the control points is sufficient for constraining the entire b-spline [33]. Therefore we set up heuristic rules for control points.

3.2.1. Rotated Coordinate System

Searching points within a 3-D Cartesian coordinate system has been widely used in current studies. But it is usually inefficient since the heading direction is rarely consistent with axes, so the inherent sequential relationships between waypoints or control points are underutilized. In this paper, we use the rotated coordinate system $O_R-X_R Y_R Z$ to initialize the candidate solutions, as shown in Figure 3, where X_R is the direction from S to T . The transformation between the two coordinates can be obtained by the rotating matrix:

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} x^R \\ y^R \\ z^R \end{pmatrix} \begin{pmatrix} \cos \theta & \sin \theta & 0 \\ -\sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{pmatrix} + \begin{pmatrix} x_S \\ y_S \\ 0 \end{pmatrix} \quad (23)$$

where θ is the angle from the X axis to the X_R axis, x_S and y_S denote the x coordinate and y coordinate of the start point S , and the superscript R represents the coordinates in the rotating coordinate system.

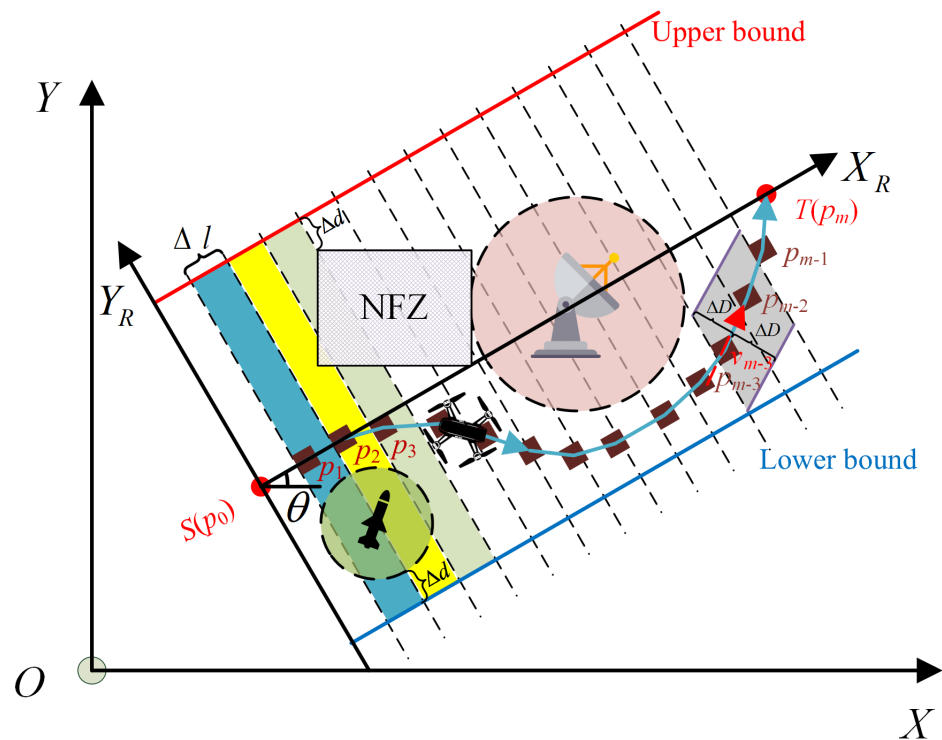


Figure 3. Division of UAV mission space.

In $O_R-X_R Y_R Z$, we assume that the control points are monotonically increasing along the X_R axis, which means the quadcopters cannot move backwards. Actually, some other scholars [12,34,35] have also utilized this benefit in the phase of path initialization and they have proved that the solution set is almost never lost.

3.2.2. Physical Plausibility

Physically plausible paths should ensure the quadcopters keep motor speed changing smoothly, implying that the distance moved in each time interval should not differ much. According to this rule, we divided the \overline{ST} by $(n + 1) \Delta l$ segments and defined each equilateral point as the expected position of the control points. Considering that the distribution of each point is different and correlated, we limited a control point's x^R value between the expected position of its previous and next point in the initialization phase.

Similarly, the y^R coordinate can be constrained. As shown in Figure 3, we define two boundaries parallel to the velocity that constrain the velocity direction variation trend with the metric ΔD . The value of ΔD is determined based on the maximum flight speed of the drone. In this paper, We set ΔD equal to Δl , so that the components of the path along the x and y axes are of the same order of magnitude at initialization. Therefore, we can roughly determine the x^R and y^R range of p_j according the position of p_{j-1} (e.g., p_{m-2} is locked in the gray area). This restrict is modeled as a Markov Chains, which require a considerable computational cost to be handled, and also in order not to lose the possible solution, it's only used in the initialization phase.

In addition, we defined a mission space. On the X_R -axis, the x^R coordinate is confined between 0 and $|\overline{ST}|$. And on the Y_R -axis, two lines in Figure 3 determine the upper and lower bound, which are obtained by extending outward a constant distance Δd of the

points from the nearest safe areas around \overline{ST} . Under the boundary, the y^R coordinate is restricted in $[y_{min}, y_{max}]$, which are calculated as (24) and (25), respectively:

$$y_{min} = \min \left\{ \min_i \{y_{threat,i}^* - R_i\}, 0 \right\} - \Delta d \tag{24}$$

$$y_{max} = \max \left\{ \max_i \{y_{threat,i}^* + R_i\}, 0 \right\} + \Delta d \tag{25}$$

where R_i is the radius or the circumradius of the i^{th} threat, $y_{threat,i}^*$ is the vertical coordinate of i^{th} threat in the rotated coordinate frame O_R .

3.2.3. Initialization

In practice, we introduce three criteria to the initialization process.

