electronics

MDPI

Article

Unmanned Ground Vehicle Path Planning Based on Improved DRL Algorithm

Lisang Liu [1,2,*], Jionghui Chen [1,2], Youyuan Zhang [1,2], Jiayu Chen [1,2], Jingrun Liang [1,2] and Dongwei He [1,2]

[1] School of Electronic, Electrical Engineer and Physics, Fujian University of Technology, Fuzhou 350118, China; 2221905020@smail.fjut.edu.cn (J.C.); 2221905038@smail.fjut.edu.cn (Y.Z.); 2221905010@smail.fjut.edu.cn (J.C.); liangjingrun@smail.fjut.edu.cn (J.L.); he_dw@fjut.edu.cn (D.H.)

[2] Fujian Province Industrial Integrated Automation Industry Technology Development Base, Fuzhou 350118, China

* Correspondence: liulisang@fjut.edu.cn

Abstract: Path planning and obstacle avoidance are fundamental problems in unmanned ground vehicle path planning. Aiming at the limitations of Deep Reinforcement Learning (DRL) algorithms in unmanned ground vehicle path planning, such as low sampling rate, insufficient exploration, and unstable training, this paper proposes an improved algorithm called Dual Priority Experience and Ornstein–Uhlenbeck Soft Actor-Critic (DPEOU-SAC) based on Ornstein–Uhlenbeck (OU noise) and double-factor prioritized sampling experience replay (DPE) with the introduction of expert experience, which is used to help the agent achieve faster and better path planning and obstacle avoidance. Firstly, OU noise enhances the agent's action selection quality through temporal correlation, thereby improving the agent's detection performance in complex unknown environments. Meanwhile, the experience replay is based on double-factor preferential sampling, which has better sample continuity and sample utilization. Then, the introduced expert experience can help the agent to find the optimal path with faster training speed and avoid falling into a local optimum, thus achieving stable training. Finally, the proposed DPEOU-SAC algorithm is tested against other deep reinforcement learning algorithms in four different simulation environments. The experimental results show that the convergence speed of DPEOU-SAC is 88.99% higher than the traditional SAC algorithm, and the shortest path length of DPEOU-SAC is 27.24, which is shorter than that of SAC.

Keywords: path planning; soft actor-critic (SAC); deep reinforcement learning (DRL); double-factor prioritized sampling experience replay (DPE); expert experience

Citation: Liu, L.; Chen, J.; Zhang, Y.; Chen, J.; Liang, J.; He, D. Unmanned Ground Vehicle Path Planning Based on Improved DRL Algorithm. *Electronics* **2024**, *13*, 2479. https://doi.org/10.3390/electronics13132479

Academic Editor: Arturo de la Escalera Hueso

Received: 24 May 2024
Revised: 11 June 2024
Accepted: 21 June 2024
Published: 25 June 2024

1. Introduction

As a large-scale comprehensive multifunctional system, the study of unmanned ground vehicle path planning involves multidisciplinary comprehensive knowledge [1]. It is an artificial intelligence discipline integrating environment perception, dynamic decision making, and planning. In the current context of rapid development of artificial intelligence and robotics, unmanned ground vehicles are gradually developing from the areas of national defense to industry and agriculture as well as the service industry [2], for example, logistics and transportation, disaster emergency treatment and rescue [3], household intelligent services [4], medical support, etc., and its application prospects are very broad. Path planning is a classic and important research content in mobile robot applications, which mainly refers to the intelligent sensors carried by the robot to fully perceive the surrounding environment, and use the obtained information as the basis for the path planning from the starting point to the termination point, and ensure that there is no collision requirement [5]. Due to the complexity and diversity of unmanned ground vehicle application environments, how to teach unmanned ground vehicles in different environments strategies to avoid static and dynamic obstacles, and plan a successful path to reach the target location has become a research hotspot to improve the robot's autonomous navigation planning

capability [6]. How to perform path planning with unknown environmental information becomes a major problem.

