

## Article

# An Improved Distributed Sampling PPO Algorithm Based on Beta Policy for Continuous Global Path Planning Scheme

Qianhao Xiao <sup>1</sup>, Li Jiang <sup>1</sup>, Manman Wang <sup>2</sup> and Xin Zhang <sup>1,\*</sup>

<sup>1</sup> School of Electronic Engineering, XI'AN University of Posts&Telecommunications, Xi'an 710121, China; xiaoqianhao@stu.xupt.edu.cn (Q.X.); jiangli@xupt.edu.cn (L.J.)

<sup>2</sup> School of Physical Science and Technology, Tiangong University, Tianjin 300387, China; 2031121198@tiangong.edu.cn

\* Correspondence: zhx@xupt.edu.cn; Tel.: +86-1814-937-0410

**Abstract:** Traditional path planning is mainly utilized for path planning in discrete action space, which results in incomplete ship navigation power propulsion strategies during the path search process. Moreover, reinforcement learning experiences low success rates due to its unbalanced sample collection and unreasonable design of reward function. In this paper, an environment framework is designed, which is constructed using the Box2D physics engine and employs a reward function, with the distance between the agent and arrival point as the main, and the potential field superimposed by boundary control, obstacles, and arrival point as the supplement. We also employ the state-of-the-art PPO (Proximal Policy Optimization) algorithm as a baseline for global path planning to address the issue of incomplete ship navigation power propulsion strategy. Additionally, a Beta policy-based distributed sample collection PPO algorithm is proposed to overcome the problem of unbalanced sample collection in path planning by dividing sub-regions to achieve distributed sample collection. The experimental results show the following: (1) The distributed sample collection training policy exhibits stronger robustness in the PPO algorithm; (2) The introduced Beta policy for action sampling results in a higher path planning success rate and reward accumulation than the Gaussian policy at the same training time; (3) When planning a path of the same length, the proposed Beta policy-based distributed sample collection PPO algorithm generates a smoother path than traditional path planning algorithms, such as A\*, IDA\*, and Dijkstra.

**Keywords:** artificial intelligence; path planning; reinforcement learning; deep learning



**Citation:** Xiao, Q.; Jiang, L.; Wang, M.; Zhang, X. An Improved Distributed Sampling PPO Algorithm Based on Beta Policy for Continuous Global Path Planning Scheme. *Sensors* **2023**, *23*, 6101. <https://doi.org/10.3390/s23136101>

Academic Editors: Sergio Toral Marín and Jesús Ureña

Received: 13 April 2023

Revised: 16 June 2023

Accepted: 30 June 2023

Published: 2 July 2023



**Copyright:** © 2023 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

Trade transportation is the basis of economic development, and, with the growth in trade volume in various countries, shipping accounts for more than 2/3 of international cargo transportation. Accidents, however, have been continuously posing risks to maritime transportation; according to one survey, more than 80% of accidents are caused by human factors in the process of ship navigation [1]. Path planning, as one of the key technologies for traditional autonomous navigation scheme, aims to find a safe and economic path for ships and has received increasing attention from both industry and academia in recent years [2].

The plan of the path provides the optimal route based on various factors such as distance, time, etc. Considering divergent factors, path planning can be divided into local path planning and global path planning [3]. Local path planning indicates that the changes in the surrounding environment are unknown (mainly reefs and dynamic obstacles), and the path needs to be corrected based on sensor data in real time. Global path planning involves planning a route from the starting point to the endpoint, founded on known environmental information at a large geographic scale (1 km to 100 km level, or more).

This paper presents an in-depth study of global path planning. Traditional path planning algorithms currently used to solve global path planning include the A\*, Dijkstra, and

artificial potential energy field methods. Most of these algorithms use spatial discretization to represent the environment as equal-sized raster maps and discretize the action space into four-connected or eight-connected [4]. The limitations of these path planning algorithms include the following. First, these algorithms rely on prior knowledge of the system when used for global path planning. Consequently, if there are changes in the environment, such as the emergence of new obstacles, it is necessary to obtain updated and accurate information about the environment in order to re-plan the route. In addition, these algorithms do not take into account the ship's navigational capability, only searching the driving path without giving the bow heading and velocity vector of the nodes in the path. Moreover, these algorithms search the way topologically, leading to much higher computation and lower efficiency required for planning with the increasing complexity in the planning scene.

To address the challenges, as mentioned before, of traditional path planning, Sun proposed a dynamic trajectory generation model for ship rust removal robots based on a bionic neural network and utilized a kinematic state matching search method to achieve dynamic path planning while considering the robot's kinematic model [5]. However, during actual testing, it was observed that the proposed method lacked robustness in the planning process, making it unable to guarantee successful planning. To improve the robustness and stability of the algorithm in the path planning task, Shen et al. from Dalian Maritime University proposed an intelligent collision avoidance navigation method for unmanned ships, which considered navigation experience and collision avoidance rules using Deep Competitive Q-learning (DCQN) with the A\* algorithm [6]. Similarly, Wang et al., also from Dalian Maritime University, proposed the DRLOAD algorithm based on the Deep Q-Network (DQN), which achieved a multi-dimensional navigation by designing a reward function specifically for safe obstacle avoidance [7]. Both methods employed value learning methods in reinforcement learning to solve path planning problems in a discrete action space. Nevertheless, these approaches faced issues with the reward function's design and the ship propulsion strategy's incomplete structure during the planning process. Another relevant contribution is the path planning algorithm proposed by Kuczkowski, which is based on the Deep Deterministic Policy Gradient (DDPG) algorithm. This approach integrates several subgoals of path planning, collision avoidance, compliance with the Convention on the International Regulations for Preventing Collisions at Sea (COLREGs), and task completion by designing four reward functions [8]. Kuczkowski successfully achieved path planning and dynamic obstacle avoidance that satisfied COLREGs in complex scenarios. While adequately designing the reward functions and providing each path node's coordinates and velocity vectors during planning, the environment modeling did not consider the ship's kinematic model.

A summary of the above related work is presented in Table 1.

Based on the above research work, this paper proposes a distributed sampler PPO algorithm based on Beta policy to address the pain points, i.e., that traditional path planning increases computationally in the case of rising scenario complexity and cannot directly deal with high-dimensional data. Furthermore, it addresses the problems that reinforcement learning algorithms face in solving path planning problems, such as the unreasonable design of the reward function—most algorithms only perform path planning in the discrete action space and primarily use deterministic policy in planning. The main technical contributions of this paper are as follows:

1. Designing and constructing a simulation environment that conforms to the ship propulsion characteristics, using the Box2D physics engine as the framework;
2. Constructing a reward function that integrates the mainline reward and auxiliary reward, where the mainline reward is determined based on the distance between the agent and the arrival point, while the auxiliary reward is potential-based, superimposed from boundary control, obstacles, and arrival point;
3. Proposing a distributed sampling policy on the basis of the stochastic policy algorithm PPO, which significantly enhances sampling performance and enables robust ship path planning in continuous domain spaces;

4. Introducing the Beta policy as a replacement for the Gaussian policy in the action selection policy. This substitution effectively resolves the issue of action boundary constraints.

**Table 1.** Related work.

Algorithm	Deterministic/ Stochastic Policy	Action Space	Advantages	Disadvantages
Traditional Path Planning	Deterministic	Discrete	High computational efficiency; easy to adjust algorithm parameters.	<ul style="list-style-type: none"> <li>• Unable to directly process high-dimensional data;</li> <li>• As the size of the scene map increases, the calculation increases;</li> <li>• Unable to provide the recommended sailing information in path planning;</li> <li>• Requires accurate environmental modeling;</li> <li>• The ship kinematic model is not considered.</li> </ul>
Sun Proposed	Deterministic	Discrete	Consider the ship's dynamics model.	<ul style="list-style-type: none"> <li>• Discrete action space.</li> </ul>
DCQN + A*	Deterministic	Discrete	Shorter convergence time by combining Q-learning and A*.	<ul style="list-style-type: none"> <li>• Discrete action space;</li> <li>• Dynamic model simplification limits propulsion strategy design comprehensiveness;</li> <li>• The reward function design lacks reasonability and overlooks the planning boundary problem.</li> </ul>
DRLOAD	Deterministic	Discrete	Propose safe obstacle avoidance and target approach (reach point approach) reward function.	<ul style="list-style-type: none"> <li>• Discrete action space;</li> <li>• Ship navigation capability is not considered in planning;</li> <li>• The reward function design is too simple.</li> </ul>
Kuczkowski Proposed	Deterministic	Continuous	Design the reward functions for different sub-objectives.	<ul style="list-style-type: none"> <li>• The ship kinematic model is not considered.</li> </ul>

The advantages of the algorithm proposed in this paper are reflected as follows. By employing the stochastic policy algorithm, this paper achieves better planning results by simulating a ship's propulsion characteristics in a continuous state and action space. The integration of the mainline and auxiliary rewards results in a smooth trajectory, with each planned path node providing position and velocity vectors. Our proposed algorithm attains shorter planning paths than other reinforcement learning algorithms. Moreover, when compared to traditional path planning methods with similar path lengths, our algorithm generates smoother trajectories.

## 2. Background

In solving global path planning problems by reinforcement learning, the learning tasks chosen vary for the different discrete and continuous action spaces in global path planning. Reinforcement learning is usually described by a Markov decision process (MDP), which consists of a state-space  $\mathcal{S}$  and an action space  $\mathcal{A}$ . In MDP, the individual who performs the learning and decision-making is called the agent, and the range with which the agent is interacting is called the environment. At the discrete moment  $t$ , the agent observes the states  $S_t (S_t \in \mathcal{S})$  fed by the environment and gives feedback actions  $A_t (A_t \in \mathcal{A})$  to all these states, according to the policy  $\pi$ , where the policy can be deterministic or stochastic. The environment provides a reward  $R_t$  in response to the action and moves to the next state  $S_{t+1} \in \mathcal{S}$ ; the state-action pairs generated by this agent-environment interaction constitute a trajectory:

$$S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, R_2, \dots, S_n, A_n, R_{n+1}$$

The learning tasks of reinforcement learning (RL) can be divided into value-based and policy-based categories.