- x^R coordinate

In the rotated coordinate system we established before, the x^R value is monotonically increasing. In addition, we assume x_j^R obeying the normal distribution with mean ($j\Delta l$) and standard deviation ($\Delta l/3$) so that its value will fall in the range we expected with 99.7% probability according to the pauta criterion [36].

- y^R coordinate

For y coordinate, it's highly dependent on the state of the previous point. So we initialize the y_i^R in the range of $(-\Delta D + y'_j, \Delta D + y'_j)$ under uniform distribution, where y'_j reflects the state of the previous point and is calculated by (26).

$$y'_j = \begin{cases} 0 & , j < 1 \\ \frac{y_{j-1}^R - y_{j-2}^R}{x_{j-1}^R - x_{j-2}^R} (x_j^R - x_{j-1}^R) + y_{j-1}^R & , \text{otherwise} \end{cases} \tag{26}$$

- z coordinate

Heuristically, the drone's path follows the ups and downs of the terrain. So we initialize the z -value obeying the normal distribution with mean z'_i and standard deviation Δh , where Δh is set roughly equal to $\Delta l/3$ in this paper according to the maneuverability of quadcopters.

$$z'_j = z_{j-1} + Map(x_j, y_j) - Map(x_{j-1}, y_{j-1}) \tag{27}$$

3.3. Hybrid Operators

3.3.1. Penalty Function

We use the penalty function as the constraint-handling method to evaluate the particles better. This approach defines the particle's fitness function as the sum of the objective function and the penalty term due to constraint violation:

$$fit(\xi_k) = F(\xi_k) + \varphi(\xi_k, r^{(\alpha)}) \tag{28}$$

where $\varphi(\xi_k, r^{(\alpha)})$ indicates the penalty term of the k^{th} particle in α^{th} generation and it can be calculated as follows:

$$\varphi(\xi_k, r^{(\alpha)}) = r^{(\alpha)} \left(\sum_{i=1}^4 (\text{Max}[0, g_i(\xi_k)])^2 + h_1(\xi_k) \right) \tag{29}$$

where $r^{(\alpha)}$ is the penalty factor, it changes with iteration to ensure that infeasible individuals suffer from more selection pressure at the later stage of iteration:

$$0 < r^{(1)} < r^{(2)} < \dots < r^{(\alpha-1)} < \lim_{\alpha \rightarrow \infty} r^{(\alpha)} = +\infty \tag{30}$$

In this paper, the penalty coefficient is designed as a quadratic function of α .

$$r_\alpha = 10 \times \alpha^2 / I_{\max}^2 \quad (31)$$

where I_{\max} denotes the maximum iteration algebra.

3.3.2. Cauchy Mutation

Like other variants of PSO, the heuristic PSO faces the problem of premature convergence; that is, its particles converge to a local optimum in some scenarios. Intuitively, a stochastic mechanism might help the premature particles escape from the current trapped local optimum, thus avoiding premature convergence. Genetic algorithms have a similar mechanism called **mutation** to help the individual escape from local optima [37]. Suppose we apply the mutation operator to the premature particles, and then use the fitness function to evaluate the results. A good mutation means that the mutated particle has a better fitness value than the original one. Following this criterion, we design a mutation operator for PSO.

Since the Cauchy distribution has a small peak value at the origin but a long distribution at both ends, it can generate a larger disturbance near the individual to jump out of the local optimum, thus we choose Cauchy distribution to generate trail variable. The operator process is shown below:

Firstly, we sorted all particles according to their fitness and took out the inferior half of the particles as the target of the mutation operator.

Secondly, for each selected particle, the trail variable is generated from its current position in the following way:

$$\zeta'_k = \zeta_k + C(0, \gamma) \quad (32)$$

where ζ_k denote the position of the k^{th} particle and $C(0, \gamma)$ represents a Cauchy random vector of the same dimension as ζ_k with a location parameter 0 and a scale parameter γ .

Finally, replace the origin particle with the trail variable if the mutated particle has better fitness value.

3.3.3. Injection

Besides, there is an approach commonly used in the meta-heuristic algorithm to increase the randomness of the population, namely **injection** [38]. Due to the role of heuristic rules, the initial particles of our algorithm are of high quality. Therefore, if the inferior particle can be initialized with a higher fitness value; it is beneficial for the population to escape from the current trapped local optimum. Similar to the rules of mutation, the injection operator operates on the sorted particles at the following scale:

$$\lambda^{(\alpha+1)} = \zeta \lambda^{(\alpha)} \quad (33)$$

where $\lambda^{(\alpha)}$ indicates the number of injected particles in α -th generation, ζ is the decay factor.

Since the substantial irregularity of the injected particles, the new particle swarm is sorted to ensure the effectiveness of the injection, and the λ_α particles at the bottom are eliminated.

3.4. Algorithm Presentation

To summarize, we use the standard PSO as the prototype, initialize particles with heuristic rules, hybrid Penalty function operator, Cauchy mutation operator, and Injection operator to improve the search capability and propose the Heuristic Hybrid Particle Swarm Algorithm (HHP SO). The pseudo-code of HHP SO is given in Algorithm 1. It's worth noting that the fitness $fit(\xi)$ in (28) converts the control points $[p_0, \xi, p_m]$ to waypoints using a cubic b-spline curve.

