Journal of *Marine Science and Engineering*

MDPI

Article

A Bounded Near-Bottom Cruise Trajectory Planning Algorithm for Underwater Vehicles

Jingyu Ru [1], Han Yu [1], Hao Liu [2], Jiayuan Liu [2], Xiangyue Zhang [1] and Hongli Xu [1,*]

[1] School of Robot Science and Engineering, Northeastern University, Shenyang 110819, China
[2] School of Information Science and Engineering, Northeastern University, Shenyang 110819, China
* Correspondence: xuhongli@mail.neu.edu.cn

Abstract: The trajectory planning algorithm of underwater vehicle near-bottom cruise is important to scientific investigation, industrial inspection, and military affairs. An autonomous underwater vehicle (AUV) often faces the problems of complex underwater environment and large cruise area in a real environment, and some robots must hide themselves during the cruise. However, to the best of our knowledge, few studies have focused on trajectory planning algorithms for AUVs with multiple constraints on large-scale maps. The currently used algorithms are not effective at solving length-constraint problems, and the mainstream trajectory planning algorithms for robots cannot be directly applied to the needs of underwater vehicle sailing near the bottom. Therefore, we present a bounded ridge-based trajectory planning algorithm (RA*) for an AUV to go on a near-bottom cruise. In the algorithm, we design a safety map based on a spherical structure to ensure the safe operation of the robot. In addressing the length-constraint problem and large-scale map planning problem, this paper proposes a two-stage framework for RA*, which designs map compression and parallel computation using a coarse-fine planning framework to solve the large-scale trajectory planning problem and uses a bounded search method to meet the trajectory planning requirements of length constraint. In this study, experiments based on the virtual ocean ridge are conducted, and the results validate the effectiveness and efficiency of the proposed RA* with MCPC algorithm framework.

Keywords: trajectory planning; autonomous underwater vehicle; A-Star algorithm; parallel computation

Citation: Ru, J.; Yu, H.; Liu, H.; Liu, J.; Zhang, X.; Xu, H. A Bounded Near-Bottom Cruise Trajectory Planning Algorithm for Underwater Vehicles. *J. Mar. Sci. Eng.* **2023**, *11*, 7. https://doi.org/10.3390/jmse11010007

Academic Editor: Rafael Morales

Received: 6 November 2022
Revised: 13 December 2022
Accepted: 15 December 2022
Published: 21 December 2022

1. Introduction

With the rapid development of information technology and artificial intelligence technology, the application scenario of trajectory planning has changed to interpretable, large scale, and practical [1]. Trajectory planning for underwater vehicles, as a classical application scenario in related research fields, has been the classical research direction in related fields [2].

Trajectory planning can be divided into online planning and off-line planning, which correspond to different use scenarios [3]. Before the departure of an autonomous underwater vehicle (AUV), its navigation trajectory must be pre-planned on the basis of the requirements, which is suitable for offline trajectory planning algorithm [4]. In recent years, the off-line trajectory planning algorithm with special practical application requirements has gradually become a research hotspot [5,6]. In the existing research results, some studies aim at the multi-objective optimization objectives in trajectory planning, using the evolutionary algorithm to search a trajectory satisfying Pareto optimum [7]. Some studies focus on improving the efficiency of algorithms, which can deal with large-scale maps by establishing multiple heuristic functions [8,9]. Other studies aim to plan/optimize the trajectory based on the motion ability of objects [10]. However, these works still have some problems when being applied to a real environment. The evolutionary algorithm has strong randomness, and it cannot ensure the stability of trajectory planning results. Moreover, efficiency cannot be ensured in large-scale cases. Although the search-based

method can make up for the lack of predictability to a certain extent, the algorithm cannot run under the framework of multi-objective definitions. These limitations lead to the poor predictability of the algorithm, which leads to users' doubts about the practicability of the algorithm. At present, among the special requirements of many AUV, there is little need for sailing near the bottom of the ocean ridge, and terrain must be considered. Therefore, when facing large-scale maps, for example, the size reaches 10,000×10,000, the trajectory planning algorithm that meets the requirements of concealment and safety and defines the optimization goal has rarely been studied.

Studying the bottom-up path planning of bounded AUV for large maps has many benefits. With regard to quantitative optimization indicators, this planning algorithm can provide stable and convincing trajectory planning results. With regard to safety, it can provide a more accurate and safe distance for the AUV and reduce the probability of collision between the AUV and the ocean ridge during near-bottom navigation. In some special applications, the algorithm can also reduce the detection probability of sonar and camera by terrain which could reduce the cost and production difficulty [11].

In a complex seabed environment, the implementation of the abovementioned trajectory planning scheme primarily faces the following three challenges. First, designing a trajectory planning algorithm that satisfies the optimal distance condition and bound is a challenge. Second, considering the different definitions of safe distance in different scenarios, it has different effects on trajectory planning; thus, considering safety in planning is a challenge. Finally, computational efficiency is important for practicability. Traditional algorithms involve a trajectory planning compression algorithm that considers planning accuracy and computational efficiency. Furthermore, constraining the trajectory and splitting the algorithm into parallel processing methods is a challenge.

In addressing the above-mentioned challenges, we propose a novel trajectory planning framework: ridge-based A* trajectory planning algorithm, abbreviated as RA* algorithm. The algorithm (i) proposes a bounded trajectory planning algorithm, (ii) designs a safety map construction algorithm, and (iii) proposes a parallel bounded map planning framework. In particular, the contributions of this paper are as follows.

1. To our knowledge, this study is the first to explore the near-bottom trajectory planning of AUV in a large-scale oceanic ridge environment.
2. A bounded trajectory planning algorithm is proposed, which can combine the path length with near-bottom navigation based on the suboptimal bound of the path.
3. A fuzzy trajectory planning framework is proposed. The algorithm can create a mosaic of regional features based on scene requirements and then carry out distributed trajectory planning for each block through a parallel computing module.
4. We propose a general safety map construction algorithm that provides a safe navigation trajectory map for AUV, which can meet the needs of different application scenarios and adapt to complex seabed terrain environment.
5. In this paper, a large number of simulation experiments are carried out by using virtual ocean ridge maps, and the results are compared with those of classical trajectory planning algorithms. A case study is conducted for practical multi-attribute detection requirement projects, which further verifies the practicability and effectiveness of the algorithm.

2. Motivation

Our research was based on the experimental scenario of AUV simulation trajectory planning (Figure 1). First, in the typical AUV detection mission, AUV must cross the undulating oceanic ridge to detect the safety of any area in a long distance. The terrain environment to be detected is complex, and it cannot be described by a formula. The area is also large, and can reach 10,000 × 10,000 scale data.

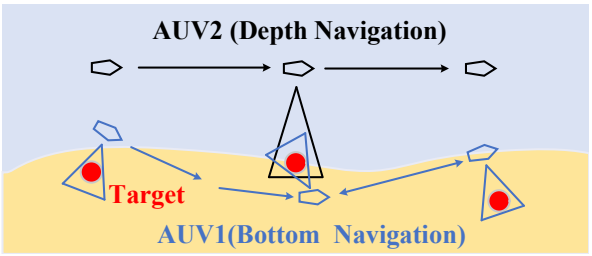


Figure 1. 2D exemplification of AUV simulation trajectory planning.

The planned results can cause the AUVs to be in a safe state. In this study, the "safe state" has two meanings. One meaning refers to navigation safety, that is, an AUV must meet a certain safe distance from obstacles when sailing. In general, AUVs will be disturbed by various conditions during driving, including the disturbance of surface ocean currents from the environment. AUVs usually have insufficient power [12,13]. When disturbed by ocean currents, they will continue to consume power to resist ocean currents. The other meaning describes how safe AUVs are for detection. When the AUV performs concealment tasks, it should minimize the exposure rate and avoid the interference and detection from underwater sonar. We hope that the AUV will fit the bottom of the ridge as much as possible during the voyage to avoid sonar irradiation by using undulating landforms. In addition, near-bottom navigation is more convenient for AUV to perform some minor tasks in navigation missions, such as scanning the bottom pipeline or finding other mission targets.

Therefore, we propose a large-scale and fine-grained trajectory planning algorithm and use geographical factors such as ridges to avoid ocean current disturbance and sonar detection.

2.1. Safety Distance Requirement

Making a global route plan under the precondition of security is necessary. In the traditional underwater safety trajectory planning, an AUV must keep a certain underwater cruise depth to avoid collision with the underwater bottom (Figure 2).