### 2.1. Value-Based Methods for Discrete Domain

Traditional RL uses the Q-learning method and stores the transfer state by Q-table when applied to path-planning tasks. To solve the problem that, when the state complex increases, the efficiency of the Q-learning update drops, DQN, as proposed in 2013, adds the deep neural network and experience replay based on Q-learning [9]. Although it improves the policy performance dramatically and can be used to handle high-dimensional state-space tasks, DQN generates an unavoidable overestimation problem in the process of action-value Q estimation, which occurs because DQN performs action-value estimation based on the Bellman Equation (1):

$$Q^*(s, a) = E_{\pi} \left[ r_{t+1} + \gamma \cdot \max_{a \in \theta'} Q^*(s_{t+1}, a'; \theta'_t) \right] \quad (1)$$

In this context, the action-value function  $Q^*(s_t, a_t)$  is defined as the expected value of the immediate reward  $r_{t+1}$  obtained by taking action  $a_t$  in the current state  $s_t$ , plus the expected value with discounted maximum action-value  $Q^*(s_{t+1}, a'; \theta'_t)$  of the next state  $s_{t+1}$ , corresponding to the action  $a'$  chosen by the target policy  $\pi$  with parameters  $\theta'_t$ .

Here,  $\theta'_t$  represents the target network parameters for the policy  $\pi$ , and  $\gamma$  is the discount factor  $0 \leq \gamma < 1$ . The target network always selects the action that maximizes the action-value  $Q^*(s_t, a_t)$  according to its own policy  $\theta'_t$ , which leads to bootstrap. Silver et al. [10] replaced the bootstrap action generated by the target network with online network policy to update the action value estimation:

$$Q^*(s, a) = E_{\pi} [r_{t+1} + \gamma \cdot Q(s_{t+1}, \operatorname{argmax}_{a \in \theta} Q(s_{t+1}, a'; \theta_t); \theta'_t)] \quad (2)$$

The online network  $Q(s_{t+1}, a; \theta_t)$  is used for action selection, and the corresponding action is used for target calculation; due to a delay in the update of the network target, the actions that correspond to the value q-max of the online network are not necessarily identical to the actions that correspond to the value q-max in the network target. Thus, the problem of overestimating is reduced.

### 2.2. Policy-Based Method for Continuous Domain

This value-based learning method cannot solve the stochastic policy problem well because it is a deterministic policy in essence. The policy gradient-based [11] reinforcement learning algorithm optimizes a policy function by fitting a continuous function representing the current task performance metric and using gradient ascent. In this state, the value function can be used to assist the policy training, without need for action choice. The policy gradient-based approach has more robust convergence than the action-value-based approach; the policy gradient  $\nabla_{\theta} J(\pi_{\theta})$  is expressed as follows:

$$\begin{aligned} \nabla_{\theta} J(\pi_{\theta}) &= \int_{\mathcal{S}} \rho^{\pi}(s) \int_{\mathcal{A}} \nabla_{\theta} \pi_{\theta}(a | s) Q^{\pi}(s, a) d a d s \\ &= \int_{\mathcal{S}} \rho^{\pi}(s) \int_{\mathcal{A}} \pi_{\theta}(a | s) g_q d a d s \\ &= \mathbb{E}_{s \sim \rho^{\pi}, a \sim \pi_{\theta}} [g_q] \end{aligned} \quad (3)$$

where  $\rho^{\pi}(s)$  is the state distribution,  $\pi_{\theta}(a | s)$  denotes current state  $s$  according to the policy sampled by the parameter  $\theta$ , and  $g_q$  represents the policy gradient estimator with  $Q_{\pi}(s, a)$  as the target:

$$g_q = \nabla_{\theta} \log \pi_{\theta}(a | s) Q^{\pi}(s, a) \quad (4)$$

In the actual calculation operation, the policy gradient is estimated by sampling  $g_q$  in large numbers, so the sample mean of  $g_q$  converges to its expected value  $E[g_q] = \nabla_{\theta} J(\pi_{\theta})$  when the number of samples is sufficient.

However, the policy gradient based method has the problems of falling into local optima and inefficient policy evaluation. To solve these problems, Sutton et al. proposed Actor–Critic. The Actor refers to the policy function, i.e., the agent action decision part. Critic refers to the value function that can be regarded as an evaluator of the current state. The Critic and Actor parameters are updated by the data obtained from the real-time interaction between the agent and the environment, which makes the algorithm converge.

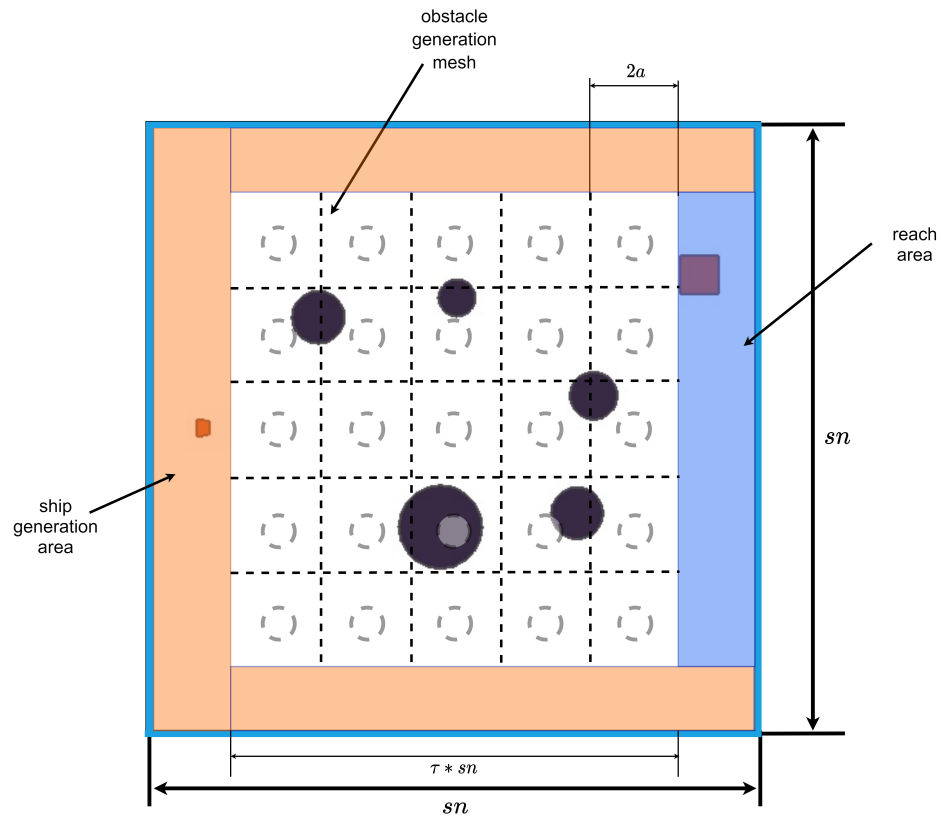
Schulman et al. proposed the Proximal Policy Optimization (PPO) algorithm [12] by using the Trust Region Policy Optimization (TRPO) [13] surrogate objective; the PPO clips the surrogate objective to a smaller range and replaces the advantage function  $A^{\pi}(s, a)$  in the agent with a Generalized Advantage Estimator (GAE) [14], which is used for high-dimensional continuous action training by GAE to obtain stable policy. To ensure continuous exploration, even in states where the policy is relatively mature, PPO incorporates entropy rewards when calculating the surrogate objective. Additionally, the inherent advantage of its stochastic policy in path exploration prevents the algorithm from ending up trapped in local optima during the search process [15].

This paper uses PPO as the baseline algorithm for global path planning implementation and optimization. Firstly, we conducted path planning environment design, and in the environment design process, we needed to consider the state-space and action space structure for deep reinforcement learning. For the algorithm to converge properly, we needed to design the reward function to be highly matched for the state-space of the global path planning task. The rationality of the reward function design directly guides the algorithm's convergence speed and policy performance. The PPO limits the policy update magnitude by clipping the surrogate objective, so that the old and new policies do not produce significant changes in the update and, thus, obtain more stable policy updates.

### 3. Interaction Environment Design

Deep reinforcement learning updates the state through constant interaction between the agent and the environment, and optimizes the policy to maximize the reward. This method requires environment design, including state-space, action space, and reward function. The environment design determines the scene structure and interaction logic and adds constraints to each scene element.

The simulation environment is constructed using the Box2D physics engine for structural development, and the Pyglet module is utilized for rendering the plane frame. The size of the plane frame structure is set as  $n * n$ , with a render scale of  $s$ . Obstacles are generated at the center of the frame, using a generation scale of  $\tau$ . The obstacle area is divided into a grid space of  $k * k$ , with a grid radius of  $a$ . The obstacle radius ranges from  $0.2a$  to  $1.5a$ , and the number of obstacles generated falls within the range  $[k, k^2 - k]$ . To prevent overlapping, an offset is applied to the obstacle generation coordinates. This offset follows a uniform distribution  $U(-n * \tau * 0.05, n * \tau * 0.05)$ , ensuring that no obstacles overlap with each other. Unlike other path planning simulation scenarios for RL, this simulation environment increases the ship and the endpoint generation range to improve the algorithm's robustness. As shown in Figure 1, the orange-covered area is the hull generation area, and the blue area is the arrival point generation area.



**Figure 1.** Location of simulation environment generation.

### 3.1. State-Space Design

The state information used for the global path planning task of deep reinforcement learning can be divided into the sensor data sequence and the image from the environment. DRL extracts the relationship between the features and the policy from training. Due to the sensor data sequence's low dimension, it is easy for the agent to acquire the policy according to the shift features; the image contains a higher dimensionality message, so the agent needs to learn the policy from the image extracted feature. As the network processes the state information differently, we designed two state-spaces, described below.

#### 3.1.1. Environmental Feedback Sensor Data Sequence-Based State-Space

To make the agent perceive the feature shifts in the motion, we designed an approach using LIDAR to measure the distance to avoid obstacles, as shown in Figure 2. We set the ship's radius as  $b$ ; the effective reflection range of LIDAR is within  $h$  times the vessel's radius, expressed as  $[b, h * b]$ . We set the horizontal field angle of the detection rays to  $\rho$  and uniformly arranged the  $l$  ray beams within the field. During the ship's motion, if another object truncates the ray, it is considered that the object is closing. In this case, we gave the ray identifier  $flag_i$  associated with the intercepted object, as shown in the following equation:

$$flag_i = \begin{cases} -1, & \text{object detected} \\ 0, & \text{otherwise} \\ 1, & \text{current goal} \end{cases} \quad (5)$$

Calculate the distance to the truncation point, denoted as  $ray_i$ ; the vector is shown in Equation:

$$G = \{(flag_1, ray_1), (flag_2, ray_2), \dots, (flag_l, ray_l)\} \quad (6)$$



and the image size is appropriately reduced to  $[1, 1, x, x]$  ( $x < sn$ ) for policymaking, while the grayscale image is normalized in the image processing.