Algorithm 1: HHPSO

```

1 Initialize:  $N$  particles with velocity 0 following the heuristic rules in Section 3.2;
2 Assign the best historical position  $pb$  of each particle to its position  $\xi$ ;
3 for  $\alpha = 1$  to  $I_{\max}$  do
4    $gb \leftarrow \operatorname{argmin}[fit(\xi_k, r^\alpha)]$ , for  $k = 1$  to  $N$ ;
5   for  $k = 1$  to  $N$  do ▷ standard PSO
6     | Update  $\xi_k$  and  $V_k$  using Equations (21) and (22);
7   end
8   for  $k = 1$  to  $N/2$  do ▷ Mutation
9     | Generate trail vector  $\xi'_k$  according to Equation (32);
10    | Replace  $\xi_k$  with  $\xi'_k$  and reset its velocity to 0 if  $fit(\xi'_k) < fit(\xi)$ ;
11  end
12  Generate  $\lambda_\alpha$  particles and inject them into the swarm; ▷ Injection
13  Delete  $\lambda_\alpha$  inferior particles according to their fitness;
14  Calculate  $\lambda_{\alpha+1}$  and  $r_{\alpha+1}$  according to the Equations (31) and (33);
15 end
16 return  $gb$ 

```

4. Experimental Results

To verify the effectiveness of the proposed algorithm, numerical simulations, physical model-based simulations and a real-drone experiment are designed. In the numerical simulation, we deploy the HHPSO to handle various scenarios with increasing obstacles and different terrains and set up several respective algorithms to draw comparisons. However, sim-to-real translation has been known to be a long-standing problem in robotics. But it was difficult for us to set up an accurate test site with the terrain and various constrained areas. As a compromise, we built similar scenarios on UE4 as physical model-based simulations. And for the real-drone experiment, we choose NOKOV motion capture system as the global location system and use Bitcraze Crazyflie 2.1 nano-quadrotors [39] as the flying platform for its characteristics of small volume (9cm rotor-to-rotor), lightweight (34 g), and suitable for indoor flying. All experiments were conducted on a desktop computer featuring an Intel Core i9-9900K CPU, 32 GB of DDR4 RAM, a 1 TB NVMe SSD, and an NVIDIA GeForce RTX 3070Ti GPU.

4.1. Numerical Simulation

In this paper, the proposed algorithm was run in four scenarios with different terrains and increasing obstacles, and some other recently proposed metaheuristic planners, i.e., GA [10], CIPSO [18], CIPDE [13], JADE [12], mWPS [17], were selected as the compared algorithms, the hyperparameters of these algorithms are shown in the Supplementary Materials (Tables S1–S6). Besides, the heuristic-PSO and the hybrid-PSO are also put as comparative algorithms to further discuss the influence of the two operators proposed in this paper.

For comparisons, all algorithms are shared with the same basic parameters: the population size of 30, the maximum iteration algebra of 25 and the waypoints number of 35. In addition, the main control parameters of the mentioned algorithms are shown in Table 1, the definitions of these parameters can be found from their original papers. The heuristic-PSO and the hybrid-PSO share the same control parameters as HHPSO. Since the high security requirements of the scenarios, we have set the weights ($\omega_1 = 0.2$, $\omega_2 = 0.1$, $\omega_3 = 0.3$, $\omega_4 = 0.3$, $\omega_5 = 0.1$) among the five objective functions.

Table 1. Main Control Parameters of Used Algorithms.

Algorithm	GA			CIPSO			JADE			
parameter	α	β	fr	w	c	μ	a	u_{CR}	u_F	
value	0.5	0.5	10	[0.4, 0.9]	[0.5, 3.5]	4	2	0.5	0.5	
Algorithm	CIPDE			mWPS		HHPSO				
parameter	μ_F	μ_{CR}	c	Eliminated-Qty	Safari-Wolves-Qty	w	c_1	c_2	γ	ζ
value	0.7	0.5	0.1	5	5	1	1.5	1.5	2	0.9

In order to test the performance of algorithms under different threat density and terrain, we designed four scenarios. Scenarios 1–3 are on the same flat terrain with increasing threats, and scenario 4 is extremely challenging with rugged terrain and crowded threats. The threats are randomly generated with the numbers 1, 4, 10, and 20 for each type, the radar and missile influence area is a sphere of a radius of 10, the RCS is set to -23.8 [40], and the no-fly zone randomly covers an area of 10–30.

All algorithms are implemented in scenarios 1–4, and the numerical experiment results are shown in Figure 4. To better analyze the fitness of the path that satisfies the constraint, which is critical in the drone scenario, we truncate the portion with a fitness value greater than 5.

From the 2D view of planning results, all the algorithms can complete the task in simple scenarios (Scenarios 1 and 2). But in the complex scenarios (scenarios 3 and 4), our algorithm can always find a relatively good location to avoid the threat and reach the destination, while the rest paths frequently enter the radar areas, missile areas and NFZs. More clearly seen from the fitness figure, as the scenario gets more complicated, the performance gaps between the proposed algorithm and others become more and more apparent. As the penalty coefficient r^α increases, all algorithm except HHPSO can't handle constraints well. For example, in scenario 3, there are two algorithms that complete the task, i.e., heuristic-PSO and HHPSO, while the path generated by heuristic-PSO is clearly stuck in a local optimum. But in scenario 4, only HHPSO is left. Further, we look at constraint functions. In all scenarios, HHPSO starts with a low constraint value and ends up satisfying the constraints well. This is mainly because the method fully considers the problem-dependent heuristics during initialization and the introduction of penalty term results in higher selection pressure along with iteration. And the heuristic-PSO, which also benefits from the heuristic initialization, has the same property of low starting constraint value. Moreover, it shows that the mixture of Cauchy mutation and injection operators makes our method less prone to falling into local optima and prematurity. Without these operators, the algorithms easily fall into local minimum, like the heuristic-PSO. But without the heuristical initialization, the algorithm doesn't even converge; the hybrid-PSO provides a good example. And other algorithms perform fine in simple scenarios but miserably in complex ones. Therefore, it can be inferred that the heuristic rules and the hybrid operators are of great help to planners, especially in complex scenarios.