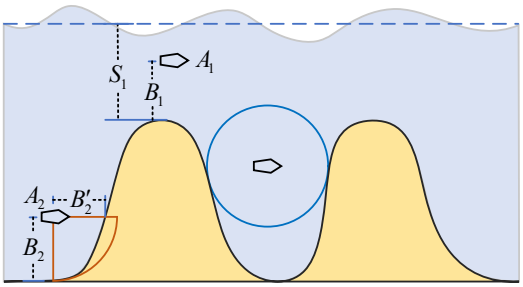


Figure 2. 2D exemplification of safety distance requirement.

As shown in Figure 2, the depth of AUV A_1 is S_1. A certain safe distance must be kept between AUV and the bottom B_1 to ensure safety. An evident deficiency is observed in the related work. In applications requiring near-bottom navigation, the abovementioned strategies cannot meet the requirements of lateral safety trajectory. For example, as shown in Figure 2, the horizontal safety distance b'_2 of AUV A_2 is evidently smaller than the vertical safety distance B_2. Therefore, considering the vertical safety distance is insufficient.

2.2. Benefits of Large-Scale and Fine-Grained Trajectory Planning

(1) Navigation safety The AUV can also use terrain to avoid specific "dangers" during traveling. For example, in military and other special purposes, if it can sail near the bottom based on terrain, then it can use the occlusion of the ridge to achieve the concealment effect; avoid sonar, photoelectric, and other detection; and increase the success rate of the mission.

In addition, in the actual use of medium and low-speed aircraft, certain collision risks will occur when traveling in areas full of obstacles because of different application requirements. Therefore, a navigation track must be planned to meet the safety requirements in advance and achieve higher safety standards.

(2) Planning interpretability During multi-objective trajectory planning, the optimization objectives include distance, safety, and fuel consumption. The commonly used method is heuristic intelligent search such as genetic algorithm or particle swarm optimization. The multi-objective optimization problem is transformed into single-objective optimization by coefficient weighting, and then the optimal solution is searched for or solved. However, during optimization, the abovementioned algorithm cannot limit the specified factors, and the optimization results have certain randomness and poor reproducibility. If the variables can be optimized in a bounded way, then the algorithm results can be reproduced. Considering that the solution is suboptimal, the risk is controllable. These methods are convenient for users to understand and increase the interpretability of the algorithm.

(3) Practicability of the system In the actual underwater exploration mission, multiple AUVs must detect multiple detection points cooperatively. In the framework of the cooperative detection system, the trajectory planning algorithm must be called frequently, and in the context of fine grains, the scale of the planning map is large, which proposes strict requirements for the efficiency and stability of the algorithm. Therefore, a stable, bounded, and efficient algorithm will inevitably increase the practicability of the trajectory planning system.

(4) Diversity of trajectories When performing tasks such as environmental detection, regional patrol, and object recognition in an area full of obstacles, the granularity and scale of environmental modeling are related to the navigation ability of AUV. In the traditional coarse-grained route planning algorithm, the carrier aims to float to the depth without obstacles and then travel in a fixed depth way. The fine-grained trajectory planning algorithm considers the terrain; thus, it can consider the security and terrain based on the experimental requirements. Consequently, the trajectory has more options and diversity.

3. Related Works

For the global path planning algorithm applied to AUV, researchers have proposed many commonly used algorithms, such as A* algorithm [14], genetic algorithm [15], particle swarm optimization algorithm [16], and other algorithms [3,17]. With the wide application of AUV in marine pollutant monitoring, pipeline detection, anti-submarine warfare, and other fields in recent years, the requirements for submarine path planning are increasing [18]. Yao et al. [19] proposed an AUV continuous-direction path planning method based on edge search algorithm. Considering energy consumption, their algorithm considers the influence of a strong current marine environment on AUV power while planning the path, thereby maximizing the use of energy. Sun et al. [20] abstracted the flow field as a static non-uniform vector field, combined directed graph with k-means method, and proposed GK-OPM algorithm to generate an AUV path with optimal time. Sang et al. [21] proposed a new MTAPF algorithm based on the improved APF and used the improved A* algorithm to obtain the global path. Yonetani et al. [22] proposed a neural A* algorithm that can be trained end to end. Yu et al. [23] proposed the Smooth-RRT algorithm to solve the problems of a complex working environment and high real-time requirements in AUV application scenarios, which meets the special requirements of AUV such as the shortest distance and maneuverability. Siddharth [8] used several different heuristic functions to improve the calculation speed of the path planning algorithm from the perspective of pure algorithm principle. Du et al. [9] proposed the simultaneous construction of multiple state spaces with different resolutions, which can improve the search speed while ensuring bounded sub-optimality. Ma et al. [24] proposed an alarm pheromone-assisted ant colony system (AP-ACS) that incorporates the alarm pheromone in addition to the traditional guiding pheromone. Roberge et al. [25] used a genetic algorithm and particle swarm optimization to generate UAV trajectory and achieved real-time path planning of

UAV by parallel computing. Oral et al. [26] proposed a multi-objective path planning algorithm based on multi-objective genetic algorithm, which improved the quality of path generation and replaced the traditional weighted optimal solution with Pareto optimal solution. Xi et al. [27] introduced real ocean current, weather, temperature, and other related information and used reinforcement learning to plan AUV in combination with the environment. Ru et al. [28] proposed an improved multi-objective genetic algorithm, which was combined with its three-dimensional omni-directional sensor, and designed an AUV planning method that can sense the surrounding environment to the greatest extent. Li et al. [29] proposed an improved DQN path planning algorithm to solve the problem of model tracking error and over-dependence in the planning process.

4. Model Design

In this paper, a new single-objective optimal path planning method is proposed, which can improve the traversal of map feature points while ensuring the bounded suboptimal total path length.

4.1. Problem Definitions and Concepts

4.1.1. Inconsistent State

In the following description, S represents all discrete states, including a starting state s_{start} and a target state s_{goal}. $g(s)$ represents the cost from the starting point to the state s, whereas $h(s)$ represents the expected cost from the state s to the end point. In this paper, the Euclidean distance is used to calculate the expected distance. In the A* algorithm, there is $f(s) = g(s) + h(s)$, where $f(s)$ represents the total generation value. The optimal cost from s_{start} to state s is represented as $g^*(s)$, and the function $c(s, s')$ represents the actual cost of the edge between states s and $c(s, s')$ (in the absence of an edge, $c(s, s') = \infty$). Assuming $c(s, s') \geq 0$, for all s and s', $SUCCS(s) = \{s' \in S | c(s, s') \neq \infty\}$, representing all successors of s. $pred(s)$represents all the predecessors of s, and $bp(s)$ represents the best predecessor of s.

In this paper, we introduce the concept of local inconsistency. In the common A* algorithm, all the extended states are in the local consistent state. For the state s, its $g(s')$ is calculated as follows:

$$g(s') = min_{s'' \in pred(s')}(g(s'') + c(s', s'')) = g(s) + c(s, s') \tag{1}$$

where $g(s) = g^*(s)$, and it will not be updated later. Therefore, this node will not be updated after joining the CLOSE table. The concept of local inconsistency indicates that the value of expanded nodes is not optimal, and it can be updated to a better value later, which is shown as follows:

$$g(s') > min_{s'' \in pred(s')}(g(s'') + c(s', s'')) \tag{2}$$

Therefore, in the common A* algorithm, except for the state with the smallest $f(s)$ value in the priority queue OPEN, other states may be locally inconsistent.

4.1.2. A* Algorithm Based on the Inconsistent State

As shown in Figure 3, from the beginning to the end, the expansion order of ordinary A* algorithm should be $A_3 \rightarrow B_2 \rightarrow C_1$, and its generation value $f(s_{goal}) = 2\sqrt{2}$. All the expansion states in this process are locally consistent. If the feature points must be passed during path planning, then the expansion order is $A_3 \rightarrow B_3 \rightarrow C_3 \rightarrow C_2 \rightarrow C_1$. At this time, $f(C_2) = 3$, $f(s_{goal}) = 4$, the states of C_2 and C_1 are locally inconsistent because the shortest path to C_2 is $A_3 \rightarrow B_3 \rightarrow C_2$ or $A_3 \rightarrow B_2 \rightarrow C_2$, $f(C_2) = 1 + \sqrt{2}$. This inconsistency will continue to pass backward with the expansion, resulting in $g(s) \neq g^*(s)$, and the length of the final path cannot be controlled. Therefore, if the number of feature points in the path is considered while planning the path, then the expanded state is locally inconsistent. Moreover, we cannot ensure the bounded sub-optimality of the path.