Li Q [7] optimized Deep Q learning (DQN). It was proposed to reduce the states with a high probability of occurrence by optimizing the calculation method of the Q value, so as to improve the robot's ability to explore the environment. However, it was only tested among static environments and did not consider the case of dynamic obstacles. Wang Chong [8] proposed a heuristic augmented learning dynamic environment planning algorithm combining self-attention and a long-term short-term memory network, which ensured maximum information sources while the global best path was used to guide local path planning. However, it increased the complexity of the neural network and increased the amount of computation, resulting in longer training time. L. Yang [9] used dynamic state normalization and prioritization experience replay techniques to help robots quickly avoid obstacles and reach their goals. P. Chen [10] addressed the low sample utilization problem caused by random sampling by proposing the use of prioritized experience replay (PER) to vary the weight of the sample and then improve the sampling efficiency. Y. Zhang [11] proposed the Soft Actor-Critic Long Short-Term Memory (SAC-LSTM) algorithm to address the lack of the SAC algorithm's ability to sustain exploration of the agent in complex situations. Che Wang [12] proposed Emphasizing Recent Experience (ERE), a simple but powerful off-policy sampling technique, which emphasized recently observed data, while not forgetting the past, and showed that SAC + PER can marginally improve the sample efficiency performance of SAC, but much less so than SAC + ERE. Maria [13] targeted improving the performance of an agent's exploration strategy during reinforcement learning training, using random perturbations generated by the Ornstein–Uhlenbeck (OU noise) process, which created a time-dependent noise. It was found that the agent could explore the environment more during training, improving the performance of the agent.

In 2020, Josef [14] addressed how to improve UGV local planning under unknown rugged terrain incorporating self-attention modules into deep reinforcement learning architectures to increase the interpretability of learning strategies. The method provides a large performance improvement over potential field or local motion planning search space methods. However, the effect of moving obstacles was not considered. Liu [15] proposed a model based on the combination of DQN and CNN in order to reduce the cost of human and material resources and improve the efficiency of power line inspection by introducing UGV into circuit inspection. However, there is still a gap between its use of discrete action space algorithms and the continuous action space of the real environment. Chen [16] proposed a search-based optimizing DRL algorithm (SO-DRL) and applied it to the path planning of real UGVs, discovering that the algorithm can guide UAVs or UGVs to navigate without adjusting any network.

This paper proposes a path-planning algorithm based on an improved Soft Actor-Critic (SAC). The proposed algorithm improves the convergence speed of the agent, reduces the training time, and finds the optimal path. Firstly, to address the underutilization of sampling, the random sampling experience pool is turned into a priority sampling experience replay buffer based on dual factors. Then, for the lack of exploration, OU noise is added to the agent's action selection to increase the agent's exploration performance. Finally, for unstable training, expert experience is introduced to help the agent jump out of the local optimal solution when it falls into the local optimal solution so as to reach the global optimal solution and train stably. In this paper, the path planning simulation based on DPEOU-SAC is compared and analyzed with other algorithms, and convergence rounds, path lengths, cumulative rewards, and other metrics are used to evaluate the speed of convergence and the quality of the planned paths.

Main contributions of this work: (1) Applying the SAC algorithm to path planning behavior in continuous action space. (2) Adding OU noise enhances agent exploration performance. (3) Innovations in experience replay buffer to address low sample utilization and sampling inefficiencies. (4) Bringing in expert experience to help the agent achieve faster and better path planning. (5) Establishing simulation experiments to prove the feasibility.

This paper is organized as follows: Section 2 presents a background of path planning in unmanned ground vehicles, unmanned ground vehicle path planning in reinforcement learning, and SAC. The optimization of the SAC algorithm (DPEOU-SAC) is described in Section 3. Section 4 shows the experiments of the simulation and analyzes the results. Finally, a conclusion is drawn in Section 5.

2. Background

2.1. Path Planning in Unmanned Ground Vehicles

Path planning in unmanned ground vehicles is an important part of robot navigation. Path planning is the process by which an unmanned ground vehicle receives information about its surroundings through its own sensors and automatically generates a collision-free and optimal path [17]. At present, scholars have achieved considerable results in the research of the problems in the field of path planning, and have proposed a variety of path planning methods: graph search method, fast extended random tree method, artificial potential field method, fuzzy logic algorithm, and neural network algorithm. Path planning is categorized into global and local path planning. The key distinction lies in the availability of environmental information: global path planning assumes complete knowledge of the environment including all obstacles, whereas local path planning operates under conditions where some obstacles may be unknown. Gao [18] proposed to realize a global environment known as path planning by Q-learning, and the planned paths are better than Breadth-First Search (BFS) and Rapidly exploring Random Tree (RRT) algorithms. Xu [19] proposed an improved A* algorithm to design a sliding track angle adjustment scheme to achieve the degree of smoothing of paths where the global environment is known. Zong [20] proposed the Regional-Sampling RRT (RS-RRT) algorithm to accomplish local path planning, which integrates Gaussian distribution sampling and local bias sampling to improve the search efficiency in the sampling phase. Szczepanski [21] used the artificial potential field method for local path planning, the algorithm addresses the problem of local minima in the artificial potential field method, for which a new artificial potential field supported by augmented reality technology is proposed to bypass the upcoming local minima.