To compare these algorithms more rigorously, we ran each algorithm 100 times in each scenario and adopted several metrics to measure the algorithm performance, i.e., Successful Rate (SR), Average Fitness (AF), Average Constraints value (AC) and Average Time (AT). Here we define the path that satisfies most conditions (at least 3), and the constraint value does not exceed 0.1 as a successful plan. The results are recorded in Table 2. Among all algorithms, HHPSO achieves the best fitness with a success rate of over 90% in all scenarios. The worst performer is CIPSO, probably because its original paper environment has no dense threats and thus weaker constraint control. Some other planners like GA, JADE, CIPDE and mWPS may get a tolerable AF in simple scenarios but perform poorly in scenarios 3 and 4. In addition, the two variants of HHPSO, i.e., heuristic-PSO and hybrid-PSO, performed reasonably well. Heuristic-PSO shows strong constraint handling ability and requires less running time than HHPSO. But its average fitness value is weaker

than HHPSO, meaning it often falls into the local minimum. Hybrid-PSO also requires less running time than HHPSO, but due to its weaker constraint handling capability, it does not perform as well as the other two.

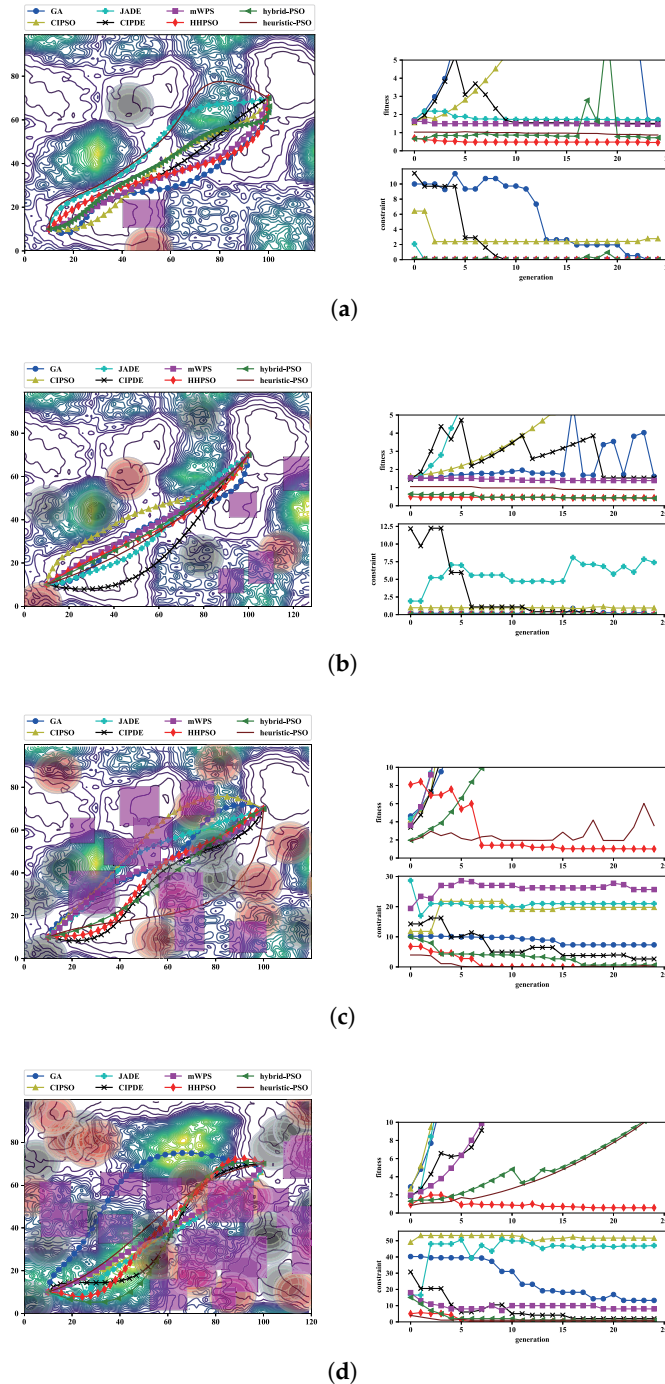


Figure 4. The comparative results among different algorithms. (a) Scenario 1. (b) Scenario 2. (c) Scenario 3. (d) Scenario 4.

From the above discussions, it can be seen that the proposed planner is more effective and efficient than the compared algorithms. In terms of effectiveness, HHPSO achieves the highest SR and the best AF in all scenarios, which is crucial in drone scenarios. And in terms of efficiency, HHPSO runs in less than 0.4 s and can be implemented for urgent tasks.

Table 2. Statistical Results for Different Algorithms.