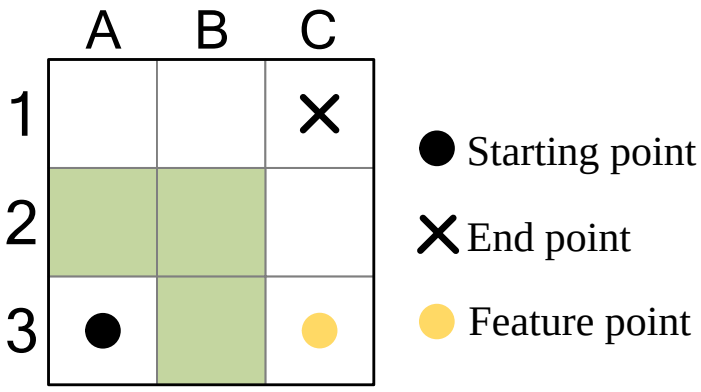


Figure 3. inconsistent expand of A*.

In this paper, an improved A* algorithm is proposed, which improves the traversal of feature points in the bounded suboptimal case. Compared with the traditional A* algorithm, this algorithm uses an extra priority queue and adopts two sets of cost functions $f(s) = g(s) + h(s)$ and $f_{incons}(s) = g_{incons}(s) + h(s)$ for all states, where $f(s)$ represents the cost of ordinary A* calculation, and $f_{incons}(s)$ represents the total cost of state s when considering characteristic points. For the state s $f(s) \leq f_{incons}(s)$, $f(s)$ is used to calculate the boundary problem in theory, and $f_{incons}(s)$ is used to calculate the actual distance when considering feature points, thereby considering the path length and the number of feature points in the path.

4.2. Construction of Safety Ridge Map

4.2.1. Safety Map

Constructing a safety map is the foundation of the system. Underwater navigation has many safety requirements, such as the vertical distance, horizontal distance, and vertical–horizontal mixed distance from terrain, which are all longer than the safety threshold. If the map data (point, edge, or surface) are expanded along the normal vector direction of the terrain, then it will bring a huge amount of computation, although the accurate expanded map can be obtained. Considering that all the safety maps constructed are grid data, this paper aims to construct a new map based on the original map and ensure that the shortest distance between every grid point on the new map and the seabed is longer than the safety threshold. Furthermore, the space above the level of the safety map is the safety space that meets the safety threshold requirements.

Each coordinate point s must be processed to generate a new safe coordinate point s_f. This paper summarizes three methods of building security maps to meet different kinds of security requirements, as shown in the following figures.

(1) Vertical safety map

A vertical safety map is the simplest and most commonly used safety map construction method, which is only suitable for scenes that are sensitive to vertical distance. All map points must increase the safety threshold d_s along the z axis, that is, $s_f(z) = s(z) + d_s$ (Figure 4a).

(2) Horizontal safety map

A horizontal safety map aims to add a horizontal safety criterion (horizontal safety threshold d'_s) based on the vertical safety map. This method is suitable for scenes that are sensitive to horizontal distance, and it has different requirements for horizontal and vertical safety distance. All grid points must find the highest point of elevation data within the range of their horizontal distance of d'_s, which is defined as s_d, and then calculate $s_f(z) = s_d(z) + d_s$ where d_s is the vertical safe distance, which is the vertical safe range of such problems. In addition, AUV must increase a pie-shaped space with a thickness of d_s along the ridge (Figure 4b). Thus, the safe distance directly below the AUV is longer than d_s, and the safe distance from the horizontal area is longer than d'_s.

(3) Spherical safety map

When the straight-line distance is used as the safe distance, the spherical safety map construction method must be used. This method is widely applicable, which is primarily considered in this paper, and the default method is considered for subsequent experiments. All grid points must be traversed in the range of horizontal distance d'_s. In addition, the highest point and the intersection point of vertical lines on the ground from s_d to s point must be defined as s_d and o, respectively, and then calculated to obtain $s_f(z) = \sqrt{d_s^2 - os_d^2} + os$ (Figure 4c).

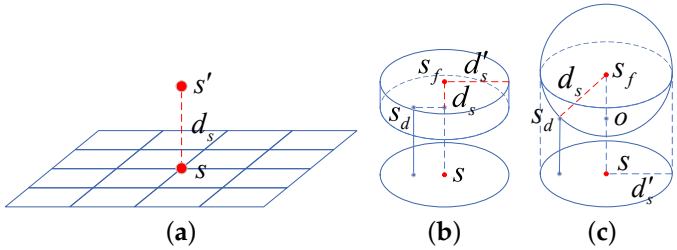


Figure 4. Description of different safety maps. (**a**) Vertical safety map. (**b**) Horizontal safety map. (**c**) Spherical safety map.

4.2.2. Construction of Oceanic Ridge Map

In this paper, the hydrological analysis method in ArcGIS toolbox is used to extract the ridge line from the digital elevation model (DEM) data. After binarization, a new map M_r with the same size as the original map is generated. In this map, $M_r(s) = 1$ indicates that the state s is located in the ridge part, and $M_r(s) = 0$ indicates that the state s is located in the non-ridge part. The effect of the algorithm is shown as follows.

4.3. Ridge-Based A Algorithm*

The process of the improved A* algorithm based on the oceanic ridge is shown as follows.

Steps of RA*

Step 1. ArcGIS is used to generate a matching ridge map based on the original DEM data.

Step 2. A priority queue $OPEN_{cost}$ and an array $OPEN_{nor}$ for storing all the explored states are constructed. The $OPEN_{cost}$ is sorted on the basis of $f(s)$, and the optimal state of $f(s)$ is s_0. ω is a weighted hyper-parameter that controls the boundary, and $Nor(s)$ represents the ridge point contained from the starting point to the state s. Let s_i represent the state in $OPEN_{nor}$ that conforms to $f_{incons}(s) < limit$ and $Nor(s)$ maximum. Among them, the $limit$ is defined as follows.

$$limit = \omega \cdot f(s_{cost}^{top}) \tag{3}$$

Step 3. s_{nor}^{top} is extended. Similarly, s_{cost}^{top} is extended when none of the states in $OPEN_{nor}$ meet the conditions.

Step 4. $SUCCS(s)$ returns all subsequent states of the extended state s and updates the subsequent states in accordance with the rules. The updated rules of $f(s)$ are the same as those of ordinary A* algorithm, and $f_{incons}(s)$ is updated in accordance with the condition of satisfying the descending order of Nor, which must satisfy $f_{incons}(s) < limit$.

Step 5. If the target point is not found, then the abovementioned steps 3 and 4 are repeated.

4.4. RA Details*

Algorithm 1 is the main function of RA* In the main function, lines 2 to 6 are initialization steps. First, a priority queue $OPEN_{cost}$ and an ordinary array $OPEN_{nor}$ are constructed, and the starting state s_{start} is assigned and added to the two priority queues. First, each

round of the loop must update *limit*, which is used as a constraint for subsequent extensions. In the selection of the current state, the top element of $OPEN_{nor}$ is selected by default. Lines 11–13 determine whether s_{nor}^{top} exists. If $OPEN_{nor}$ is empty or f_{incons} of all elements of $OPEN_{nor}$ is greater than *limit*, then s_{nor}^{top} does not exist, that is, degenerates into the original A* search algorithm. In addition, the current state is replaced by s_{cost}^{top}, and *limit* is continuously increased through subsequent updates until s_{nor}^{top} exists. Lines 14–15 indicate that if s_{cost}^{top} and s_{nor}^{top} are the same, then the extension of $OPEN_{cost}$ and $OPEN_{nor}$ is controlled simultaneously. Moreover, the top elements are replaced by other elements. For $OPEN_{cost}$, the cost of the top element is bound to increase, and subsequent extensions of the state in $OPEN_{nor}$ result in a larger *limit*. Consequently, we can update *limit* and extend the state in $OPEN_{nor}$ with just one extension, reducing the number of secondary extensions to the same state. Line 17 indicates that if s_{nor}^{top} exists, and s_{cost}^{top} is not the same as s_{nor}^{top}, then s_{nor}^{top} is extended separately to extend as many feature points as possible under constraints.