The image-based state-space consists of the following vectors:

- Current scenario seascape: image data in the shape of  $[1, 1, x, x]$  with normalization operation;
- Position vector: spatial coordinate information  $(x_i, y_i)$  where the current ship is located;
- Velocity vector: obtain the current ship's velocity vector information  $(v_x, v_y)$ .

We keep the same data overlay operation with the data sequence state-space and concatenate the two groups of processed image and navigation feature data along the direction of  $\text{dim} = 1$  (the image shape is  $[1, 1, x, x]$  and the navigation data shape is  $[1, 1, 4]$ ). Finally, we obtain the overlaid feature data with the image shape  $[1, \text{frame}_{\text{overlay}}, x, x]$  and the navigation data shape  $[1, \text{frame}_{\text{overlay}} * 4]$ .

### 3.2. Action Space Design

The Box2D physics engine is used in the environmental design for ship action control. The simulated trajectory of the ship's rigid body is calculated by the semi-implicit Euler method to generate a planning curve in path planning that matches its physical characteristics for simulated propulsion control. To ensure accurate planning paths, the action space is designed in two dimensions, i.e., angle and propulsion power:

1. Angle dimension: In this path planning model, the angle design uses continuous space, ranging from 0 to 360 degrees, and is mapped normalized to the interval  $[0, 1]$ ; differently from traditional discrete angle design (such as four-connected or eight-connected actions), the continuous angle control allows the algorithm policy to perform more accurate navigation control to achieve more precise planning;
2. Propulsion power dimension: Propulsion power is the power generated by the ship's thrusters. In this model, we use the Box2D to complete the propulsion power design, which considers the effects of thrust derating and fluid resistance. The maximum value of the main propulsion power is set to 70. Consistent with the angle treatment, the propulsion power is also normalized and mapped to the interval  $[0, 1]$ .

In summary, the action space is designed with continuous control in angle and propulsion power dimensions. The agent can flexibly adjust the thrust angle and power to competently adapt to the physical characteristics of the ship control in the path planning task and, thus, perform accurate planning in different scenarios.

### 3.3. Reward Function Design

The reward function should correlate highly with the state-space to make the agent learn the expected policy [16]. We divided the reward function into mainline and auxiliary reward, where the design of mainline reward  $R_{\text{dist}}$  mainly considers the distance between the ship and the destination, and the creation of auxiliary focuses on the effect of arrival point, boundary control, and obstacles on the traveling state. We set the initial distance between the vessel and the target generation position at initializing stage as  $\text{dist}_{\text{init}}$ . While the agent interacts with the environment, we record the distance between the ship and the endpoint as  $\text{dist}_{\text{rec}}$ . With the hull position changing, if the current distance  $\text{dist}_{\text{cur}}$  is less than  $\text{dist}_{\text{rec}}$ , a positive reward is given and rewritten  $\text{dist}_{\text{rec}}$  by  $\text{dist}_{\text{cur}}$ ; otherwise, it is returned as negative. The main line reward is as follows:

$$R_{\text{dist}} = \begin{cases} -\frac{\text{dist}_{\text{cur}}}{\text{dist}_{\text{init}}}, & \text{if } \text{dist}_{\text{cur}} \geq \text{dist}_{\text{rec}} \\ 1 - \frac{\text{dist}_{\text{cur}}}{\text{dist}_{\text{rec}}}, & \text{if } \text{dist}_{\text{cur}} < \text{dist}_{\text{rec}} \end{cases} \quad (7)$$

To prevent the agent from abnormal state-shifting in policy learning, which leads to failure to learn the expected policy, we solved this problem by creating a reward function with a potential-based reward shaping method [17], i.e., the repulsive field supported by obstacles and boundary control, and the gravitational field supported by arrival points.

The agent is given a decreasing penalty when it is close to the endpoint and an increasing penalty when it is close to the obstacles or boundary. This reward format can help the agent learn the robust correlation between state and reward, thus enabling the agent to accelerate the agent policy convergence while maintaining the invariance of the optimal policy.

The reward function is designed to remap the rendered image of  $n * n$  onto a plane of size  $m * m (n > m)$  and calculate the potential field value at each point on that plane. The specific details are as follows:

1. The repulsive field generation is calculated by the potential decline of the 2.5 times  $obsc_R$  obstacles range. The distance from the ship to the obstacle's center is  $dist_C$ . We design the reward function satisfy the factors of the potential; note that the bias calculated by  $obsc_R$  is  $\alpha$ . After the matrix is generated, the potential matrix generated for each obstacle is normalized and summed, as shown in Figure 3a and explained below:

$$R_{obsc} = \begin{cases} \ln[obsc_R * 0.05] - \alpha, & \frac{dist_C}{obsc_R} \in [0, 1) \\ \ln[dist_C - obsc_R * 0.95] - \alpha, & \frac{dist_C}{obsc_R} \in [1, 2.5) \end{cases} \quad (8)$$

$$R_{obsc} = 1 - \text{norm}(R_{obsc})$$

2. The arrival point-based gravitational field is generated by calculating the distance between different positions of the potential matrix and the arrival point, and calculating the gravitational potential decay based on the span. Setting the radius of the arrival point as  $reach_R$ , the distance between the current position and the arrival point is  $dist_C$ . The bias calculated by  $reach_R$  is  $\beta$ . Figure 3b shows the arrival point-based potential plane, as calculated below:

$$R_{reach} = \begin{cases} \beta - \ln[reach_R * 0.5], & dist_C \leq reach_R \\ \beta - \ln[dist_C - reach_R * 0.5], & dist_C > reach_R \end{cases} \quad (9)$$

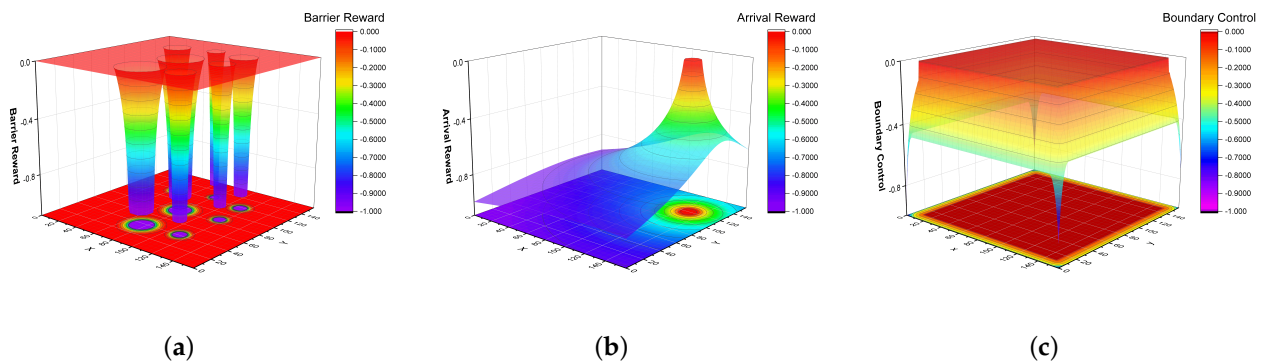
$$R_{reach} = 1 - \text{norm}(R_{reach})$$

3. The Repulsive field generation, based on boundary control, makes the agent consider the unknown of the boundary situation in path planning when the agent output action makes the ship move towards the boundary, gradually increasing the penalty. Thus, this ensures the route selected by the agent favors the known and clear center region and improves the planned route's security. If we set that the penalty boundary is  $bound_{limit}$ , the distance between the current position and the nearest edge is  $dist_{limit}$ , and the bias calculated by  $dist_{limit}$  is  $\gamma$ . Figure 3c shows the potential plane based on boundary control, calculated as follows:

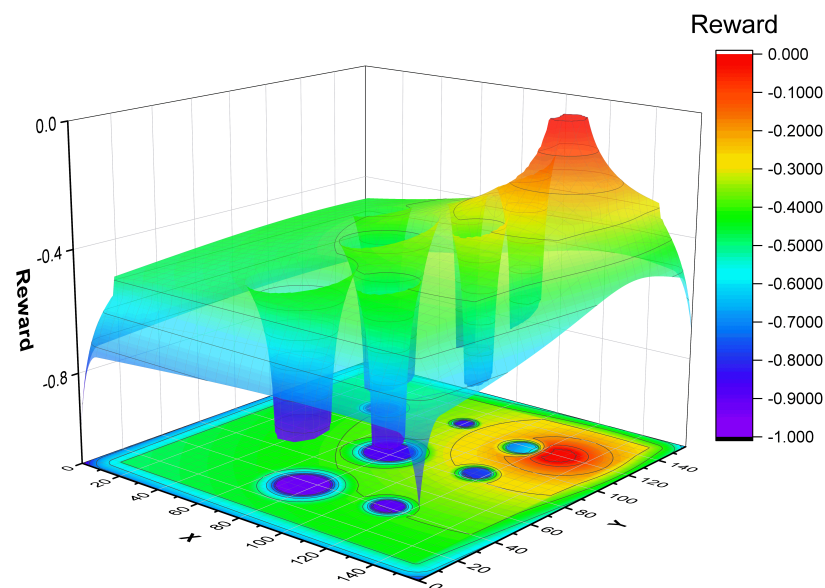
$$R_{bound} = \begin{cases} \frac{1}{-\ln[dist_{limit} + 1] + \gamma}, & dist_{limit} < bound_{limit} \\ 0, & otherwise \end{cases} \quad (10)$$

$$R_{bound} = \text{norm}(R_{bound}) - 1$$

From the above analysis, after obtaining the potential matrices with obstacles, arrival points, and boundary control, we can superimpose them and normalize them to obtain the complete potential-based reward function, as shown in Figure 4.



**Figure 3.** Potential-based reward: (a) Obstacle-based potential plane; (b) Arrival point-based potential plane; (c) Boundary control-based potential plane.



**Figure 4.** Potential plane based on obstacle, arrival point, and boundary control (superimposed).

#### 4. Beta Policy-Based Distributed Sampling PPO

When applying the policy gradient method for continuous domain control path planning, the DDPG [18] algorithm is usually used to solve complex tasks with high-dimensional data in the continuous domain. However, since DDPG is a deterministic policy, exploration noise needs to be added at the early stage of training to help the agent explore the environment. Suppose the agent cannot obtain enough mainline rewards during exploration. Under the influence of sparse rewards and deterministic policy, DDPG quickly converges to a lousy policy [19], which makes the actor enter a deadlock state, which cannot be jumped out of by increasing mainline rewards later.