		GA	CIPSO	JADE	CIPDE	mWPS	HHPSO	Heuristic-PSO	Hybrid-PSO
Scenario 1	SR(%)	87.00 ± 2.61	82.00 ± 2.46	96.00 ± 3.12	92.00 ± 2.76	95.00 ± 2.85	100.00 ± 0.00	99.00 ± 2.97	97.00 ± 2.91
	AF	1.73 ± 0.03	18.20 ± 0.91	1.75 ± 0.04	1.28 ± 0.03	1.45 ± 0.04	0.65 ± 0.03	0.85 ± 0.02	0.97 ± 0.02
	AC	0.07 ± 0.00	1.78 ± 0.04	0.09 ± 0.00	0.08 ± 0.00	0.05 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.07 ± 0.00
	AT(s)	0.39 ± 0.01	0.54 ± 0.01	0.62 ± 0.02	0.60 ± 0.02	0.89 ± 0.02	0.23 ± 0.01	0.19 ± 0.02	0.21 ± 0.01
Scenario 2	SR(%)	86.00 ± 2.58	81.00 ± 2.43	75.00 ± 2.25	84.00 ± 2.52	89.00 ± 2.67	98.00 ± 0.13	96.00 ± 2.88	89.00 ± 2.67
	AF	1.76 ± 0.04	14.85 ± 0.74	18.26 ± 0.91	1.83 ± 0.04	1.55 ± 0.04	0.49 ± 0.01	0.51 ± 0.01	0.85 ± 0.02
	AC	0.13 ± 0.00	1.41 ± 0.03	1.75 ± 0.04	0.12 ± 0.00	0.09 ± 0.00	0.00 ± 0.00	0.01 ± 0.00	0.05 ± 0.00
	AT(s)	0.33 ± 0.01	0.45 ± 0.01	0.45 ± 0.01	0.47 ± 0.01	0.77 ± 0.02	0.22 ± 0.00	0.15 ± 0.05	0.19 ± 0.00
Scenario 3	SR(%)	26.00 ± 0.78	12.00 ± 0.36	20.00 ± 0.60	48.00 ± 1.44	27.00 ± 0.81	94.00 ± 1.35	68.00 ± 2.04	25.00 ± 0.75
	AF	79.27 ± 2.38	208.20 ± 4.16	228.90 ± 5.73	45.50 ± 1.37	295.50 ± 8.87	1.16 ± 0.02	3.75 ± 0.11	10.85 ± 0.33
	AC	7.73 ± 0.2	20.54 ± 0.62	22.75 ± 0.68	4.48 ± 0.13	29.45 ± 0.88	0.02 ± 0.00	0.32 ± 0.01	0.97 ± 0.03
	AT(s)	0.41 ± 0.01	0.58 ± 0.02	0.58 ± 0.02	0.61 ± 0.02	0.92 ± 0.03	0.34 ± 0.00	0.27 ± 0.05	0.32 ± 0.00
Scenario 4	SR(%)	15.00 ± 0.45	6.00 ± 0.18	11.00 ± 0.33	25.00 ± 0.75	2.00 ± 0.06	92.00 ± 2.32	52.00 ± 1.56	15.00 ± 0.45
	AF	15.41 ± 0.46	516.30 ± 15.49	472.80 ± 14.18	22.27 ± 0.67	81.80 ± 2.45	1.35 ± 0.31	10.50 ± 0.32	13.50 ± 0.41
	AC	13.20 ± 0.40	51.50 ± 1.55	47.20 ± 1.42	2.18 ± 0.07	8.05 ± 0.24	0.05 ± 0.01	0.92 ± 0.03	1.27 ± 0.04
	AT(s)	0.44 ± 0.01	0.68 ± 0.02	0.68 ± 0.02	0.73 ± 0.02	1.23 ± 0.04	0.38 ± 0.00	0.27 ± 0.03	0.29 ± 0.00

4.2. Physical-Based Simulation

We built a suitable terrain in the realistic physics engine Unreal Engine 4 (UE4) and deployed AirSim [41] as the dynamics model of the quadcopters. The terrain is shown in Figure 5a and the drone model used in the AirSim is the Ar Drone, which is shown in Figure 5b.

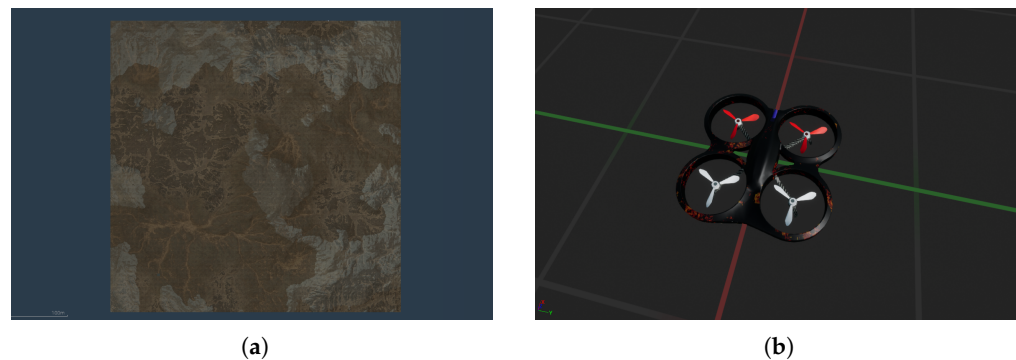


Figure 5. Basic components of the physical simulation. (a) The terrain in UE4. (b) Basic components of the physical simulation.

The terrain is augmented with different threats to generate two experimental scenarios; scenario 1 has [10, 10, 10] radars, missiles and NFZs and scenario 2 with [30, 30, 30]. The radar and missile detection radius is 20 for scenario 1 and 10 for scenario 2. Those threats, together with the interpolation point clouds for terrain scanning and the target point are used as the inputs of HHPSO to generate waypoints. The results are shown in Figure 6, the demonstration videos are available at <https://youtu.be/9-ZV-A0M3b4> and <https://youtu.be/VLxXbAZJzQQ>, (accessed on 7 January 2024). It turns out that drones can follow planned paths to complete missions, and our algorithms can address diverse scenarios, such as larger threat zones and numerous and fragmented threats.