Algorithm 1: MAIN FUNCTION OF RA*

Input: s_{start}, s_{goal}

1 **begin**
2 $g(s_{start}) = 0\ g(s_{goal}) = \infty$
3 $bp(s_{start}) = NULL$
4 $OPEN_{cost}$ and $OPEN_{nor} \leftarrow \emptyset$
5 $CLOSE_{cost}$ and $CLOSE_{nor} \leftarrow \emptyset$
6 Insert s_{start} in $OPEN_{cost}$ and $OPEN_{nor}$
7 **while** $OPEN_{cost} \neq \emptyset$ **do**
8 $limit = \omega * f(s_{cost}^{top})$
9 $s = s_{nor}^{top}$
10 Insert s in $CLOSE_{cost}$
11 **if** $s == \emptyset$ **then**
12 $s = s_{cost}^{top}$
13 Insert s in $CLOSE_{cost}$
14 **else if** $s == s_{cost}^{top}$ **then**
15 Insert s in $CLOSE_{cost}$ ans $CLOSE_{nor}$
16 **else**
17 Insert s in $CLOSE_{nor}$
18 **end**
19 **if** $s == s_{goal}$ **then**
20 return path pointed by $bp(s_{goal})$
21 **end**
22 Exploration State(s)
23 **end**
24 **end**

Algorithm 2 describes the details of the exploration state. Line 3 determines whether the current extension node is the top element of $OPEN_{cost}$. If the result is true, then the operation is performed in accordance with the A* algorithm. Lines 4–12 are responsible for inserting the new state $s\prime$and updating $OPEN_{cost}$. Line 14 determines whether the current extension node is the top element of $OPEN_{nor}$. If the judgment is true, then the extension will maximize the number of feature points. As long as it meets$g_{incons}(s) + c(s, s') < limit$, it can be extended arbitrarily with the increased direction of *Nor*. The state s can serve as the predecessor node of s', and $g_{incons}(s')$and $Nor(s')$ can also be updated.

Algorithm 2: EXPLORATION STATE

Input: s

begin
  for *all* $s\prime \in SUCCS(s)$ **do**
    if $s == s^{top}_{cost}$ **then**
      if $s\prime$ *was never visited* **then**
        $bp(s\prime) = s$
        $g(s\prime) = g(s) + c(s, s\prime)$
        Insert $s\prime$ in $OPEN_{cost}$
      else if $g(s) + c(s, s\prime) < g(s\prime)$ **then**
        $bp(s\prime) = s$
        $g(s\prime) = g(s) + c(s, s\prime)$
        Update $s\prime$ in $OPEN_{cost}$
      end
    end
    if $s == s^{top}_{nor}$ **then**
      if $g(s) + c(s, s\prime) < limit$ **then**
        if $s\prime$ *was never visited* $\vee\ Nor(s\prime) < Nor(s) + valley(s\prime)$ **then**
          $bp(s\prime) = s$
          $g_{incons}(s\prime) = g(s) + c(s, s\prime)$
          $Nor(s\prime) = Nor(s) + valley(s\prime)$
          Insert or update $s\prime$ in $OPEN_{nor}$
        end
      end
    end
  end
end

Figure 5 provides a simple 2D illustration of the RA* algorithm. In this case, $\omega = 1.1$.

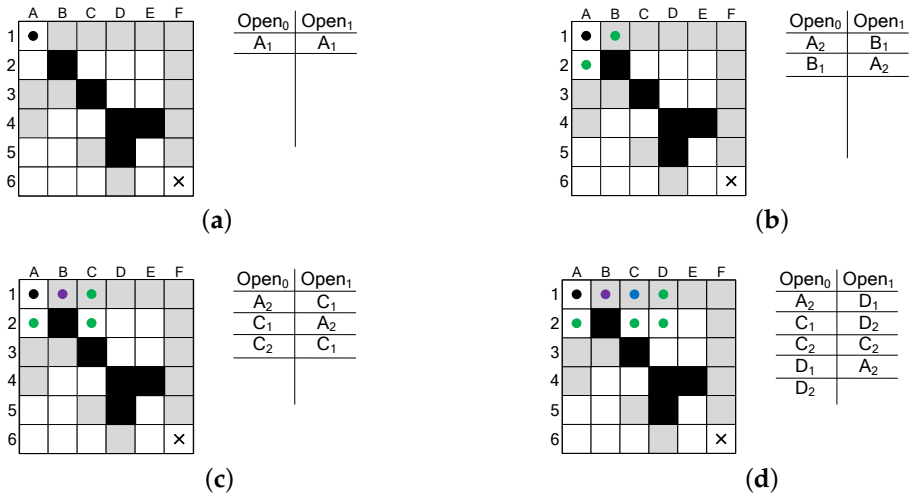


Figure 5. *Cont.*

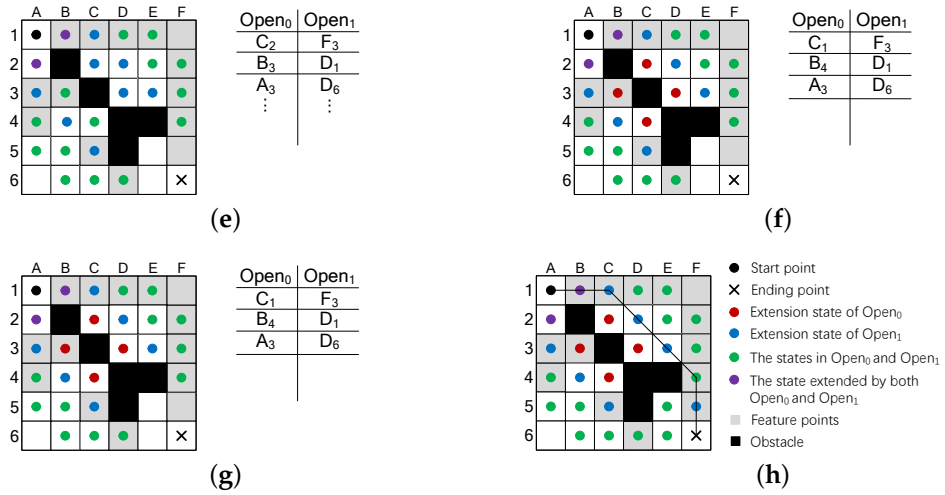


Figure 5. 2D illustration of the RA* algorithm. (**a**) Since there are only starting states in $OPEN_{cost}$ and $OPEN_{nor}$, the condition of simultaneous extension is met. After extension, $s_0 = f(b_1)$. (**b**) Same as the first step. (**c**) The ordinary A* algorithm will extend A_2, while our algorithm will give priority to the maximum state of Nor to extend, so as to ensure as many traversal of ridge points as possible, so it will extend C_1 at this time. (**d**) The effect after expansion is reflected in Figure 5d. (**e**) After several extensions, all the states in $OPEN_{nor}$ exceed the limit condition, then the limit needs to be further increased so that the states in $OPEN_{nor}$ can continue to expand, then s_0 is expanded separately. (**f**) Extend c_2 separately. When c_2 is extended, the top element of $OPEN_{cost}$ is replaced by other new states. Since f of c_2 is the smallest of $OPEN_{cost}$, f of other elements must be larger than f of c_2, so *limit* also increases. Repeat this step until the state in $OPEN_{cost}$ meets the boundary condition. (**g**) It is not until this state that some states in $OPEN_1$ meet the boundary conditions again. (**h**) This is the final state, and the black line is the final trajectory.

4.5. Map Compression and Parallel Computing

In this section, we introduce the algorithmic details of map compression and parallel computing (abbreviated as MCPC).

4.5.1. Map Compression

First, the surface is meshed and compressed, as shown in Figure 6.

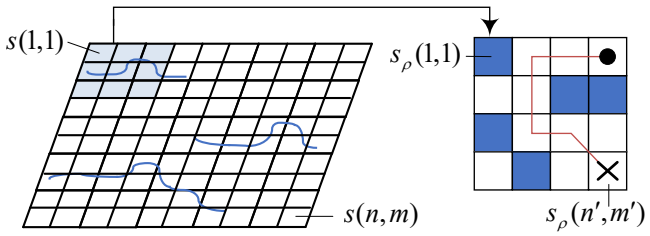


Figure 6. Schematic diagram of map compression.