DDPG suffers from long training time, low exploration efficiency, and unstable learning strategies in path planning tasks, leading to poor reproducibility and unsuitable path planning in complex environments.

##### 4.1. PPO Algorithm Applied to Global Path Planning Task and Optimization

As an on-policy algorithm, PPO uses the surrogate objective to optimize the policy, making it available to figure out discrete and continuous action space. PPO performs action sampling through the latest version of the stochastic policy, which helps the algorithm complete the environment exploration at the early training stage. With training progress, the policy gradually changes from high entropy to low entropy. Hence, the stochastic policy also benefits when facing complex stochastic environments.

### 4.1.1. PPO Algorithm Operation

The PPO algorithm is a variant of the policy gradient method, which typically calculates a policy gradient estimator and uses the stochastic gradient ascent to optimize. PPO uses the clipping of the surrogate function to reduce the problem of excessive deviation of the new policy from the old one:

$$L(s, a, \theta, \theta_{old}) = \min \left( \frac{\pi_{\theta}(a | s)}{\pi_{\theta_{old}}(a | s)} A_{\pi_{\theta_{old}}}(s, a), \text{clip} \left( \frac{\pi_{\theta}(a | s)}{\pi_{\theta_{old}}(a | s)}, 1 - \epsilon, 1 + \epsilon \right) A_{\pi_{\theta_{old}}}(s, a) \right) \quad (11)$$

Above,  $\pi_{\theta}$  and  $\pi_{\theta_{old}}$  denote the new and old policies, and the hyperparameter  $\epsilon$  is used to limit the difference between the two policies. Suppose, in state  $s$ ,  $A_{\pi}(s, a) > 0$ ; this indicates that the action-value brought by the selected act is greater than the average in the current state. In that case, the update towards the gradient ascent direction increases the selection of actions in such states.  $A_{\pi}(s, a) < 0$  indicates the action-value brought by the selected act is lower than the average in the current state, and the update occurs in the gradient descent direction to decrease the selection of actions in such states. With the Actor and Critic training, the Critic evaluates more accurately, and the policy network's distribution entropy decreases with the training to the convergence state.

### 4.1.2. Actor–Critic Network Structure

The Actor network uses the same feature extraction structure as the Critic network. However, the feature fusion differs due to the different output dimensions; the Actor and Critic networks vary slightly depending on the state-space.

The network for processing the data sequence state-space is a four-layer fully connected structure, and the hidden layer is [400, 300, 300]; the network for processing the image state-space adds feature extraction layers to extract information from image state, and the structure of the Actor in the image state-space is shown in Figure 5.

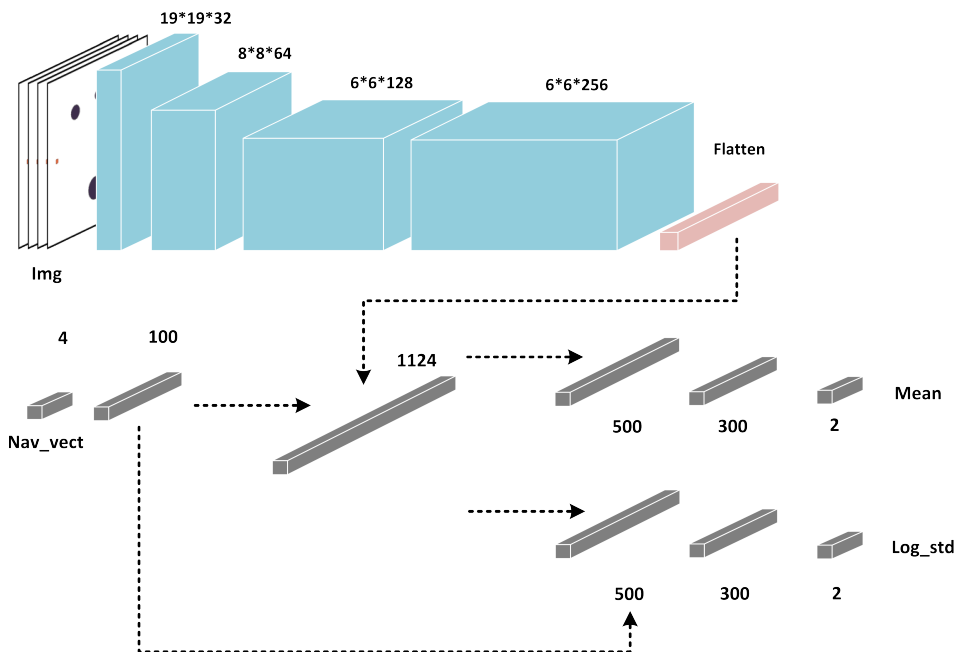


Figure 5. Actor network structure.

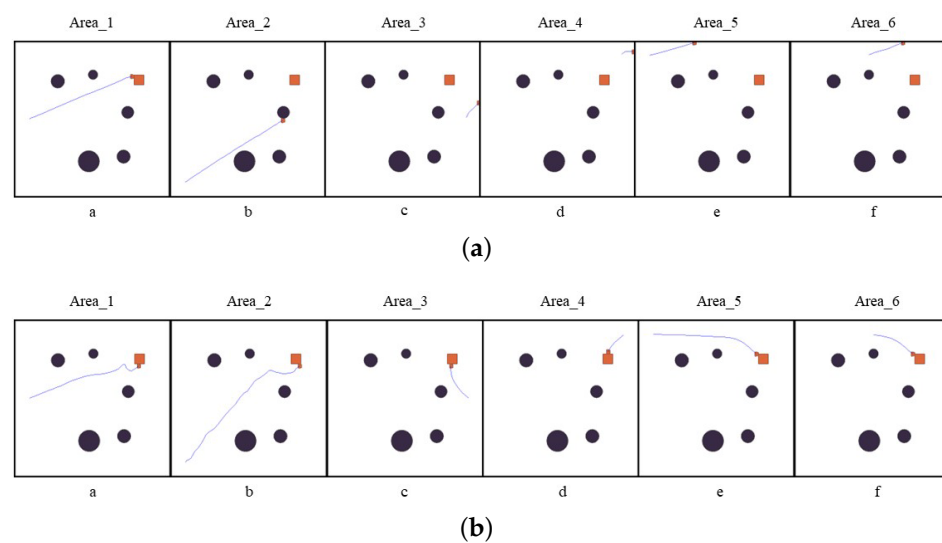
The image feature extraction structure consists of four convolutional layers. The first three layers use the same convolutional architecture of DQN when processing Atari images [9]. The fourth layer adds a convolutional layer with an output channel of 256, a kernel of 3x3, strides of 1, and a padding of 1. The extracted feature is concatenated with the current ship's navigation information and then fed into the fully connected network with

the hidden layer of [1124, 500, 300]. Finally, the mean and variance of Gaussian distribution are output.

Since the current model is used to solve the global path planning task, the model can be made to learn the spatio-temporal relationship of object motion on consecutive time frames through data frame superposition. To match the continuous data frame superposition structure used in the two state-space designs, we consider multi-frame superposition in the design of both model network structures.

#### 4.2. Problems of PPO Algorithm in Global Path Planning

When training with PPO for images, the different areas may lead to the environment not being fully explored, i.e., the generation of random positions can cause the agent to enter the termination state early due to collision with the object prematurely. This results in an extreme lack of samples, making it too difficult to train at some starting positions owing to the insufficient positive feedback, resulting in the agent learning only those policies for which data are easily obtained in the training process. Figure 6a shows the test results of six positions selected by equal spacing. From Figure 6a (subgraphs c and d), the agent can learn a better policy at the corresponding starting position in graph a due to the uneven sample acquisition. In contrast, the policies adopted by the agent at the related starting points in the b, c, e, d, and f graphs are similar to those acquired in graph a, indicating that the uneven sample acquisition during the agent's interaction with the environment affects agent policy learning.



**Figure 6.** Testing results: (a) PPO baseline planning test; (b) Distributed fixed starting position test.

#### 4.3. Distributed Sample Extraction PPO

According to the problems encountered in Section 4.2, a distributed sampling strategy based on region partitioning was designed, which separates execution from training; each executor explores from different regions and interacts with the environment [20].

The algorithm consists of  $N$  sub-processes interacting with the environment and collecting samples into their respective buffers. The process initializes the ship's position from its sub-regions (Figure 7) and interacts with the environment. After collecting  $T$  iter in the buffer, the worker calculates the advantage function and discount reward from the collected data and sends it to the main process in a package. When the main process completes training, the workers pull back the updated parameters from the global and use the new policy to interact.

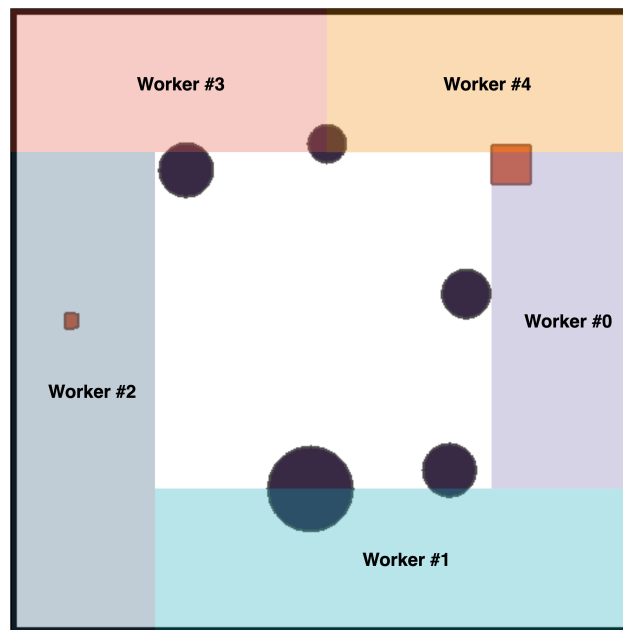


Figure 7. Refine sub-regions.

After the global network collects the data sent back from workers,  $N \cdot T$  samples are shuffled and cut into mini-batches for training. Compared with the A3C algorithm [21], which calculates the gradient by sub-process and uploads the grad to the global, the global network accumulates the grad and optimizes the policy. The newer distributed sample extraction PPO uses sub-processes to collect data and sends them to the main process for optimization. In this sampling method, data from different starting regions can be further integrated, sharing data among workers and increasing sample diversity. The experimental results in Section 5.1 show that this improved method significantly enhanced the robustness of the agent policy, which indicates that the distributed sampling policy based on region division is sufficient to make progress on complex exploration problems.