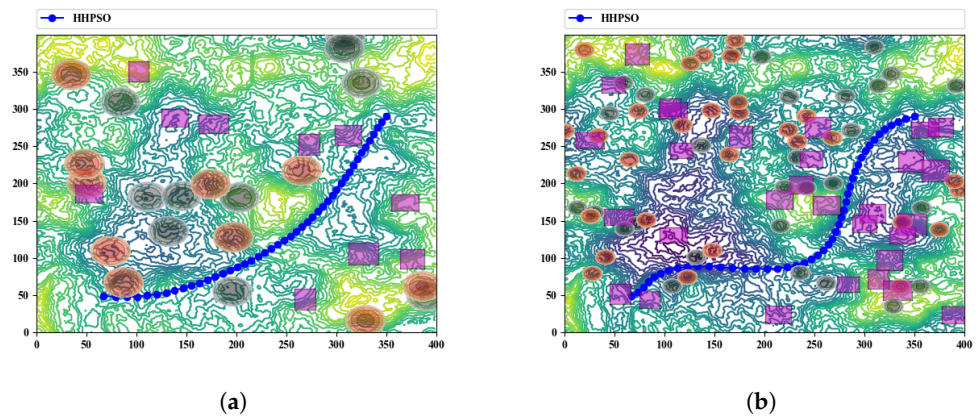


Figure 6. The planned paths of HHPSO in UE4. (a) Sparse environment. (b) Dense environment.

4.3. Real-Drone Experiment

Although UE4 simulates physical conditions, it is still just a simulation. We have carried out an in-door experiment to verify the validity of the proposed algorithm. Concretely, we use a 3×1 m test site indoors to map to the simulation system at a ratio of 1 to 10. For such a small venue, we chose Bitcraze CrazyFlie 2.1 [39], the world’s smallest quadcopter, as the executor of the algorithm. As shown in Figure 7, we used some boxes to simulate the NFZs and set the safe height as 5 m (0.5 m in the real world).



Figure 7. Real-world experimental environment

To verify the real-time performance of the algorithm, we set up a series of checkpoints for the quadcopters to pass through (point A to D in Figure 8). The mission of the planner is to plan and execute a path to the next checkpoint when the quadcopters approaches a checkpoint. Although the quadcopters can move in any direction, we force the drone to keep its heading consistent with the direction of moving during the experiment. Besides, we add the last three points of the previous plan to the next plan, and calculate the turning angle according to (12). In this way, the quadcopters can do continuous planning without hovering over an intermediate point.

The path recorded from a motion capture system is shown in Figure 8, and the video is available at <https://youtu.be/Fis1Fm25z04>, (accessed on 7 January 2024). We can see that the first path of the quadcopters from point A to B is not consistent with the subsequent planning, this is because that they arrive at point A from different directions, resulting in different initial states. Moreover, due to the limited space of the physical room, we introduced a 10x zoom, resulting in some sharp turns in the flight of the quadcopters. But

the actual planning result (Figure 8) is very smooth. This experiment demonstrates that our algorithm is efficient enough to be deployed for real-time path planning of quadcopters.

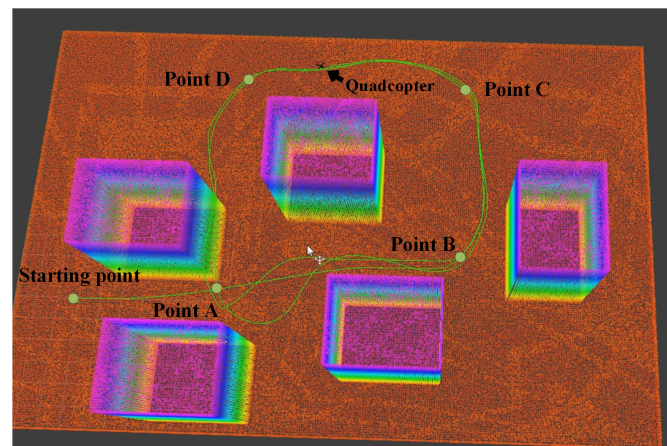


Figure 8. Trajectory of the Crazyflie

5. Conclusions and Discussions

Path planning plays a crucial role in autonomous unmanned systems. This paper presents an effective and efficient PSO-based path planning algorithm that allows the quadcopters to complete navigation tasks in complex scenarios. Concretely, we set up a series of heuristic rules during population initialization to generate high-quality particles to avoid invalid searches. But the heuristic-PSO is easy to fall into the local optimum, so we hybrid the penalty function, Cauchy mutation operator and Injection operator further to improve the global search ability of the algorithm. The proposed algorithm is named HHPSO; comparative numerical simulations of four scenarios with increasing obstacles show that HHPSO outperforms other state-of-the-art meta-heuristic algorithms. Furthermore, the physical-based simulations in UE4 show that our method can be successfully deployed in simulation models to perform complex missions on the battlefield. Finally, a real-world experiment demonstrates that the proposed method is efficient and can be used for continuous real-time path planning for quadcopters. Although the performance of HHPSO is remarkable, it has not yet been able to handle confrontational scenarios, which means enemy aircraft will also be deployed to scout and defend. We will focus on such type of scenario in the future and we have built a simulation environment in UE4. Finally, we hope this work could facilitate the applications of intelligent algorithms in path planning.

Supplementary Materials: The following supporting information can be downloaded at: <https://www.mdpi.com/article/10.3390/drones8060221/s1>. Table S1: Binary encoding of all algorithms; Table S2: 11-bits genes for Initialization; Table S3: 13-bits genes for sorting and selection; Table S4: 12-bits genes for exploitation and exploration; Table S5: 13-bits genes for Ending Criterion; Table S6: 15-bits genes for other operators.

Author Contributions: Conceptualization, J.L. and R.D.; methodology, J.L.; software, J.L.; validation, J.L., R.D. and W.W.; formal analysis, J.L.; investigation, R.D.; resources, J.L.; data curation, R.D.; writing—original draft preparation, J.L.; writing—review and editing, R.D.; visualization, R.D.; supervision, W.W.; project administration, W.W.; funding acquisition, W.W. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the National Key R&D Program of China (No. 2022ZD0116401) and the State Key Laboratory of Software Development Environment (Funding No. SKLSDE-2023ZX-20).