In path planning for Unmanned Ground Vehicles (UGVs), the goal is to plan a smooth and collision-free route from a given start point to the endpoint in a limited planar range. A schematic of the scenario is shown in Figure 1. The starting point of each of the above four maps is denoted by $\times$, and the ending point is denoted by $\bullet$. The red circles and rectangles represent static obstacles. agent has to avoid the obstacles to reach the end point from the starting point when performing the path planning. In order to be closer to reality, the following conditions are assumed in this paper:

1. Environment sensing: UGV acquires environment information through LiDAR.
2. Information exploration: the UGV only knows the starting point and the endpoint in the path planning process, and all other information is obtained through exploration.

Considering the real-time requirement of UGV, we need to reduce the amount of computation. Therefore, the following simplifying assumptions are made for the UGV model.

1. The side deviation angle of the center of mass remains unchanged during the tracking path of the UGV.
2. The steering angles of the inner and outer wheels are the same.
3. Neglecting the side-to-side and pitch motions of the UGV.
4. The suspension is assumed to be a rigid body and the vertical motion is ignored.

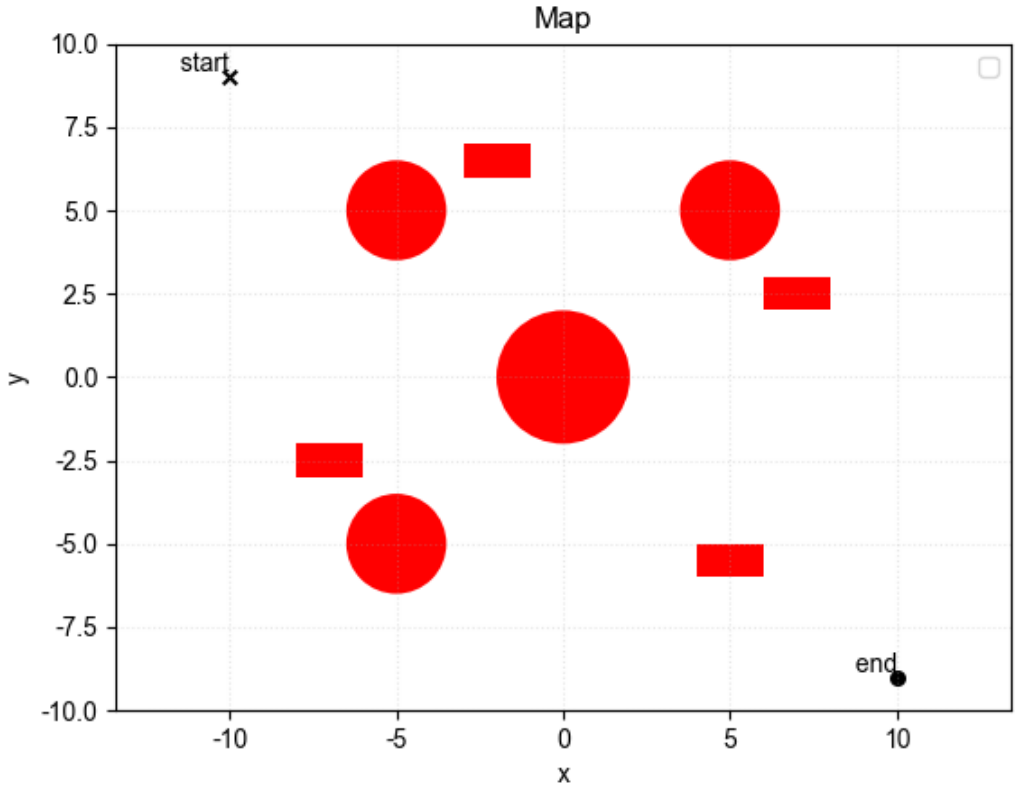


Figure 1. Schematic diagram of UGV path planning.

In this paper, a kinematic model of UGV based on continuous action space is adopted. The state of UGV is represented as position and direction (x, y, θ), and the action is represented as linear and angular velocity (v, ω). The state transfer equations of UGV are as follows:

$$\begin{cases} \dot{x} = v\cos(\theta) \\ \dot{y} = v\sin(\theta) \\ \dot{\theta} = \omega \end{cases} \tag{1}$$

In order to be closer to practical applications, the motion of the UGV is subject to the following limitations:

1. Speed limitation:

$$\begin{cases} v_{\min} \le v \le v_{\max} \\ \omega_{\min} \le \omega \le \omega_{\max} \end{cases} \tag{2}$$

$v_{\min}$ and $v_{\max}$: these two variables represent the minimum and maximum linear speed of the UGV, respectively. This means that the forward speed of UGV at any moment cannot be lower than $v_{\min}$ and cannot be higher than $v_{\max}$. $\omega_{\min}$ and $\omega_{\max}$: these two variables represent the minimum and maximum angular velocity of the UGV, respectively. This means that the steering speed of the UGV at any moment cannot be lower than $\omega_{\min}$ and cannot be faster than $\omega_{\max}$.