Figure 8 shows the distributed sampling policy based on region division. On the one hand, the number of samples at each initial position is balanced by region interaction. On the other hand, the distributed sampling policy obtains several times the data of the PPO algorithm over the same duration, thus reducing the convergence time. The robustness of the algorithm is significantly improved due to the rising number of samples. Figure 6b shows the test results. Compared with the PPO baseline (Figure 6a), the distributed sampling policy with area division can achieve better performance in the test of starting positions at equal intervals.

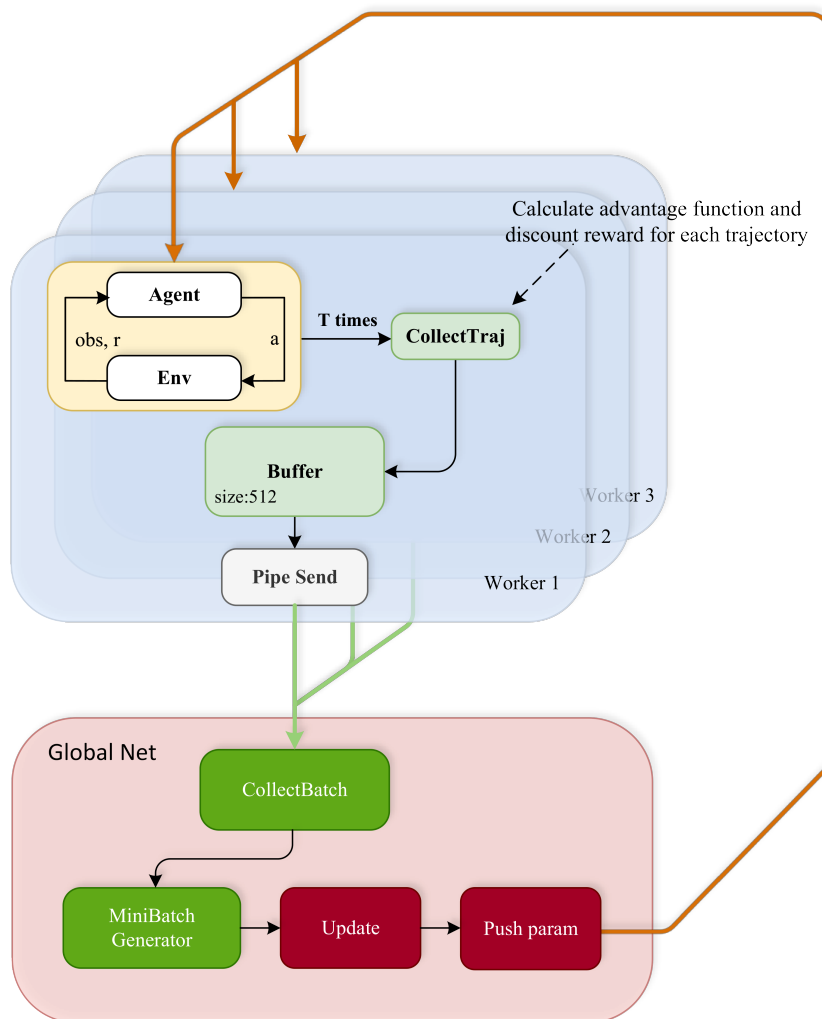


Figure 8. Distributed PPO algorithm.

#### 4.4. Improvement of Action Policy Based on Beta Distribution in Continuous Action Space

In stochastic policy continuous action control problems, the action sample is typically obtained from a Gaussian distribution  $N(\mu, \sigma^2)$  by predicting the mean and variance in the neural network and optimizing the parameters by computing the gradient of the policy relative to  $\mu$  and  $\sigma$ . However, the Gaussian distribution is an infinite support distribution. The mean and variance predicted by the neural network are limited by the action space when used in the Gaussian distribution, causing any action sampled from the distribution to be forcibly mapped back into the action space if it exceeds the action boundary. As shown in Figure 9, the Actor calculated the rudder angle according to the specified state and received the distribution parameter of  $\mu = 0.7$  and  $\sigma = 0.8$ , due to the rudder angle being limited to  $(-1, 1)$ . Still, the probability density  $\pi_\theta(a | s)$  corresponding to the action  $a$  at any position in the Gaussian distribution is greater than 0. To analyze this process, assume that the action space of rudder angle  $\mathcal{A} = [-1, 1]$ , with no clipping in the distribution sampling; then, clipping operates only in the environment. The gradient calculation uses the non-clipping action when computing the surrogate objective in PPO; the policy gradient estimator becomes  $g'_\theta = \nabla_\theta \log \pi_\theta(a | s) A^\pi(s, a')$ , where  $a'$  denotes the clipped action, and  $A^\pi(s, a')$  represents the advantage function generated in the clipped action. By calculating the gap between the gradient  $\mathbb{E}[g'] - \nabla J_\theta(\pi_\theta)$ , the expected policy gradient is as shown:

$$\mathbb{E}[g'_q] - \nabla_{\theta} J(\pi_{\theta}) = \int_S \rho_{\pi}(s) \left[ \int_{-\infty}^{-1} \pi_{\theta}(a | s) \nabla_{\theta} \log \pi_{\theta}(a | s) [Q(s, a') - Q(s, a)] da + \int_1^{\infty} \pi_{\theta}(a | s) \nabla_{\theta} \log \pi_{\theta}(a | s) [Q(s, a') - Q(s, a)] da \right] ds \quad (12)$$

It could be seen from the formula that the calculation is unbiased within the action space  $\mathcal{A}$ . This boundary effect caused by the Gaussian distribution produces a deviation in action sampling, which is passed to the calculation of the policy gradient, affecting the training speed.

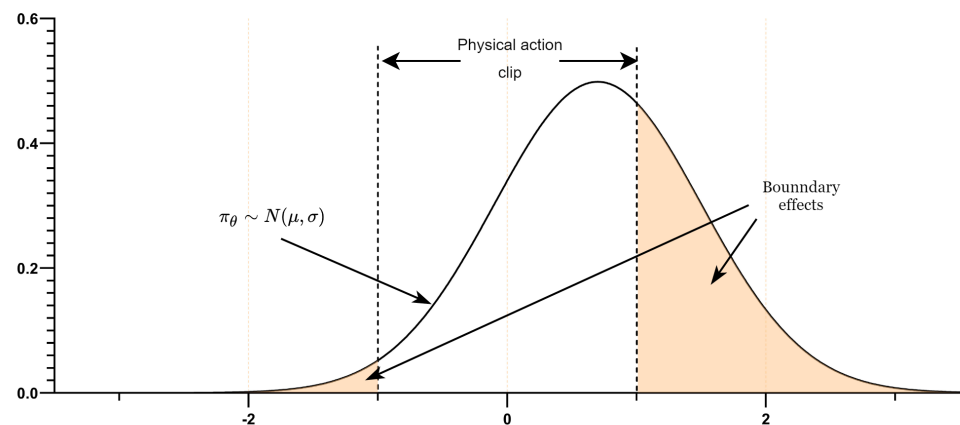


Figure 9. Gaussian boundary effects.

Chou et al. used the Beta distribution to replace the Gaussian distribution in a continuous control task and trained it faster than the infinite support distributions in the policy gradient algorithm (TRPO, ACER) [22]:

$$f(x; \alpha, \beta) = \frac{1}{B(\alpha, \beta)} x^{\alpha-1} (1-x)^{\beta-1} \quad (13)$$

$B(\alpha, \beta) = \frac{\Gamma(\alpha)\Gamma(\beta)}{\Gamma(\alpha+\beta)}$  represents the Beta function, which ensures that  $f(x; \alpha, \beta)$  integrates to 1 over the interval  $[0, 1]$ . The Beta distribution takes values in the range  $x \in [0, 1]$ , the shape of the distribution is determined by its shape parameters  $\alpha$  and  $\beta$ , and, as a finite-support distribution, it does not produce a Gaussian-like boundary effects. Therefore, it is unbiased in the gradient computation and converges faster than other infinite support distributions.

In this paper, to ensure that the action outputs conform to the different limits in the action space, it is necessary to remap the range of the Beta distribution. Therefore, the Beta policy is defined as  $\pi_{\theta}^{beta} = f\left(\frac{x+c}{2*c}; \alpha, \beta\right)$ . The value of parameter  $c$  is determined based on the action range in different dimensions of the action space, thereby avoiding the issue of sampling actions beyond the action boundaries.

The Beta distribution is generated by using policy  $\theta$  to estimate the parameters  $\alpha$  and  $\beta$ . Ensuring  $\alpha, \beta > 1$  makes the distribution a concave probability density curve in the interval with a single-peaked shape. In the experiments, when changing the sampling distribution to Beta with the same parameter settings, the training speed was faster than Gaussian, and the accumulated reward was higher than the Gaussian policy for the same training time.

**Algorithm 1** Beta policy-based distributed sampling PPO (*Worker*)

---

```

1: Set the training episode  $E$ , maxstep  $T$ 
2: Initialize the actor network  $\pi_\theta$ , and critic network  $V_\phi$  with the random parameters  $\theta, \phi$ 
3: Receive event flag from main processing
4: for  $e \in \{1, 2, \dots, E\}$  do
5:   According the divided sub-region random generate start point
6:   for  $i \in \{1, 2, \dots, T\}$  do
7:     Run  $\pi_\theta$  w.r.t Beta Policy, collect  $\{s_t, a_t, r_t\}$ 
8:     Caculate decayed Reward  $\hat{R}_t^{(i)} = \sum_{t=i}^{T-1} r_t * \gamma^{T-1-t} + V_\phi(s_t)$ 
9:     Caculate  $\delta_t^V = r_t + \gamma V_\phi(s_{t+1}) - V_\phi(s_t)$ 
10:    Caculate GAE  $\hat{A}_t^{(i)} = \sum_{l=0}^{i-1} (\gamma\lambda)^l \delta_{t+l}^V$ 
11:  end for
12:  Send  $T$  length trajectories buffer  $g = \{s_t, a_t, r_t, \hat{R}_t^{(i)}, \hat{A}_t^{(i)}\}$  to global through pipe
13:  if not event.is_set() then
14:    event.wait()
15:  end if
16:  Pull Global parameter to local
17: end for