Data Availability Statement: The datasets generated during the current study are not publicly available due to confidential agreement but are available from the corresponding author on reasonable request.

Conflicts of Interest: The authors declare no conflicts of interest.

References

- Huang, H.; Savkin, A.V.; Ni, W. Decentralized Navigation of a UAV Team for Collaborative Covert Eavesdropping on a Group of Mobile Ground Nodes. *IEEE Trans. Autom. Sci. Eng.* **2022**, *19*, 3932–3941. [[CrossRef](#)]
- Brunner, G.; Szebedy, B.; Tanner, S.; Wattenhofer, R. The Urban Last Mile Problem: Autonomous Drone Delivery to Your Balcony. In Proceedings of the 2019 International Conference on Unmanned Aircraft Systems (ICUAS), Atlanta, GA, USA, 11–14 June 2019; pp. 1005–1012.
- Dissanayaka, D.; Wanasinghe, T.R.; Silva, O.D.; Jayasiri, A.; Mann, G.K.I. Review of Navigation Methods for UAV-Based Parcel Delivery. *IEEE Trans. Autom. Sci. Eng.* **2022**, *21*, 1068–1082. [[CrossRef](#)]
- He, W.; Qi, X.; Liu, L. A novel hybrid particle swarm optimization for multi-UAV cooperate path planning. *Appl. Intell.* **2021**, *51*, 7350–7364. [[CrossRef](#)]
- Julius Fusic, S.; Ramkumar, P.; Hariharan, K. Path planning of robot using modified dijkstra Algorithm. In Proceedings of the 2018 National Power Engineering Conference (NPEC), Madurai, India, 9–10 March 2018; pp. 1–5.
- Pehlivanoglu, Y.V. A new vibrational genetic algorithm enhanced with a Voronoi diagram for path planning of autonomous UAV. *Aerosp. Sci. Technol.* **2012**, *16*, 47–55. [[CrossRef](#)]
- Meng, B. UAV Path Planning Based on Bidirectional Sparse A* Search Algorithm. In Proceedings of the 2010 International Conference on Intelligent Computation Technology and Automation, Changsha, China, 11–12 May 2010; Volume 3, pp. 1106–1109.
- Li, J.; Deng, G.; Luo, C.; Lin, Q.; Yan, Q.; Ming, Z. A Hybrid Path Planning Method in Unmanned Air/Ground Vehicle (UAV/UGV) Cooperative Systems. *IEEE Trans. Veh. Technol.* **2016**, *65*, 9585–9596. [[CrossRef](#)]
- Hangxuan, H.; Haibin, D. A multi-strategy pigeon-inspired optimization approach to active disturbance rejection control parameters tuning for vertical take-off and landing fixed-wing UAV. *Chin. J. Aeronaut.* **2022**, *35*, 19–30.
- Pehlivanoglu, Y.V.; Pehlivanoglu, P. An enhanced genetic algorithm for path planning of autonomous UAV in target coverage problems. *Appl. Soft Comput.* **2021**, *112*, 107796. [[CrossRef](#)]
- Phung, M.D.; Ha, Q.P. Safety-enhanced UAV path planning with spherical vector-based particle swarm optimization. *Appl. Soft Comput.* **2021**, *107*, 107376. [[CrossRef](#)]
- Yang, P.; Tang, K.; Lozano, J.A.; Cao, X. Path Planning for Single Unmanned Aerial Vehicle by Separately Evolving Waypoints. *IEEE Trans. Robot.* **2015**, *31*, 1130–1146. [[CrossRef](#)]
- Pan, J.S.; Liu, N.; Chu, S.C. A Hybrid Differential Evolution Algorithm and Its Application in Unmanned Combat Aerial Vehicle Path Planning. *IEEE Access* **2020**, *8*, 17691–17712. [[CrossRef](#)]
- Yu, X.; Li, C.; Zhou, J.F. A constrained differential evolution algorithm to solve UAV path planning in disaster scenarios. *Knowl.-Based Syst.* **2020**, *204*, 106209. [[CrossRef](#)]
- Li, D.; Wang, L.; Cai, J.; Ma, K.; Tan, T. Research on Terminal Distance Index-Based Multi-Step Ant Colony Optimization for Mobile Robot Path Planning. *IEEE Trans. Autom. Sci. Eng.* **2022**, *20*, 2321–2337. [[CrossRef](#)]
- Yu, X.; Chen, W.N.; Gu, T.; Yuan, H.; Zhang, H.; Zhang, J. ACO-A*: Ant Colony Optimization Plus A* for 3-D Traveling in Environments With Dense Obstacles. *IEEE Trans. Evol. Comput.* **2019**, *23*, 617–631. [[CrossRef](#)]
- YongBo, C.; YueSong, M.; JianQiao, Y.; XiaoLong, S.; Nuo, X. Three-dimensional unmanned aerial vehicle path planning using modified wolf pack search algorithm. *Neurocomputing* **2017**, *266*, 445–457. [[CrossRef](#)]
- Shao, S.; Peng, Y.; He, C.; Du, Y. Efficient path planning for UAV formation via comprehensively improved particle swarm optimization. *ISA Trans.* **2020**, *97*, 415–430. [[CrossRef](#)] [[PubMed](#)]
- Zhang, X.; Xia, S.; Li, X.; Zhang, T. Multi-objective particle swarm optimization with multi-mode collaboration based on reinforcement learning for path planning of unmanned air vehicles. *Knowl.-Based Syst.* **2022**, *250*, 109075. [[CrossRef](#)]
- Zhao, R.; Wang, Y.; Xiao, G.; Liu, C.; Hu, P.; Li, H. A method of path planning for unmanned aerial vehicle based on the hybrid of selfish herd optimizer and particle swarm optimizer. *Appl. Intell.* **2022**, *52*, 16775–16798. [[CrossRef](#)]
- Shin, J.J.; Bang, H. UAV path planning under dynamic threats using an improved PSO algorithm. *Int. J. Aerosp. Eng.* **2020**, *2020*. [[CrossRef](#)]
- Sanders, A. *An Introduction to Unreal Engine 4*; CRC Press: Boca Raton, FL, USA, 2016.
- Perlin, K. An Image Synthesizer. *SIGGRAPH Comput. Graph.* **1985**, *19*, 287–296. [[CrossRef](#)]
- Besada-Portas, E.; de la Torre, L.; Jesus, M.; de Andrés-Toro, B. Evolutionary trajectory planner for multiple UAVs in realistic scenarios. *IEEE Trans. Robot.* **2010**, *26*, 619–634. [[CrossRef](#)]
- Patel, J.S.; Fioranelli, F.; Anderson, D. Review of radar classification and RCS characterisation techniques for small UAVs or drones. *IET Radar Sonar Navig.* **2018**, *2*, 911–919. [[CrossRef](#)]
- Anderson, E.; Beard, R.; McLain, T. Real-time dynamic trajectory smoothing for unmanned air vehicles. *IEEE Trans. Control. Syst. Technol.* **2005**, *13*, 471–477. [[CrossRef](#)]
- Wu, X.; Bai, W.; Xie, Y.; Sun, X.; Deng, C.; Cui, H. A hybrid algorithm of particle swarm optimization, metropolis criterion and RTS smoother for path planning of UAVs. *Appl. Soft Comput.* **2018**, *73*, 735–747. [[CrossRef](#)]
- Elhoseny, M.; Tharwat, A.; Hassanien, A.E. Bezier curve based path planning in a dynamic field using modified genetic algorithm. *J. Comput. Sci.* **2018**, *25*, 339–350. [[CrossRef](#)]
- Zhou, X.; Zhu, J.; Zhou, H.; Xu, C.; Gao, F. EGO-Swarm: A Fully Autonomous and Decentralized Quadrotor Swarm System in Cluttered Environments. In Proceedings of the 2021 IEEE International Conference on Robotics and Automation (ICRA), Xian, China, 30 May–5 June 2021; pp. 4101–4107.