2. Acceleration limit:

$$\begin{cases} a_{\min} \le a \le a_{\max} \\ \alpha_{\min} \le \alpha \le \alpha_{\max} \end{cases} \tag{3}$$

$a_{\min}$ and $a_{\max}$: these two variables represent the minimum and maximum linear acceleration of the UGV, respectively. These limits ensure that the UGV does not exceed these preset limits when accelerating or decelerating, helping to avoid instability or loss of control due to too rapid acceleration or deceleration.

$\alpha_{\min}$ and $\alpha_{\max}$: These two variables represent the minimum and maximum angular acceleration of the UGV, respectively. These limits control the acceleration of the UGV steering and help to ensure smooth and safe steering maneuvers.

3. Steering angle limitation:

$$\left|\dot{\theta}\right| \leq \theta_{\max} \tag{4}$$

$\theta_{\max}$: this variable represents the maximum rate of change in steering angle of the UGV. It limits the maximum change in steering angle of the UGV per unit of time, which helps to prevent the UGV from turning sharply in a short period of time, thus increasing the stability and safety of driving.

With the above state transfer equations and motion restrictions, UGV can perform path planning in continuous state and action spaces. In this paper, a deep reinforcement learning algorithm is used, and the UGV learns to select the optimal action in different states by interacting with the environment, so as to realize smooth and collision-free path planning from the start point to the endpoint.

In this paper, we consider path planning for UGVs in a 2D environment in order to simplify the computational process, combined with the kinematic model, the vehicle and the target can be considered as mass points. Figure 2 shows all the possible maneuvers of the vehicle.

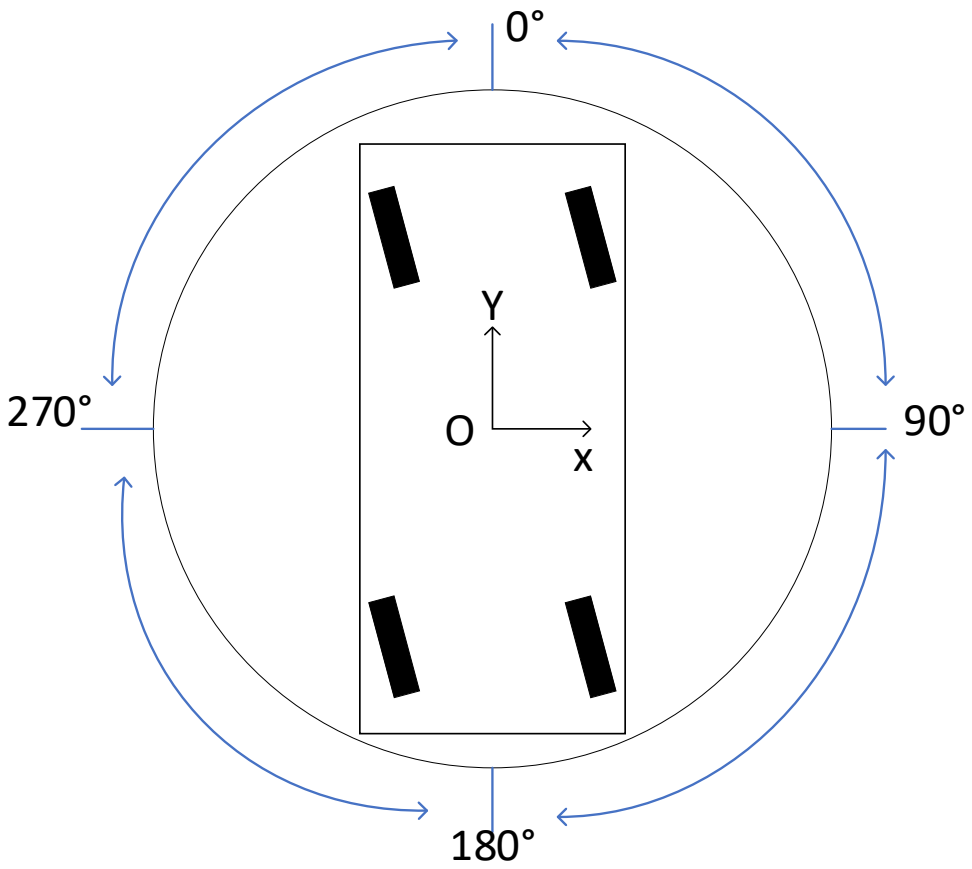


Figure 2. All possible actions of the UGV.