```

---

**Algorithm 2** Beta policy-based distributed sampling PPO (*Global*)

---

```

1: Set the training episode  $E$ , Number of workers  $N$  and training iterations  $K$ 
2: Initialize the actor network  $\pi_\theta$ , and critic network  $V_\phi$  with the random parameters  $\theta, \phi$ 
3: Initialize event flag
4: for  $e \in \{1, 2, \dots, E\}$  do
5:   Initialize global buffer  $G$ 
6:   event.clear
7:   for  $p \in \{1, 2, \dots, N\}$  do
8:     Receive worker buffer  $g$  from pipe and collect to global buffer  $G$ 
9:   end for
10:  Sample shuffle, divided  $N$  worker samples to  $K$  iter batch,  $K * batch\_size < NT$ 
11:  for  $iter \in \{1, 2, \dots, K\}$  do
12:    Caculate  $\pi_\theta(a_t | s_t)$  using  $s_t$ 
13:     $J(\theta) = \sum_{t=1}^T \min(\frac{\pi_\theta(a_t | s_t)}{\pi_{\theta_{old}}(a_t | s_t)} A_t, clip(\frac{\pi_\theta(a_t | s_t)}{\pi_{\theta_{old}}(a_t | s_t)}, 1 - \epsilon, 1 + \epsilon) A_t)$ 
14:    Update Actor parameter  $\theta$  by caculate gradient w.r.t.  $J(\theta)$ 
15:     $L(\phi) = \sum_{t=1}^T [\hat{R}_t - V_\phi(s_t)]^2$ 
16:    Update Critic parameter  $\phi$  by caculate gradient w.r.t.  $L(\phi)$ 
17:  end for
18:  event.set
19: end for

```

---

**5. Experiments**

The environment building parameters are shown in Table 2, the training hyperparameters are set as shown in Table 3, and the training scenarios are shown in Table 4. We set

the training and testing stage frame size as  $n = 16$ , rendering mapping size  $s = 30$ , and rendering screen size as  $480 * 480$ . Testing found that rendering the potential field in the range of  $[80, 160]$  improved the algorithm performance when the mapping size increased. Still, when  $m$  was set above 160, the improvement of the algorithm effect was obviously weakened; thus, we set the potential field remapping parameter to  $m = 160$ .

**Table 2.** Experiment environment parameters.

Parameter		Value
Plane frame structure size	$n$	16
Render scale	$s$	30
Obstacle grid size	$k$	5
Obstacle generation ratio	$\tau$	0.6
Ship radius	$b$	0.36
Laser horizontal field angle	$\rho$	360
Laser effective reflection range	$h$	3.6
Laser beam	$l$	24
Reward function mapping size	$m$	160
$obsc_R$ based offset	$\alpha$	1.5
$reach_R$ based offset	$\beta$	6.0
$dist_L$ based offset	$\gamma$	3.0

**Table 3.** Hyperparameters.

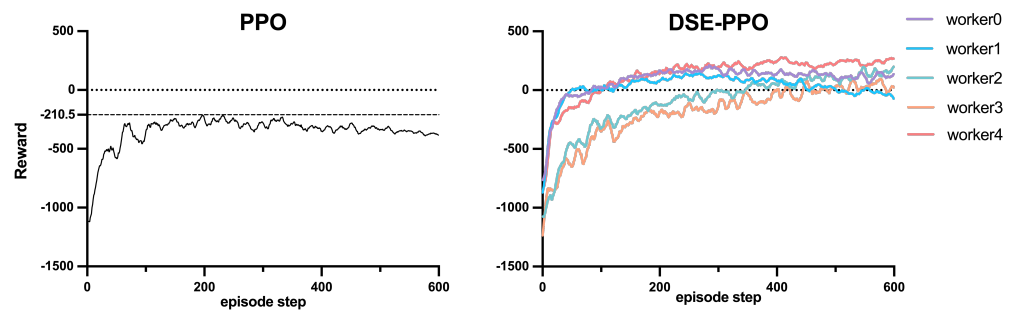
Parameter		Value
Actor training episode	$E_\pi$	16
Actor learning rate	$lr_\pi$	30
Actor learning rate decay	$StepLR_\pi$	5
Actor gradient clipping (L2)	$cn_\pi$	0.6
Critic training Episode	$E_v$	0.36
Critic training Episode	$lr_v$	360
Critic learning rate	$StepLR_v$	3.6
Critic learning rate decay	$gnorm_v$	24
Critic gradient clipping (L2)	$\gamma$	160
Discount factor	$\epsilon$	1.5
Clip ratio	$\beta$	6.0
Episode training iter	$train_{iter}$	3.0
GAE lambda	$\lambda$	6.0
Adapt KL target	$KL_{tg}$	6.0
Batch Size	$batch_{iter}$	6.0
Max Step	$max_{timestep}$	6.0

**Table 4.** Training set.

Parameter	Value
CPU	Intel Xeon E5 2666v (Boost 3.5 GHz)
Memory	Quad Channel Team DDR4-2133
Motherboard	Msi X99A SLI
Strong	SAMSUNG 840 EVO 1T
GPU	2*Gainward DuraPro GTX 1080 Ti Hurricane

### 5.1. PPO Baseline Compared with Distributed Sample Extraction PPO

To compare the performance of distributed sample extraction PPO with PPO baseline in the path planning task, we compared the accumulated rewards of the training phase, as shown in Figure 10.

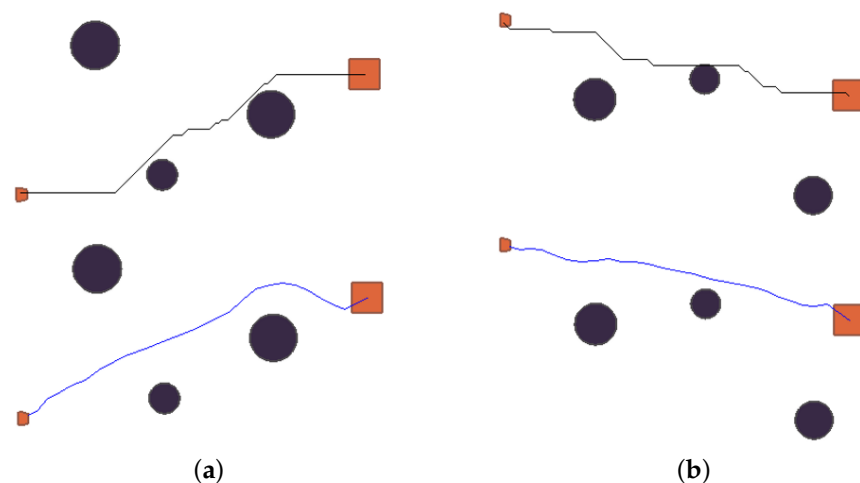


**Figure 10.** Accumulated rewards of PPO baseline compared with distributed sample extraction PPO.

Analyzing the training results, the accumulated reward of the PPO baseline converged to a value of  $-210.5$  at 230 episodes, but, as the training proceeded, the reward showed a dropping trend, which means that an attenuation occurred in policy, and this status could not improve, even with increased training. The distributed sample extraction PPO algorithm uses multiple workers to collect samples from different regions. On the one hand, this sampling method increases the number of samples; on the other hand, it ensures that the dataset samples are collected in each region in a balanced manner, hence increasing the stability of training. In the improved sampling policy, the accumulated reward from the several workers was significantly higher than that obtained by the random position generation based on the initial region in the PPO baseline.

### 5.2. Comparison of Reinforcement Learning and Traditional Path Planning Algorithms for Planning Paths

To compare the difference between RL and traditional path planning methods in the planned paths, we selected two scenarios. We set the same initial status in each system and used the A\* algorithm and RL method to compare. The scenario settings and the results of the planned paths of the two methods are shown in Figure 11.



**Figure 11.** Planning path comparison: (a) A\* and RL search path in scenario1; (b) A\* and RL search path in scenario2.

Figure 11a,b shows the paths searched by the A\* and RL in two different scenarios. The top figure shows the A\*-searched trajectory, and the bottom shows the RL-searched trajectory. As seen in Figure 11a,b, the trajectory that RL searched was smoother than A\*, and the planned paths did not have trouble with being too close to obstacles. Due to a large number of reefs near small islands, deceleration and detours are usually performed when passing through the reef area to ensure navigation safety. The paths of A\* did not take into

account the problem of the safety of navigation. In contrast, routes with economies and security could be planned with RL.

### 5.3. Beta Policy and Gaussian Policy

To compare the performance of the Beta and Gaussian policies in the path planning task, we trained the Beta and Gaussian policies in two different dimension state-spaces and compared the accumulated reward of each worker, as shown in Figure 12. Meanwhile, the path planning test was driven for both state-spaces, comparing the planned path length distribution and the routed success rate in different policies, as shown in Figure 13.