Each complex surface consists of a basic mesh surface G, which consists of $n \times m$ sets $s_{1,1}, s_{1,2} \ldots s_{n,m}$, with $n \times m$ vertices. The height value of each surface is determined by its four vertices $g(i,j) = (s_{i,j}, s_{i+1,j}, s_{i,j+1}, s_{i+1,j+1})/4$. These points are compressed ρ times, and the compressed map becomes $g_p(1,1), g_p(1,2) \ldots g_p(n',m')$, where $g_\rho(i',j') = \gamma(g(i,j))/\rho$, and $\gamma(\cdot)$ is the average of the data for the range satisfying $\text{ceil}(i/\rho) \leq i' \leq \text{ceil}(i/\rho+1)$ and $\text{ceil}(j/\rho) \leq j' \leq \text{ceil}(j/\rho+1)$. In addition, each grid has a ridge point attribute $ridge(i,j)$, which is calculated in the same way as the elevation data. The compressed ridge point attribute is defined as $ridge_\rho(i',j')$, where it is the average of the data for the range satisfying $\text{ceil}(i/\rho) \leq i' \leq \text{ceil}(i/\rho+1)$ and $\text{ceil}(j/\rho) \leq j' \leq \text{ceil}(j/\rho+1)$. In addition, considering that each face has the ridge point attribute $ridge(i,j)$, which is

similar to elevation data, the compressed ridge point attribute is correspondingly defined as $ridge_\rho(i', j')$.

The starting point and end point of trajectory planning are also compressed in equal proportion. The compressed start and end points are as follows.

$$\begin{aligned} s'_{start}(x,y) &= s_{start}(x,y)/\rho \\ s'_{goal}(x,y) &= s_{goal}(x,y)/\rho \end{aligned} \tag{4}$$

First, use the A* algorithm to complete the rough planning trajectory from $s'_{start}(x,y)$ to $s'_{goal}(x,y)$, as shown by the red trajectory in Figure 6, which is represented by $\overrightarrow{s'_{start}, s'_{goal}} = \{s'_1, s'_2, \ldots, s'_k\}$. Then, we must cut the original map into pieces through the $\overrightarrow{s'_{start}, s'_{goal}}$.

4.5.2. Map Segmentation and Computing

Using the segmentation algorithm, the map can be segmented equally along the trajectory obtained by rough planning, and several block maps with the same size can be obtained, in which the starting point and ending point of the segmented trajectory are at the edge of the sub-map. The specific algorithm of the map segmentation algorithm is as follows.

First, the trajectory must be restored proportionally $\{\bar{s}_1, \bar{s}_2, \ldots, \bar{s}_k\} = \overrightarrow{s'_{start}, s'_{goal}} \cdot \rho = \{s'_1, s'_2, \ldots s'_k\} \cdot \rho$. The schematic diagram of the algorithm is shown in Figure 7a. First, based on the previous compression ratio ρ, the trajectory is reverse restored, and $\{\bar{s}_1, \bar{s}_2, \ldots, \bar{s}_k\}$ is obtained. Then, we use the idea of a sliding window to traverse all coordinate points and pause when the maximum length interval Δl or the maximum width interval Δw is larger than the threshold value T_s. If $\bar{s}_k$ satisfies the condition, then the point is saved from the starting time to the previous time $\bar{s}_{k-1}$ into t_{temp}. Moreover, the center point of t_{temp} is determined and defined as c_p, as shown by black dots in the figure.

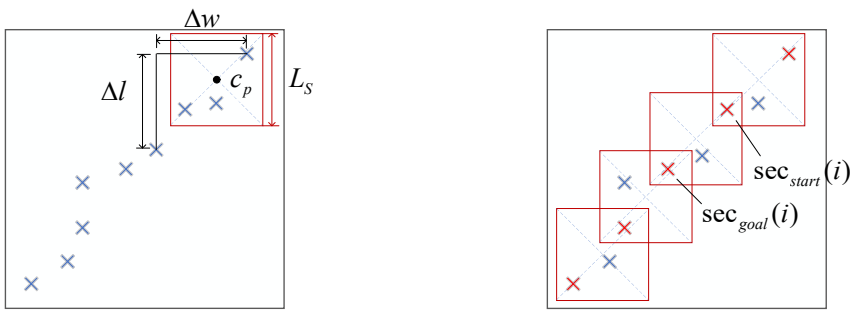


Figure 7. Map segmentation and parallel computing. (**a**) Map segmentation. (**b**) Parallel computing.

Then, the map and ith group data are clipped and saved into $Section[i]$, with c_p as the center and $L_s = max(\Delta l, \Delta w)$ as the side length, as shown in the red box in the figure. The four vertices of the square region are $\{c_p + (-L_s/2, -L_s/2), c_p + (-L_s/2, L_s/2), c_p + (L_s/2, -L_s/2), c_p + (L_s/2, L_s/2)\}$. Finally, the current vertex $\bar{s}_k$ is saved as the starting point of the next t_{temp} and iterated until all the points in $\{\bar{s}_1, \bar{s}_2, \ldots, \bar{s}_k\}$ are traversed. In addition, the starting point and ending point $\bar{s}_k$ in t_{temp} are recorded as the starting point and ending point $[\text{sec}_{start}(i), \text{sec}_{goal}(i)]$ of $Section[i]$ (Figure 7b).

Finally, different $Section[i]$ are sent to different CPU cores, and the results are calculated by the MRA* algorithm proposed in this paper with their respective starting points and ending points $[\text{sec}_{start}(i), \text{sec}_{goal}(i)]$. Finally, the sub-trajectories are connected end to end in the order of $Section[i]$ and spliced into the final trajectory. The split sections are sent to the cores of different CPUs, and the results are obtained by using the RA* algorithm mentioned in the previous section at their respective starting points and ending points. Algorithm 3 describes the details of the map segmentation and computation. Finally, the results are connected end to end and spliced into the final trajectory.

Algorithm 3: MAP SEGMENTATION AND COMPUTATION.

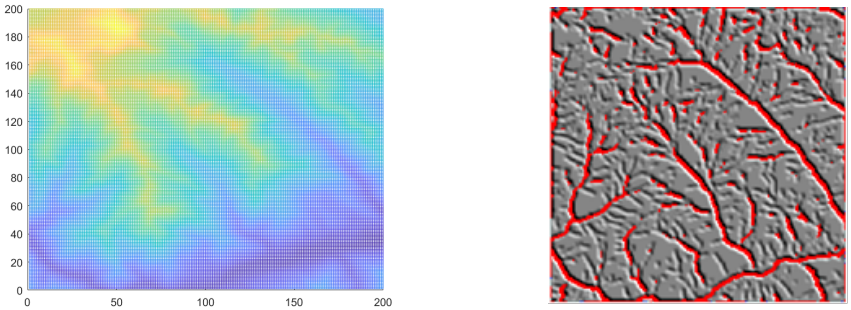
Input: $\{\bar{s}_1, \bar{s}_2, \ldots, \bar{s}_k\}$
begin
  for $i \leftarrow 1$ *to* k **do**
    Put $\bar{s}_k$ in t_{temp};
    Get $[x_{max}, y_{max}, x_{min}, y_{min}]$ in t_{temp};
    j=1;
    Δl=$x_{max} - x_{min}$; Δw=$y_{max} - y_{min}$;
    if $\Delta l >= T_s || \Delta w >= T_s$ **then**
      $L_s = \max(\Delta l, \Delta w)$;
      $c_x = (x_{max} - x_{min})/2$;
      $c_y = (y_{max} - y_{min})/2$;
      $c_p = (c_x, c_y)$;
      insert (c_p, L_s) and $[\mathrm{sec}_{start}(i), \mathrm{sec}_{goal}(i)]$ into section[j];
      $j = j + 1$;
      t_{temp}=[ ];
    end
  end
end

5. Evaluation

In this section, all simulation experiments are performed with MATLAB 2020A. The experiments were executed on a desktop with an AMD Ryzen 9-3950X central processing unit (CPU) @ 3.50 GHz and with a 64 GB memory.

5.1. Generation of Ridge Map

Figure 8a shows the topographic thermal map generated by MATLAB, with yellow representing shallow areas and blue representing deep areas. Figure 8b is the map generated by ArcGIS toolbox, which is formed by overlapping two layers, in which red represents ridge points ($M_r(s) = 1$), which are obtained by binary classification. Moreover, the remaining gray protrusions represent ordinary depressed areas, which are determined as non-ridge areas ($M_r(s) = 0$), because they are not enough to meet the criteria for judging ridges. The scale of the following experimental map is 1:30, that is, each pixel or grid represents 30 m.