2.2. Unmanned Ground Vehicle Path Planning in Reinforcement Learning

Reinforcement learning is a method in which an agent takes an action to continuously interact with the environment, acquires information about the interaction, and uses the acquired information to guide the action of the agent. Existing reinforcement learning (RL) algorithms are categorized according to the learning objectives: Policy-Based, Value-Based,

and Actor-Critic [22]. The Policy-Based reinforcement learning algorithm is a direct output of the probability of the next action, based on which the action is selected. However, it does not necessarily select the action with the highest probability, but selects the action based on the strategy [23]. It is applicable to both discontinuous and continuous actions. A Value-Based reinforcement learning algorithm outputs the value of each action and selects the action with the highest value. This approach is suitable for environments with discrete, non-continuous actions. The Actor-Critic reinforcement learning algorithm combines the advantages of Policy-Based and Value-Based. The Actor network makes the action based on the probability, and the Critic network gives the value based on the action, thus speeding up the learning process. The core of the RL algorithm is the Markov Decision Process (MDP), which is introduced by ref. [24]:

State S_t represents the specific situation of the system at a given moment in time.

Action A_t represents that the agent can take in S_t.

Reward R_t indicates the immediate feedback the agent receives after taking A_t in S_t.

P is the state transfer probability that the state at moment t (S_t) is transferred to the state at moment t + 1 (S_{t+1}).

$\gamma \in [0, 1]$ is the discount factor that describes how the agent considers the future reward.

Policy ($\pi(a|s)$) is the policy that the agent takes to estimate the next action based on the current state.

$V_\pi(s)$ is based on the state value function when the $\pi(a|s)$, which represents the expectation of the gain obtained when following the $\pi(a|s)$.

$Q_\pi(s, a)$ is the action value function, which represents the expectation of the harvest that can be obtained by acting A_t in state S_t.

In RL, $\pi(a|s)$ denotes the probability distribution of an agent choosing an action from the set of actions in the current state. $V_\pi(s)$ and $Q_\pi(s, a)$ denotes the Markov Decision Process (MDP) combined with Bellman's equation.

$$\pi(a|s) = P[A_t = a|S_t = s] \tag{5}$$

$$V_\pi(s) = E[R_{t+1} + \gamma V_\pi(s')\big|S_t = s] \tag{6}$$

$$Q_\pi(s, a) = E[R_{t+1} + Q_\pi(s', a')\big|S_t = s, A_t = a] \tag{7}$$

The aim of unmanned ground vehicle path planning is to generate a sequence of waypoints as targets for autonomous navigation. This involves a sequential decision-making process, which therefore exhibits MDP [25].

Figure 3 clearly shows how Reinforcement Learning (RL) can be applied to the path planning of an Unmanned Ground Vehicle (UGV). State S_t represents the current position of the UGV. Action A_t represents the direction of the UGV's movement and the step size. Reward R_t represents the instantaneous feedback that the UGV receives after it takes A_t in S_t. P represents the probability that the UGV will move to S_{t+1} after it has taken A_t in S_t. In a path planning problem, P is usually deterministic, and the UGV will transfer deterministically to S_{t+1} by taking A_t in S_t. The UGV makes trial and error in the environment and chooses the best action based on the feedback from the environment and the current state. During training, the vehicle learns by trial and error and continuously optimizes its strategy and value function.