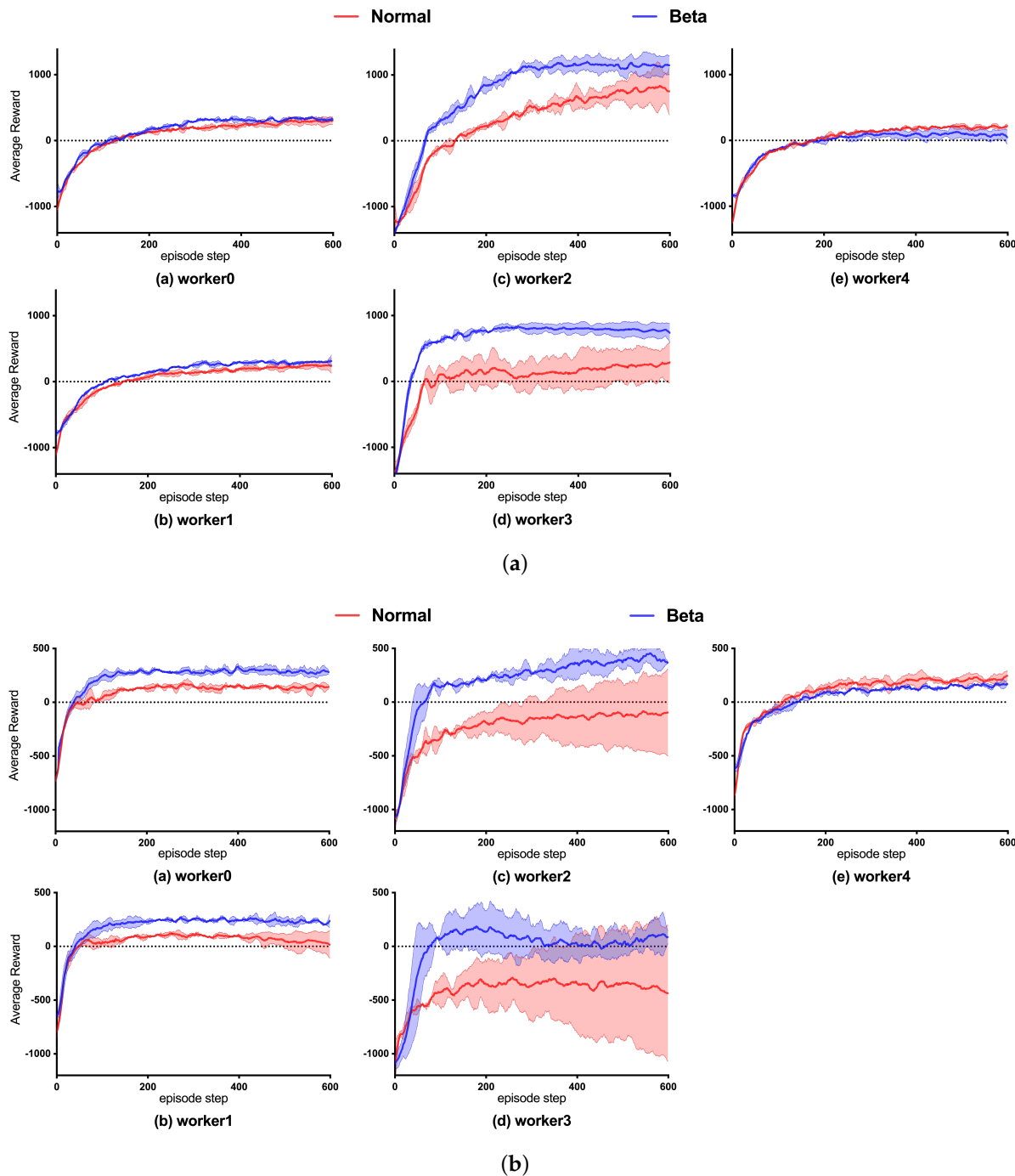
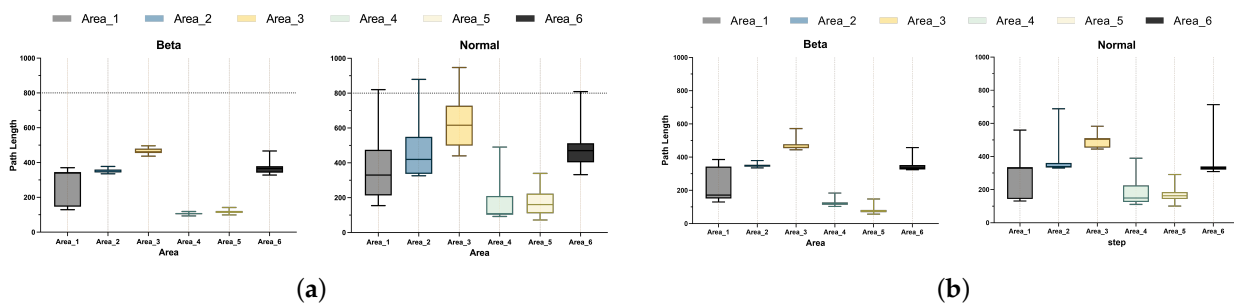


Figure 12. Sub-region accumulated rewards: (a) Data sequence state-space; (b) Image state-space.



**Figure 13.** Comparison of planning path length: (a) Data sequence state-space; (b) Image state-space.

- Comparison of accumulated reward during training:

The Beta and Gaussian policies were trained for five rounds. Figure 12 shows the reward with the corresponding mean curves and error intervals. In the worker0 and worker3 cases, the Beta policy had a higher reward than the Gaussian policy during the training, in terms of both data sequence and image. Meanwhile, the Gaussian policy in the worker1 and worker2 cases oscillated in the late training period, i.e., the issue of policy decline occurred. This occurred because the policy certainty of the Gaussian gradually increases during the training, with the gradual decrease in  $\sigma$  leading to the high variance in the Gaussian, which creates a heavy oscillation when it is close to 600 episodes. In worker3, the Gaussian policy based on the data sequence and image had an obvious decay after 100 iterations. Conversely, the Beta policy had a more obvious recovery after 400 episodes, and the reward of the Beta was always greater than the Gaussian at the same iteration. Since the area corresponding to worker4 was closer to the endpoint, Gaussian and Beta demonstrated less difference in accumulative rewards;

Comparing the reward convergence of these two state-spaces in training, it was found that the number of iterations required for the image to converge in training was significantly less than that of the data sequence. This is mainly because the image provides more comprehensive global information, enabling the agent to have more feature information to be considered in execution, thus planning a more comprehensive path because the global information is knowable. However, because the state-space based on the data sequence cannot obtain complete information in the environment and can only detect through radar data, the data sequence takes a long time to acquire samples from interacting, so the image converges faster than the data sequence. In summary, the Beta policy learned by each worker significantly improved compared to Gaussian, and, in terms of stability and robustness, Beta was also higher than the Gaussian policy;

- Comparison of planning path length:

Next, we took the Beta and Gaussian policy models with the same number of training iterations for the tests of data sequence and image. In testing, we used the same test position in Section 4, and each part was used for 200 iterations of path planning. We drew a box diagram of the planned length of each starting position according to the results, and compared the distance between the quartiles and the range of the extreme points by box plots to determine the stability of the planned path length. Figure 13 shows the test results. The Beta policy was significantly better than the Gaussian in the path planning stability in different state-spaces, and this advantage was shown in the case of testing at the same initial position, in which the median of Area1 was nearly 50% higher in the Beta policy (data sequence 171, image 180) than the Gaussian policy (data sequence 329, image 370). The differences between Q3 and Q1 of the Beta policy in Area2, Area4, and Area5 were lower than those of the Gaussian (more obviously in the data sequence case), and the distribution of planning path length was relatively concentrated. As well, the median of the data was significantly lower than that of the Gaussian strategy, indicating that the Beta strategy was more stable during the test;

- Planning shortest path length:

According to the shortest path length analysis based on the above data, as shown in Table 5 and Figure 13, the path length of Gaussian policy search in regions 1, 4, and 5 was 4.1%, 6.5%, and 43.9% higher than that of Beta, and the path length of Gaussian policy search in regions 2, 3, and 6 was not much different from that of Beta;

- Planning success rate:

As shown in Table 6, the planning success rates of Gaussian in regions 3, 4, and 5 were not as expected, but, in Beta, the success rate performance was higher in all areas, and the planning success rate was higher than 75% in each region.

**Table 5.** Shortest planning path.

Policy	Area1	Area2	Area3	Area4	Area5	Area6
Beta	<b>129.83</b>	<b>335.52</b>	<b>444.53</b>	<b>103.06</b>	<b>56.28</b>	<b>324.59</b>
Normal	135.50	338.99	444.58	110.27	100.39	327.88
Disparity	4.1%	1.0%	0.0%	6.5%	43.9%	1.0%

**Table 6.** Planning success rate.

Policy	Area1	Area2	Area3	Area4	Area5	Area6
Beta	<b>0.87</b>	<b>0.76</b>	<b>0.88</b>	<b>0.92</b>	<b>0.89</b>	<b>0.79</b>
Normal	0.85	0.76	0.31	0.36	0.15	0.71

#### 5.4. Comparison of Distributed Sampling PPO Based on Beta Policy and Traditional Path Planning Algorithm

In this model, reinforcement learning (RL) used continuous action space exploration for global path planning. It leveraged Box2D for continuous angle manipulation and simulated ship engine propulsion effects in the action space framework. This approach resulted in smoother trajectories compared to traditional algorithms.

To verify whether there were other aspects of performance loss with smoother tracks in RL, we created experiments to compare the performance of distributed extractor PPO based on a Beta policy with the traditional algorithms. We refactored the code of the heatmap obstacles to generate the rasterized map. We discretized a map to a size of 160\*160 and marked barriers, in particular, to generate the map, which corresponded to simulations. The traditional algorithms searched the track in a rastered map and mapped the nodes searched to the dimensions of the simulation to determine the distance. The comparison between reinforcement learning path planning and traditional path planning algorithms was mainly assessed in three aspects: planning time, the number of planning loop operations, and path length.

The time required for the algorithm to perform path planning in different areas (Table 7) and the number of planning loop operations (Table 8), as well as the planned path length (Table 9), were considered separately during the test. We compared the traditional algorithms IDA\*, Dijkstra, A\*, and bidirectional A\* with the reinforcement learning algorithm TD3 [23], as well as the optimized Gaussian policy PPO algorithm (Normal-PPO) and the Beta policy optimized PPO algorithm (Beta-PPO).

From Tables 7–9, Dijkstra took the longest time of the traditional algorithms and required the most iterations, but obtained the best solution for regions 3 and 4. IDA took significantly longer to track than other algorithms in regions 1, 3, and 5. However, its search paths were the shortest in each region, compared with different algorithms of the same type. The Bi-directional A\* took significantly shorter planning time than other algorithms in each area and had the lowest number of calls among traditional algorithms; however, Bi-directional A\* did not guarantee that the searched path was the best solution.