(**a**) (**b**)

Figure 8. Ridge map generation. (**a**) Seabed topographic map. (**b**) Ridge map.

5.2. Evaluation of Safety Map

Figure 9 shows the comparison effect of the spherical safety map. The original map includes (200 × 200) DEM data, with a safety threshold of 300 m and a distance of 100 m between two points. As shown in the map, the basic topography can be preserved after processing. After calculation, the average height of the processed safety map increases by

338 m, and any distance between two points in the safety map and the ground are higher than 300 m. Table 1 shows the average fluctuation of the three safety maps when the safety distance is 200 and 300 m. As shown in the table, the horizontal safety consideration has the highest raising distance, whereas the shortest distance consideration primarily used in this paper has a moderate raising distance, which is higher than the vertical safety consideration and can meet the safety distance requirements of various application scenarios.

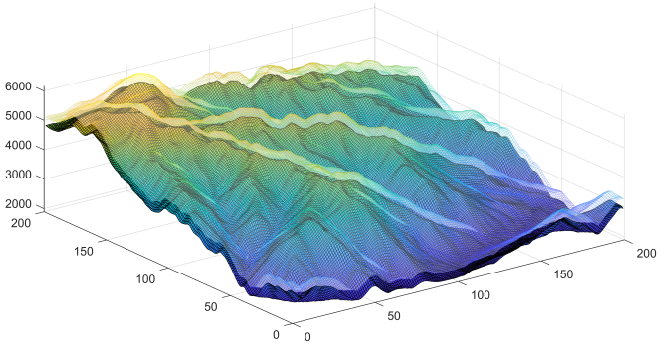


Figure 9. Map segmentation and parallel computing.

Table 1. Comparison of safety maps.

Safety Distance (m)	Spherical Safety Map (m)	Horizontal Safety Map (m)	Vertical Safety Map (m)
300	+338	+442	+300
200	+222	+342	+200

*5.3. Evaluation of RA**

This section describes the experiment on DEM data of 200 × 200. Figure 10a,b shows that when $\omega = 2.5$, the agent will advance along the ridge point as much as possible. A large number of ridge points are observed in the trajectory, which can achieve a good effect of traversing ridge feature points. However, the trajectory does not completely coincide with ridge points, which is limited by the overall cost, and it will make a compromise between them. When $\omega = 2$, the boundary constraint of the overall cost is greater. Based on the performance of the RA* algorithm, when ω decreases greatly, the overall trajectory cost will suddenly decrease, and the trajectory shape will change greatly. When ω decreases slightly, the overall trajectory does not change much, but some details are slightly adjusted. The abovementioned results show that this algorithm is sensitive to the change of ω, which can adapt to the change of boundary, utilize boundary constraints, and improve the traversal of ridge points as much as possible. When $\omega = 1$, RA* is equivalent to the original A* algorithm, which is consistent with the same planning result of Dijkstra.

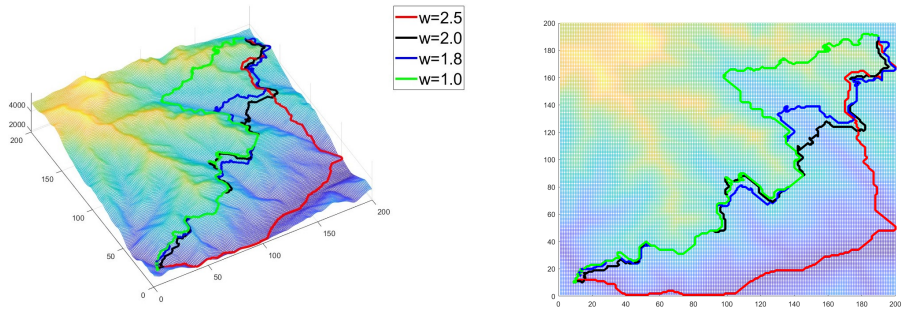


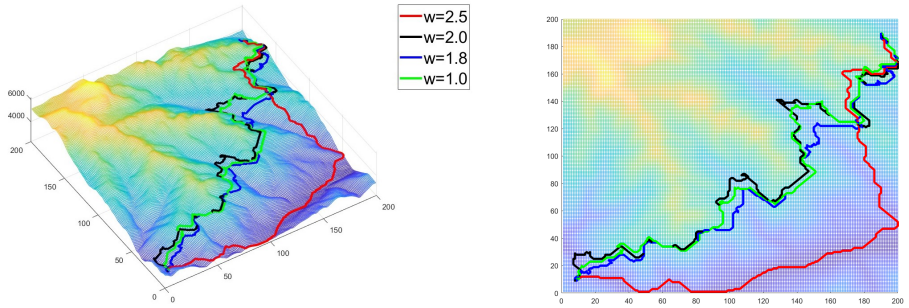
Figure 10. RA* without safety map. (**a**) Three-dimensional view. (**b**) Top view.

Three indexes are shown in Table 2, including the total path length, the length of ridge, and the ridge ratio, where the ridge ratio represents the ratio of the ridge length to the total length of the path.

Table 2. Data comparison of RA* without safety map.

	$\omega = 2.5$	$\omega = 2.0$	$\omega = 1.8$	$\omega = 1.0$
Total distance (km)	8.194	8.001	7.867	5.250
Length of ridge (km)	6.423	5.753	5.325	1.954
Ridge ratio (%)	78.4	71.9	67.7	37.2

The following part uses the same starting point, end point, and ω value as the above mentioned experiment and analyzes the trajectory planning results when only the spherical safety map is used. As shown in Figure 11 and Table 3, slight changes are observed in the total length of the path, the length of ridge, and ridge ratio by using the safety map, which can ensure that the path keeps the same safe distance from the terrain and will not have a great effect on the overall trend of trajectory planning.



(**a**) (**b**)

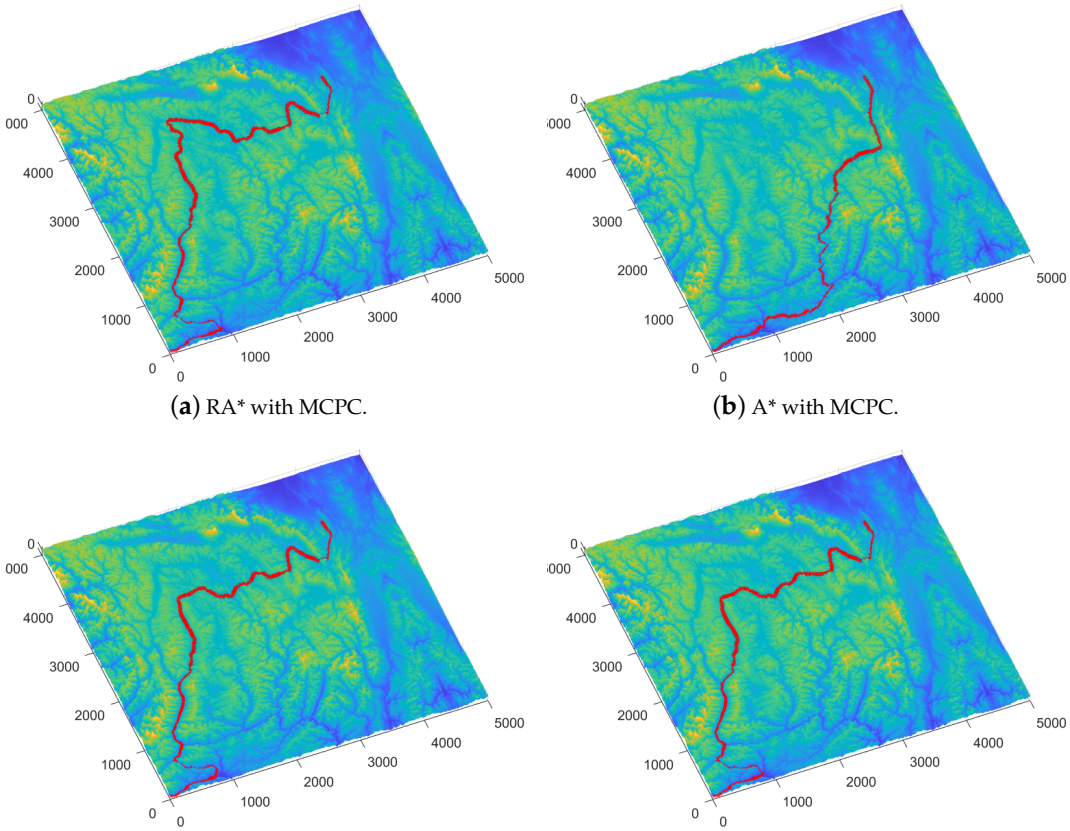
Figure 11. RA* with safety map. (**a**) Three-dimensional view. (**b**) Top view.