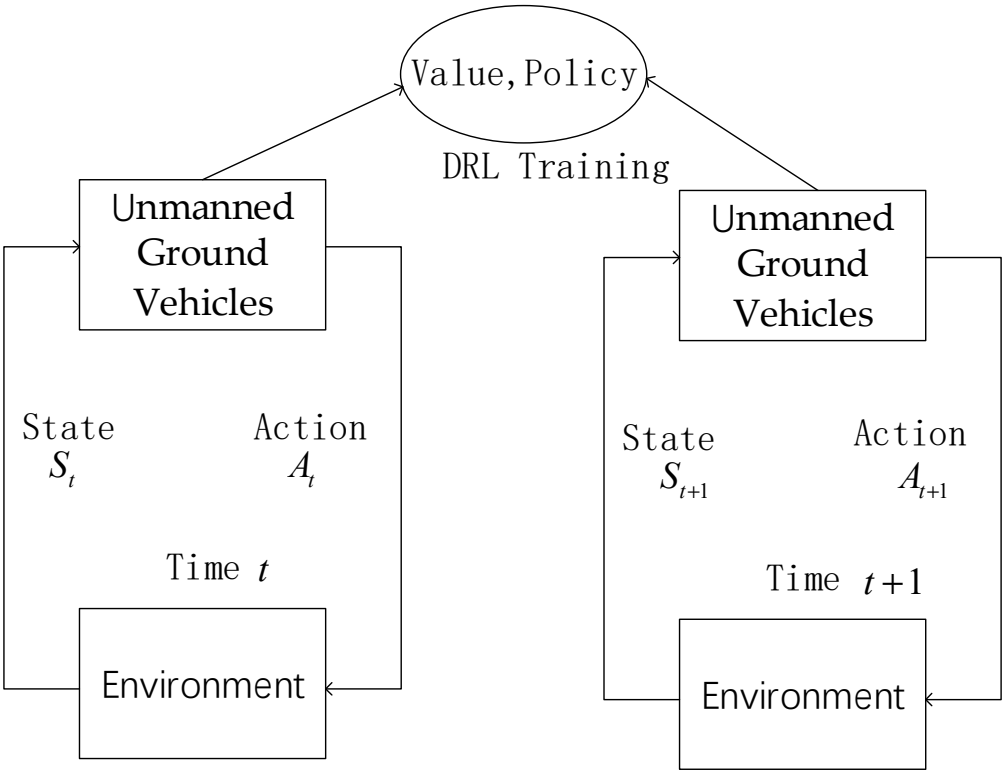


Figure 3. Unmanned ground vehicle path planning in RL framework.

2.3. SAC

The SAC algorithm combines the advantages of the Actor-Critic algorithm, maximum entropy model, and offline strategies. It is based on Deep Deterministic Policy Gradient (DDPG), uses randomized strategies, and introduces maximum strategy entropy to learn the strategies. The schematic diagram of the SAC algorithm is shown in Figure 4.

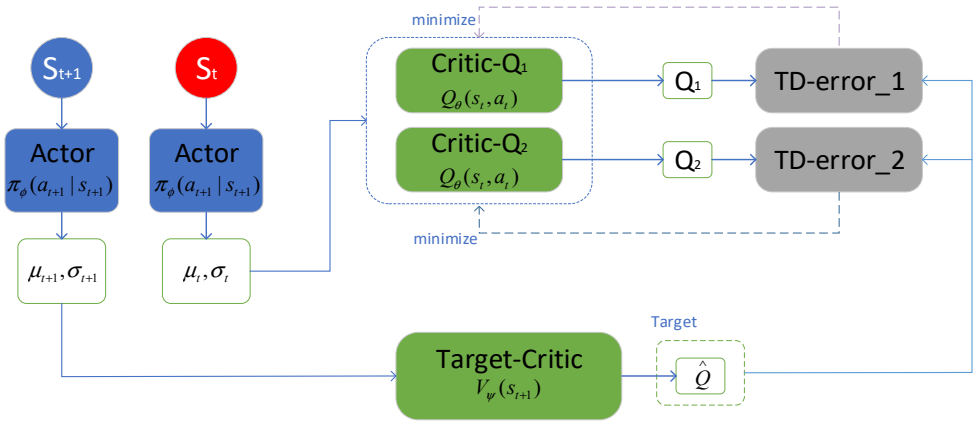


Figure 4. Schematic diagram of SAC algorithm.

The SAC algorithm has one Actor network, two Critic networks, and two Critic target networks: the Actor network receives the input states and outputs the mean and variance of the spatial probability distribution of the actions, the Critic network is used for evaluating the actions taken by the robot in the current state, and the Critic target network is a copy of the Critic network but with slower parameter updates to prevent oscillations and instabilities in the Q-value estimation process [26]. The Critic target network is a copy of the Critic network, but its parameters are updated more slowly, which prevents oscillations and instability in the Q-value estimation process. The Critic target network

is updated using a method known as "soft updating" [27]. Specifically, each time the Critic network is updated, the parameters of the Critic network are shifted by a certain ratio to the parameters of the Critic target network. This proportion is controlled by the hyper-parameter τ, and the value of τ is usually set very small, so that the parameters of the Critic target network will slowly follow the parameters of the Critic network. In the SAC learning process, the Actor network is first used to generate an action based on the next state, and then the next state and the action are inputted into the target Q network to obtain the Q value of the next state, which is combined with a reward and a discount factor, and thus the target Q value can be obtained. The action output from the Actor network makes the action entropy and the value of the action output from the Critic network become larger, which indicates that the action taken is a better action, and makes the Critic target network assess the value of the current state with a larger value [28].

$$H(\pi(\cdot|s_{t+1})) = -\lg\pi(a_{t+1}|s_{t+1}) \tag{8}$$

The action entropy is given by Equation (8). The entropy of a policy can be thought of as a measure of its randomness. Policies with high entropy are more random, while those with low entropy are more deterministic. By maximizing entropy, SAC encourages agents to explore the environment more fully.

$$\hat{Q}(s_t, a_t) = r(s_t, a_t) + \gamma E_{s_{t+1}\sim p}[V_{\overline{\psi}}(s_{t+1})] \tag{9}$$