**Table 7.** Algorithm planning time comparison (in seconds).

Policy	Area1	Area2	Area3	Area4	Area5	Area6
IDA*	13.799	0.051	86.633	0.119	9.287	0.051
Dijkstra	1.371	5.015	5.302	0.79	0.237	3.111
A*	0.057	7.118	2.231	0.057	0.037	1.961
Bi-A*	0.083	0.509	0.975	0.058	0.048	0.182
TD3	0.046	0.102	0.134	0.038	0.024	0.095
Beta	0.038	0.074	0.093	0.029	0.029	0.071
Normal	<b>0.025</b>	<b>0.049</b>	<b>0.072</b>	<b>0.029</b>	<b>0.022</b>	<b>0.047</b>

**Table 8.** Comparison of planning operands (iterative loop operands).

Policy	Area1	Area2	Area3	Area4	Area5	Area6
IDA*	$1.02 \times 10^6$	385	$6.58 \times 10^6$	$9.14 \times 10^3$	$7.09 \times 10^5$	359
Dijkstra	$6.47 \times 10^3$	$1.96 \times 10^4$	$2.07 \times 10^4$	$4.12 \times 10^3$	$1.85 \times 10^3$	$1.44 \times 10^4$
A*	433	$7.82 \times 10^3$	$3.77 \times 10^3$	262	236	$2.31 \times 10^3$
Bi-A*	452	$1.85 \times 10^3$	$2.31 \times 10^3$	364	382	923
TD3	15	36	45	12	8	33
Beta	10	21	26	8	7	20
Normal	<b>6</b>	<b>12</b>	<b>17</b>	<b>7</b>	<b>6</b>	<b>13</b>

**Table 9.** Planning path length comparison (\*0.1 km).

Policy	Area1	Area2	Area3	Area4	Area5	Area6
IDA*	134.26	349.66	428.67	97.32	<b>55.32</b>	333.50
Dijkstra	137.27	349.66	<b>427.43</b>	<b>96.08</b>	58.32	333.51
A*	140.5	353.26	434.76	99.31	59.53	335.43
Bi-A*	141.76	355.39	440.11	105.05	61.42	333.15
TD3	159.29	369.10	467.07	132.75	87.45	351.83
Beta	<b>129.83</b>	<b>335.52</b>	444.53	103.06	56.28	<b>324.59</b>
Normal	135.50	338.99	444.58	110.27	100.39	327.88

During the comparison of reinforcement learning algorithms, the training occurred under the same hyperparameters, including interaction environment parameters, batch size, and number of iterations. The best policy model from the training process was then selected for testing. Upon analyzing the data in Tables 7–9, we observe that the TD3 algorithm performs markedly worse than other algorithms for planned path length in regions 1–6. Additionally, it requires the highest number of algorithm calls among the reinforcement learning algorithms. Based on this analysis, we can conclude that the TD3 algorithm is notably less effective than the optimized PPO algorithm in generating random initial regions.

In the comparison between the optimized Gaussian policy PPO algorithm and the Beta policy optimized PPO algorithm, the Gaussian policy requires fewer algorithm calls than the Beta policy, resulting in better planning time in testing. However, the experimental results show that the Beta policy has a better planning success rate and path length than the Gaussian policy. This is because the Beta policy employs more algorithm calls in the planning process, enabling finer maneuvers during path search that lead to shorter paths in global path planning. When comparing the differences between RL and traditional path planning algorithms in terms of planning time, number of algorithm calls, and path length, as shown in Tables 7–9, we see that RL considers the propulsion characteristics of the ship for continuous control in path planning. This approach yields smoother trajectories than traditional algorithms, and RL's planning task performance and path length are comparable to that of conventional algorithms.

## 6. Conclusions

This study aims to address the limitations of both traditional planning algorithms and reinforcement learning algorithms in the context of path planning tasks. Traditional planning algorithms require precise environment modeling, while most reinforcement learning algorithms can only perform path planning in discrete action spaces. To this end, this paper designs a generic environment framework for path planning tasks, builds the environment structure using the Box2D physics engine, and implements the environment design for continuous state-space and action space for reinforcement learning. On this basis, a deep reinforcement learning path planning, without precise modeling to enable dynamic decision-making, is realized by designing two state-spaces based on sensor data and image data.

In addition, this study solves the problems that reinforcement learning faces in path planning tasks, such as the unreasonable design of the reward function, using deterministic policies, and incomplete consideration of ship propulsion characteristics, as well as that traditional planning algorithms increase computation when facing scenario enlargement and other situations and cannot handle high-dimensional data. In this study, a reward function is designed to match the state-space, which considers the distance between the agent and the arrival point as the main part and incorporates the potential-based superposition of the arrival point, obstacles, and boundary control as the auxiliary parts. Moreover, the stochastic policy algorithm PPO is employed as a baseline for path planning. This enables the reinforcement learning algorithm to learn from the continuous angle control and engine propulsion features simulated by Box2D, effectively incorporating the ship's propulsion characteristics into the path planning process.

Building upon this, we propose a distributed sampling PPO algorithm based on the Beta policy. This algorithm addresses the issue of sample collection imbalance in global path planning tasks and effectively improves the performance and robustness of the algorithm. By resolving the action boundary problem associated with Gaussian policy, the proposed approach enhances the algorithm's performance in terms of accumulated reward and planning success rates across different sub-regions.

However, this study still has certain limitations. Specifically, the absence of testing in real marine environments during the experiments may lead to potential performance degradation in practical applications. To address this limitation, we are currently engaged in the deployment of the algorithm on hardware to assess the performance of the deep reinforcement learning global path planning algorithm in the context of sim2real. Furthermore, this study did not include a comparison with genetic algorithms for path planning. As another essential algorithm in path planning, genetic algorithms deserve investigation regarding their performance within the environment framework designed in this study. Thus, we will introduce genetic algorithms into the path planning environment designed in this study to evaluate their performance.

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

**Funding:** This research received no external funding.

**Institutional Review Board Statement:** Not applicable.

**Informed Consent Statement:** Not applicable.

**Data Availability Statement:** Not applicable.

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

## References

1. EMSA. Annual Overview of Marine Casualties and Incidents. 2019. Available online: <https://emsa.europa.eu/we-do/safety/accident-investigation/items.html?cid=141&id=3734> (accessed on 23 October 2022).
2. Felski, A.; Zwolak, K. The ocean-going autonomous ship—Challenges and threats. *J. Mar. Sci. Eng.* **2020**, *8*, 41. [CrossRef]
3. Chen, P.; Huang, Y.; Papadimitriou, E.; Mou, J.; van Gelder, P. Global path planning for autonomous ship: A hybrid approach of Fast Marching Square and velocity obstacles methods. *Ocean. Eng.* **2020**, *214*, 107793. [CrossRef]
4. Souissi, O.; Benatitallah, R.; Duvivier, D.; Artiba, A.; Belanger, N.; Feyzeau, P. Path planning: A 2013 survey. In Proceedings of the 2013 International Conference on Industrial Engineering and Systems Management (IESM), Rabat, Morocco, 28–30 October 2013; pp. 1–8.
5. Sun, L. A real-time collision-free path planning of a rust removal robot using an improved neural network. *J. Shanghai Jiaotong Univ. (Sci.)* **2017**, *22*, 633–640. [CrossRef]
6. Sheng, H.; Guo, C.; Li, T.; Yu, Y. Study navigation method for unmanned ship intelligent avoidance. *Harbin Eng. Univ. Acad. Rep.* **2018**, *39*, 998–1005.
7. Wang, C.; Zhang, X.; Cong, L.; Li, J.; Zhang, J. Research on intelligent collision avoidance decision-making of unmanned ship in unknown environments. *Evol. Syst.* **2019**, *10*, 649–658. [CrossRef]
8. Kuczkowski, Ł.; Śmierczalski, R. Path planning algorithm for ship collisions avoidance in environment with changing strategy of dynamic obstacles. In Proceedings of the Polish Control Conference, Krakow, Poland, 18–21 June 2017; Springer: Berlin/Heidelberg, Germany, 2017; pp. 641–650.
9. Mnih, V.; Kavukcuoglu, K.; Silver, D.; Graves, A.; Antonoglou, I.; Wierstra, D.; Riedmiller, M. Playing atari with deep reinforcement learning. *arXiv* **2013**, arXiv:1312.5602.
10. Van Hasselt, H.; Guez, A.; Silver, D. Deep reinforcement learning with double q-learning. In Proceedings of the AAAI Conference on Artificial Intelligence, Phoenix, AZ, USA, 12–17 February 2016; Volume 30.
11. Sutton, R.S.; McAllester, D.; Singh, S.; Mansour, Y. Policy gradient methods for reinforcement learning with function approximation. *Adv. Neural Inf. Process. Syst.* **1999**, *12*, 1057–1063.
12. Schulman, J.; Wolski, F.; Dhariwal, P.; Radford, A.; Klimov, O. Proximal policy optimization algorithms. *arXiv* **2017**, arXiv:1707.06347.
13. Schulman, J.; Levine, S.; Abbeel, P.; Jordan, M.; Moritz, P. Trust region policy optimization. In Proceedings of the International Conference on Machine Learning, PMLR, Lille, France, 7–9 July 2015; pp. 1889–1897.
14. Schulman, J.; Moritz, P.; Levine, S.; Jordan, M.; Abbeel, P. High-dimensional continuous control using generalized advantage estimation. *arXiv* **2015**, arXiv:1506.02438.
15. Engstrom, L.; Ilyas, A.; Santurkar, S.; Tsipras, D.; Janoos, F.; Rudolph, L.; Madry, A. Implementation matters in deep policy gradients: A case study on PPO and TRPO. *arXiv* **2020**, arXiv:2005.12729.
16. Ng, A.Y.; Harada, D.; Russell, S. Policy invariance under reward transformations: Theory and application to reward shaping. In Proceedings of the Sixteenth International Conference on Machine Learning (ICML), Bled, Slovenia, 27–30 June 1999; Volume 99, pp. 278–287.
17. Dong, P.; Zhang, Z.; Mei, X.; Zhu, S. Reinforcement learning path planning algorithm introducing potential field and trap search. *Comput. Eng. Appl.* **2018**, *1*, 129–134.
18. Lillicrap, T.P.; Hunt, J.J.; Pritzel, A.; Heess, N.; Erez, T.; Tassa, Y.; Silver, D.; Wierstra, D. Continuous control with deep reinforcement learning. *arXiv* **2015**, arXiv:1509.02971.
19. Matheron, G.; Perrin, N.; Sigaud, O. The problem with DDPG: Understanding failures in deterministic environments with sparse rewards. *arXiv* **2019**, arXiv:1911.11679.
20. Heess, N.; TB, D.; Sriram, S.; Lemmon, J.; Merel, J.; Wayne, G.; Tassa, Y.; Erez, T.; Wang, Z.; Eslami, S.; et al. Emergence of locomotion behaviours in rich environments. *arXiv* **2017**, arXiv:1707.02286.
21. Mnih, V.; Badia, A.P.; Mirza, M.; Graves, A.; Lillicrap, T.; Harley, T.; Silver, D.; Kavukcuoglu, K. Asynchronous methods for deep reinforcement learning. In Proceedings of the International Conference on Machine Learning, PMLR, New York, NY, USA, 20–22 June 2016; pp. 1928–1937.
22. Chou, P.W.; Maturana, D.; Scherer, S. Improving stochastic policy gradients in continuous control with deep reinforcement learning using the beta distribution. In Proceedings of the International Conference on Machine Learning, PMLR, Sydney, Australia, 6–11 August 2017; pp. 834–843.
23. Fujimoto, S.; Hoof, H.; Meger, D. Addressing function approximation error in actor-critic methods. In Proceedings of the International Conference on Machine Learning, PMLR, Stockholm, Sweden, 10–15 July 2018; pp. 1587–1596.

**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.