30. Qu, C.; Gai, W.; Zhang, J.; Zhong, M. A novel hybrid grey wolf optimizer algorithm for unmanned aerial vehicle (UAV) path planning. *Knowl.-Based Syst.* **2020**, *194*, 105530. [[CrossRef](#)]
31. Xue, Y.; Sun, J.Q. Solving the path planning problem in mobile robotics with the multi-objective evolutionary algorithm. *Appl. Sci.* **2018**, *8*. [[CrossRef](#)]
32. Kennedy, J.; Eberhart, R. Particle swarm optimization. In Proceedings of the ICNN'95—International Conference on Neural Networks, Perth, WA, Australia, 27 November–1 December 1995; Volume 4, pp. 1942–1948.
33. Zhou, X.; Wang, Z.; Ye, H.; Xu, C.; Gao, F. EGO-Planner: An ESDF-Free Gradient-Based Local Planner for Quadrotors. *IEEE Robot. Autom. Lett.* **2021**, *6*, 478–485. [[CrossRef](#)]
34. Zhang, X.; Duan, H. An improved constrained differential evolution algorithm for unmanned aerial vehicle global route planning. *Appl. Soft Comput. J.* **2015**, *26*, 270–284. [[CrossRef](#)]
35. Wang, J.; Chi, W.; Li, C.; Wang, C.; Meng, M.Q.H. Neural RRT*: Learning-Based Optimal Path Planning. *IEEE Trans. Autom. Sci. Eng.* **2020**, *17*, 1748–1758. [[CrossRef](#)]
36. Zheng, L.; Zhang, P.; Tan, J.; Chen, M. The UAV Path Planning Method Based on Lidar. In *Intelligent Robotics and Applications*; Yu, H., Liu, J., Liu, L., Ju, Z., Liu, Y., Zhou, D., Eds.; Springer: Cham, Switzerland, 2019; pp. 303–314.
37. Tao, X.; Guo, W.; Li, Q.; Ren, C.; Liu, R. Multiple scale self-adaptive cooperation mutation strategy-based particle swarm optimization. *Appl. Soft Comput.* **2020**, *89*, 106124. [[CrossRef](#)]
38. Sallhi, S.; Petch, R.J. A GA Based Heuristic for the Vehicle Routing Problem with Multiple Trips. *J. Math. Model. Algorithms* **2007**, *6*, 591–613. [[CrossRef](#)]
39. Giernacki, W.; Skwierczyński, M.; Witwicki, W.; Wroński, P.; Kozierski, P. Crazyflie 2.0 quadrotor as a platform for research and education in robotics and control engineering. In Proceedings of the 2017 22nd International Conference on Methods and Models in Automation and Robotics (MMAR), Międzyzdroje, Poland, 28–31 August 2017; pp. 37–42.
40. Guay, R.; Drolet, G.; Bray, J.R. Measurement and modelling of the dynamic radar cross-section of an unmanned aerial vehicle. *IET Radar Sonar Navig.* **2017**, *11*, 1155–1160. [[CrossRef](#)]
41. Shah, S.; Dey, D.; Lovett, C.; Kapoor, A. AirSim: High-Fidelity Visual and Physical Simulation for Autonomous Vehicles. In *Field and Service Robotics*; Hutter, M., Siegwart, R., Eds.; Springer: Cham, Switzerland, 2018; pp. 621–635.

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.