Table 3. Data comparison of RA* with safety map.

	$\omega = 2.5$	$\omega = 2.0$	$\omega = 1.8$	$\omega = 1.0$
Total distance (km)	8.389	8.129	7.963	5.463
Length of ridge (km)	6.563	5.646	5.268	2.047
Ridge ratio (%)	78.2	69.5	66.2	37.5

5.4. Evaluation of Map Compression and Parallel Computing

Based on the 5000 × 5000 large-scale map, this part compares the calculation results of RA* and original A* and analyzes the efficiency of the two methods with and without MCPC module. Figure 12 is an algorithm result based on RA*+MCPC, from which it can be seen that the algorithm in this paper greatly improves the calculation speed and path quality under the condition, that is, the total length of the ridge is constrained ($\omega = 3$), and more than half of the whole path is covered by ridge points, which improves the detectability of the feature points. Compared with the original A* algorithm, this algorithm sacrifices part of the optimality, but the calculation speed range is exponentially improved. The trajectory is only using the RA* algorithm. Although the quality of the path is the highest among the four paths, the search speed is too slow, which is not applicable to the planning application of robots in real working environment. However, compared with the original A* , the coverage ability of RA* to ridge points is greatly improved.



(**a**) RA* with MCPC. (**b**) A* with MCPC.

(**c**) RA* without MCPC. (**d**) Original A*.

Figure 12. Experiments of RA* and MCPC.

Four indexes are shown in Table 4, including the total path length, the length of ridge, the ridge ratio, and the running time of four different algorithms.

Table 4. Data comparison of different algorithms.

	RA* with MCPC	A* with MCPC	RA* without MCPC	Original A*
Total path length (km)	342.5	293.6	257.6	163.7
Length of ridge (km)	227.6	113.2	176	39.8
Ridge ratio (%)	66.45	38.54	68.32	24.31
Calculation time (s)	2.58	2.03	20.54	15.36

5.5. Evaluation of Computational Efficiency

This section verifies how the computational efficiency of the proposed algorithm is affected by the size of the map. As shown in Table 5, the average time spent on map sizes in 20 scales of $500 \times 500, 1000 \times 1000, \ldots$ 10,000 $\times$ 10,000 when $\omega = 2$ is shown. The map of each scale randomly selects 20 starting points and ending points for planning. As shown in the table, when the size of the map increases, the time spent by the planning algorithm increases linearly with the increase of the map size, reaching the size of 10,000 $\times$ 10,000. Using the experimental equipment in this paper, no dimension of disaster occurs.

Table 5. Comparison of computational efficiency.

Map size	500 × 500	1000 × 1000	1500 × 1500	2000 × 2000	2500 × 2500
Calculation time (s)	0.262	0.475	0.896	0.91	1.07
Map size	3000 × 3000	3500 × 3500	4000 × 4000	4500 × 4500	5000 × 5000
Calculation time (s)	1.35	1.48	1.53	1.76	1.89
Map size	5500 × 5500	6000 × 6000	6500 × 6500	7000 × 7000	7500 × 7500
Calculation time (s)	2.14	2.39	2.42	2.65	2.78
Map size	8000 × 8000	8500 × 8500	9000 × 9000	9500 × 9500	10,000 × 10,000
Calculation time (s)	2.83	2.91	3.02	3.04	3.12

As can be seen from the Figure 13, the calculation time increases linearly with the size of the map.

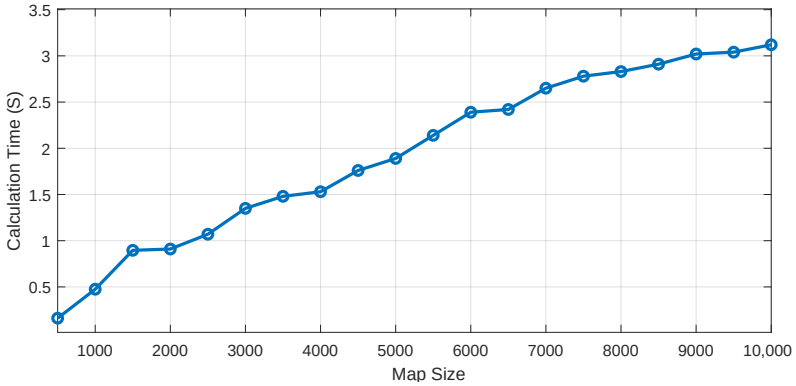


Figure 13. Calculation time vs. map size.

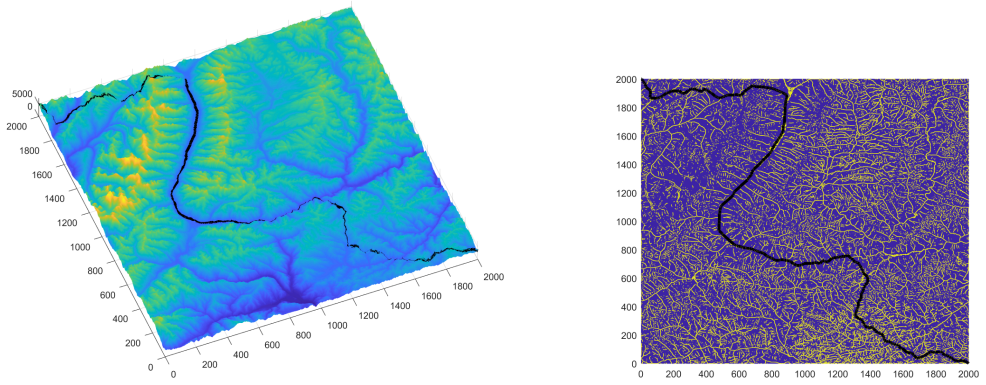
In this paper, the MRA* algorithm, RRT* algorithm, and WA* algorithm are selected and compared with the RA* algorithm ($\omega = 2$) proposed in this paper (Table 6). All the experiments are calculated on the map of 10,000 × 10,000 scale, and the average number of calculations is 20 times. All the algorithms have the same starting point and ending point at each calculation. Therefore, the trajectory generation method in this paper is faster than the current mainstream trajectory planning algorithm, but it has the characteristic of bounded detection of feature points, which is not available in traditional algorithms.

Table 6. Comparison with different algorithms.

	RA* ($\omega = 2$)	**MRA***	**RRT***	**WA***
Calculation time (s)	3.12	8.46	20.54	25.36
Cost (km)	356.4	232.3	351.5	281.4

Case Study

Finally, we show a specific case study on the 2000 × 2000 map (RA*+MCPC with $\omega = 3$). Figure 14a on the left shows a planned trajectory. The calculation time is 0.94 s, and the average safety distance is 341 m. Its starting point and ending point are the relative vertices of the square area. Figure 14b on the right shows the characteristic map of ridge points in this area. The yellow area in the map is ridge points, and the black solid line is the corresponding planning trajectory. Based on the two figures, the algorithm proposed in this paper can effectively and simultaneously complete the trajectory planning task and detect ridge points.



(**a**) RA*+MCPC($\omega = 3$). (**b**) Trajectory on the ridge feature map.

Figure 14. Case study of our algorithm.

6. Conclusions

In addressing the problem of trajectory planning for AUV near bottom cruise, this paper presents an improved heuristic algorithm and parallel computing scheme by combining global and local planning with a safety map to ensure the effectiveness of the algorithm considering safety factors. Extensive experiments have been performed to study the effectivity of our algorithm. The simulation results show that the trajectory generation method (RA*) in this paper can fully traverse task feature points under bounded suboptimal conditions while ensuring computational efficiency compared with other mainstream trajectory planning, which is essential in practical applications. This work considers both security and boundedness, and is suitable for functional operation, providing a feasible scheme for researchers in need.