γ is the discount factor that determines the importance of future rewards in the current decision. When γ is close to one, the algorithm places more importance on future rewards and considers long-term payoffs. When γ is close to zero, the algorithm places more importance on immediate rewards and ignores long-term rewards. The formula for Critic target network value estimation $\hat{Q}(s_t, a_t)$ is shown in Equation (9).

$$V_\varphi(s_t) = E_{a_t\sim\pi}[Q(s_t, a_t) - \log\pi(a_t|s_t)] \tag{10}$$

$$J_V(\psi) = E_{s_t\sim D}[\frac{1}{2}\left(V_\psi(s_t) - E_{a_t\sim\pi_\phi}[Q_\theta(s_t, a_t) - \log\pi_\phi(a_t|s_t)]\right)^2 \tag{11}$$

$$\overset{\wedge}{\nabla}_\psi J_V(\psi) = \nabla_\psi V_\psi(s_t)(V_\psi(s_t) - Q_\theta(s_t, a_t) + \log\pi_\phi(a_t|s_t)) \tag{12}$$

Equation (10) is a value function that represents the expected value of choosing an action a_t according to the strategy in the state s_t. This expected value consists of the Q-value function $Q(s_t, a_t)$ minus the logarithmic probability $\log\pi(a_t|s_t)$. Equation (11) is the parameter ψ used by the value objective function to update the value network. The purpose of this objective function is to minimize the difference between the value function $V_\psi(s_t)$ and the expected value in Equation (10), and Equation (12) is the strategy gradient used to calculate the gradient of the strategy network parameter ψ. This gradient indicates how the parameters of the policy network can be adjusted to minimize the policy objective function $J_V(\psi)$. By using this gradient, the strategy network can be updated to better select actions that maximize long-term returns. Maximizing the strategy objective function $V_\varphi(s_t)$ in Equation 6 allows the strategy network to select those actions that have a high Q-value in the current state, whilst maintaining a certain level of exploration. By minimizing the value objective function $J_V(\psi)$ in Equation (11), the value network can be made to estimate the value of the current strategy more accurately. Using Equation (12), it is possible to progressively optimize the strategy network so that at each step it is tuned toward maximizing long-term returns. D is a replay buffer and the gradient of Equation (11).

$$J_Q(\theta) = E_{(s_t,a_t)\sim D}[\frac{1}{2}(Q_\theta(s_t, a_t) - \hat{Q}(s_t, a_t))^2] \tag{13}$$

$$\hat{\nabla}_\theta J_Q(\theta) = \nabla_\theta Q_\theta(a_t, s_t)(Q_\theta(s_t, a_t) - r(s_t, a_t) - \gamma V_{\overline{\psi}}(s_{t+1})) \tag{14}$$

The soft Q-function parameters θ can be trained to minimize the soft Bellman residual in Equation (13), and calculate the gradient of Equation (13) to obtain Equation (14) to update parameter θ.

$$J_\pi(\phi) = E_{s_t \sim D, \varepsilon \sim N}[\log \pi_\phi(f_\phi(\varepsilon_t; s_t)|s_t) - Q_\theta(s_t, f_\phi(\varepsilon_t; s_t))] \tag{15}$$

$$\hat{\nabla}_\phi J_\pi(\phi) = \nabla_\phi \log \pi_\phi(a_t|s_t) + (\nabla_{a_t} \log \pi_\phi(a_t|s_t) - \nabla_{a_t} Q(s_t, a_t)) \nabla_\phi f_\phi(\varepsilon_t; s_t) \tag{16}$$

$\nabla_\phi \log \pi_\phi(a_t|s_t)$ is the gradient of the logarithmic probability of the strategy, representing the gradient of the probability of choosing the action a_t in state s_t.

$\nabla_{a_t} \log \pi_\phi(a_t|s_t)$ is the gradient of the log probability with respect to the action a_t, which represents the rate of change in the probability of selecting the action a_t in state s_t.

$\nabla_{a_t} Q(s_t, a_t)$ is the gradient of the Q-value function with respect to action a_t, and represents the rate of change in the Q-value of selecting action a_t in state s_t.

$\nabla_\phi f_\phi(\varepsilon_t; s_t)$ is the gradient of the parameter ϕ of the policy network, which represents the rate of change in the output of the policy network under the state s_t and random noise ε_t.

The composite gradient computation method, by combining the gradient of the log probability and the gradient of the Q-value function, is able to more accurately tune the parameters of the strategy network. By including the gradient term of log probability, this method is able to maintain a certain degree of exploratory nature so that the strategy network does not converge to a suboptimal solution too early. By combining multiple gradient terms, this method is able to find the optimal policy faster, thus speeding up the training process of the policy network. By combining the gradients of the log-probability and Q-value functions, this method is able to provide more stable gradient estimates, which reduces fluctuations during the training process.