Author Contributions: J.R. and H.L. proposed the idea of the paper; J.R. and H.Y. designed and performed the experiments; H.X. and J.L. retouched English and provide data analysis; X.Z. analyzed the data.; J.R., H.L. and H.Y. wrote the paper. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the National Natural Science Foundation of China (61872073, 62203099), the Fundamental Research Funds for the Central Universities (N2126005), the Fundamental Research Fund for the Central Universities of China (N2126002), the Fundamental Research Funds for the Central Universities (N2126006), the National Defense Preliminary Research Project(grant No.50911020604) and the Science Foundation of Liaoning (2021-MS-101).

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Data available on request from the authors.

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

References

1. Gasparetto, A.; Boscariol, P.; Lanzutti, A.; Vidoni, R. Path Planning and Trajectory Planning Algorithms: A General Overview. In *Motion and Operation Planning of Robotic Systems: Background and Practical Approaches*; Springer International Publishing: Cham, Switzerland, 2015; pp. 3–27.
2. Li, D.; Du, L. AUV Trajectory Tracking Models and Control Strategies: A Review. *J. Mar. Sci. Eng.* **2021**, *9*, 1020. [CrossRef]
3. Zeng, Z.; Lian, L.; Sammut, K.; He, F.; Tang, Y.; Lammas, A. A survey on path planning for persistent autonomy of autonomous underwater vehicles. *Ocean Eng.* **2015**, *110*, 303–313. [CrossRef]
4. Panda, M.; Das, B.; Subudhi, B.; Pati, B.B. A Comprehensive Review of Path Planning Algorithms for Autonomous Underwater Vehicles. *Int. J. Autom. Comput.* **2020**, *17*, 321–352. [CrossRef]
5. Ni, J.; Wu, L.; Shi, P.; Yang, S.X. A Dynamic Bioinspired Neural Network Based Real-Time Path Planning Method for Autonomous Underwater Vehicles. *Intell. Neurosci.* **2017**, *2017*, 10–25. [CrossRef] [PubMed]
6. Yan, Z.; Li, J.; Wu, Y.; Zhang, G. A Real-Time Path Planning Algorithm for AUV in Unknown Underwater Environment Based on Combining PSO and Waypoint Guidance. *Sensors* **2018**, *19*, 20. [CrossRef] [PubMed]

7. Yin, S.; Mao, J.; Li, B. AUV 3D Path Planning Based on Improved Empire Competition Algorithm in Ocean Current Environment. In *6th International Technical Conference on Advances in Computing, Control and Industrial Engineering (CCIE 2021)*; Springer Nature Singapore: Singapore, 2022; pp. 36–49.
8. Aine, S.; Swaminathan, S.; Narayanan, V.; Hwang, V.; Likhachev, M. Multi-Heuristic A*. *Int. J. Robot. Res.* **2016**, *35*, 224–243. [CrossRef]
9. Du, W.; Islam, F.; Likhachev, M. Multi-Resolution A*. In Proceedings of the Thirteenth Annual Symposium on Combinatorial Search, Vienna, Austria, 26–28 May 2020.
10. Zhang, G.; Liu, J.; Sun, Y.; Ran, X.; Chai, P. Research on AUV Energy Saving 3D Path Planning with Mobility Constraints. *J. Mar. Sci. Eng.* **2022**, *10*, 821. [CrossRef]
11. Kolev, G.; Solomevich, E.; Rodionova, E.; Kopets, E.; Rybin, V. Sensor subsystem design for small unmanned surface vehicle. *IOP Conf. Ser. Mater. Sci. Eng.* **2019**, *630*, 012022–012039. [CrossRef]
12. Qu, N.; Chen, G.; Shen, Y. A Three-Dimensional Path Planning System for AUV Diving Process Considering Ocean Current and Energy Consumption. In Proceedings of the OCEANS 2021, San Diego–Porto, San Diego, CA, USA, 20–23 September 2021; pp. 1–7.
13. Zhang, J.; Liu, M.; Zhang, S.; Zheng, R. AUV Path Planning Based on Differential Evolution with Environment Prediction. *J. Intell. Robot. Syst.* **2022**, *104*, 23. [CrossRef]
14. Carroll, K.; McClaran, S.; Nelson, E.; Barnett, D.; Friesen, D.; William, G. AUV path planning: An A* approach to path planning with consideration of variable vehicle speeds and multiple, overlapping, time-dependent exclusion zones. In Proceedings of the 1992 Symposium on Autonomous Underwater Vehicle Technology, Washington, DC, USA, 2–3 June 1992; pp. 79–84.
15. Alvarez, A.; Caiti, A. A Genetic Algorithm for Autonomous Undetwater Vehicle Route Planning in Ocean Environments with Complex Space-Time Variability. *IFAC Proc. Vol.* **2001**, *34*, 237–242. [CrossRef]
16. Yang, G.; Zhang, R. Path Planning of AUV in Turbulent Ocean Environments Used Adapted Inertia-Weight PSO. In Proceedings of the 2009 Fifth International Conference on Natural Computation, Tianjin, China, 14–16 August 2009; Volume 3, pp. 299–302.
17. Cheng, C.; Sha, Q.; He, B.; Li, G. Path planning and obstacle avoidance for AUV: A review. *Ocean Eng.* **2021**, *235*, 109355–109373. [CrossRef]
18. Li, D.; Wang, P.; Du, L. Path Planning Technologies for Autonomous Underwater Vehicles-A Review. *IEEE Access* **2019**, *7*, 9745–9768. [CrossRef]
19. Yao, P.; Zhao, S. Three-Dimensional Path Planning for AUV Based on Interfered Fluid Dynamical System Under Ocean Current. *IEEE Access* **2018**, *6*, 42904–42916. [CrossRef]
20. Sun, Y.; Gu, R.; Chen, X.; Sun, R.; Xin, L.; Bai, L. Efficient time-optimal path planning of AUV under the ocean currents based on graph and clustering strategy. *Ocean Eng.* **2022**, *259*, 111907–111923. [CrossRef]
21. Sang, H.; You, Y.; Sun, X.; Zhou, Y.; Liu, F. The hybrid path planning algorithm based on improved A* and artificial potential field for unmanned surface vehicle formations. *Ocean Eng.* **2021**, *223*, 108709–108735. [CrossRef]
22. Yonetani, R.; Taniai, T.; Barekatain, M.; Nishimura, M.; Kanezaki, A. Path Planning using Neural A* Search. In Proceedings of the 38th International Conference on Machine Learning, Virtual Event, 18–24 July 2021; Proceedings of Machine Learning Research. PMLR: London, UK, 2021; Volume 139, pp. 12029–12039.
23. Yu, L.; Wei, Z.; Wang, Z.; Hu, Y.; Wang, H. Path optimization of AUV based on smooth-RRT algorithm. In Proceedings of the 2017 IEEE International Conference on Mechatronics and Automation (ICMA), Kagawa, Japan, 6–9 August 2017; pp. 1498–1502.
24. Ma, Y.N.; Gong, Y.J.; Xiao, C.F.; Gao, Y.; Zhang, J. Path Planning for Autonomous Underwater Vehicles: An Ant Colony Algorithm Incorporating Alarm Pheromone. *IEEE Trans. Veh. Technol.* **2019**, *68*, 141–154. [CrossRef]
25. Roberge, V.; Tarbouchi, M.; Labonte, G. Comparison of Parallel Genetic Algorithm and Particle Swarm Optimization for Real-Time UAV Path Planning. *IEEE Trans. Ind. Inform.* **2013**, *9*, 132–141. [CrossRef]
26. Oral, T.; Polat, F. MOD* Lite: An Incremental Path Planning Algorithm Taking Care of Multiple Objectives. *IEEE Trans. Cybern.* **2016**, *46*, 245–257. [CrossRef]
27. Xi, M.; Yang, J.; Wen, J.; Liu, H.; Li, Y.; Song, H.H. Comprehensive Ocean Information-Enabled AUV Path Planning Via Reinforcement Learning. *IEEE Internet Things J.* **2022**, *9*, 17440–17451. [CrossRef]
28. Ru, J.; Yu, S.; Wu, H.; Li, Y.; Wu, C.; Jia, Z.; Xu, H. A Multi-AUV Path Planning System Based on the Omni-Directional Sensing Ability. *J. Mar. Sci. Eng.* **2021**, *9*, 806. [CrossRef]
29. Li, J.; Chen, Y.; Zhao, X.; Huang, J. An Improved DQN Path Planning Algorithm. *J. Supercomput.* **2022**, *78*, 616–639. [CrossRef]