The pseudo-code of the general SAC algorithm is shown in Algorithm 1 and the parameters are updated as shown in Equations (8)–(16).

Algorithm 1: The pseudo-code of the general SAC algorithm

Initialize parameter vectors ψ, $\overline{\psi}$, ϕ, θ
for each iteration **do**
  for each environment step **do**
    $a_t \sim \pi_\phi(a|s)$
    $s_{t+1} \sim p(s_{t+1}|s_t, a_t)$
    $D \leftarrow D \cup \{(s_t, a_t, r(s_t, a_t), s_{t+1})\}$
  end for
  for each gradient step **do**
    $\psi \leftarrow \psi - \lambda_V \hat{\nabla}_\psi J_V(\psi)$
    $\theta_i \leftarrow \theta_i - \lambda_Q \hat{\nabla}_{\theta_i} J_Q(\theta)$ $for \quad i \quad \in \quad \{1, 2\}$
    $\phi \leftarrow \phi - \lambda_\pi \hat{\nabla}_\phi J_\pi(\phi)$
    $\overline{\psi} \leftarrow \tau\psi + (1 - \tau)\overline{\psi}$
  end for
end for

The SAC algorithm is designed for continuous action spaces. It allows the algorithm to select from a continuous range of actions by outputting a probability distribution of actions, which is necessary for fine control in path planning. Secondly, the SAC algorithm introduces the concept of entropy, which encourages the strategy to explore more combinations of actions and avoids premature convergence to a locally optimal solution. SAC reduces the

variance during training and improves the stability of the algorithm through the use of dual Q-networks and goal-policy smoothing techniques. In path planning, this means that the algorithm is able to more reliably converge to a valid solution, maintaining performance even in the face of environmental noise and dynamic changes.

3. Method

In this section, we propose a path-planning algorithm based on the improved SAC algorithm. By improving the randomly sampled replay buffer to a replay buffer based on dual priority sampling of time and TD-error, and modifying the action selection part, OU noise is added to improve the exploration ability of the agent. Expert experience helps the agent find the optimal route with faster training speed and avoid falling into local optima, thus achieving stable training. The sampling method of replay buffer in the traditional SAC algorithm is random sampling; random sampling suffers from low sample utilization, lack of sample efficiency, and delayed agent convergence process; as shown in the paper [29], the random sampling is modified to replay buffer of ER, which improves the performance of SAC and improves the sampling efficiency.

The experience replay sampling method is shown in Algorithm 2. The traditional SAC algorithm in Step3 uses the rule by uniform sampling, which suffers from low sample utilization, slow learning progress, and unstable training.

Algorithm 2: The pseudo-code of the general sampling process of the experience replay buffer

Step1. Agent interacts with the environment
Step2. Store experience in the replay buffer
Step3. Sample from replay buffer according to rules
Step4. Update strategies and value functions by buffer

3.1. Double-Factor Prioritized Sampling Experience Replay

In the SAC algorithm, replay buffer is an important component for storing and reusing past experiences to improve learning efficiency. The sampling process of the experience replay buffer is shown in Algorithm 2. The sampling process of the experience replay buffer is $D \leftarrow D \cup \{(s_t, a_t, r(s_t, a_t), s_{t+1})\}$ in Algorithm 1. Improvements are made for Step3 of Algorithm 2 by changing the traditional random sampling to prioritized sampling based on dual factors. The first factor is TD-error, and the second is time. By calculating the TD-error of the samples, we can rank the priority of the samples to be learned. If the TD-error is larger, it indicates that the prediction accuracy of the collected samples has significant room for improvement, thus these samples should have a higher learning priority. Factor time into prioritized sampling; i.e., sample more aggressively from recent experience and rank updates to ensure that updates from older data do not overwrite updates from newer data. Combining TD-error with a time factor—taking samples that have been sorted by a prioritized playback mechanism and then multiplying them by a recent weight used to emphasize the most recent experience gives the newest samples a higher probability of being sampled to be captured, and depositing the new probabilities that are generated into the linear space allows the intelligence to ensure that the most recent experience has a higher priority and the earlier experience has a lower priority through linear interpolation when capturing the experience.

$$T_d = Q_j - Q_s \tag{17}$$

The formula for calculating TD-error (T_d) is Equation (17).

$$\begin{cases} |\delta| = \frac{(abs(T_d(Q_1)) + abs(T_d(Q_2)))}{2} \\ p_i = |\delta| + \varepsilon \\ P(i) = \frac{{p_i}^{\beta_1}}{\sum_j {p_j}^{\beta_1}} \end{cases} \tag{18}$$

Next, the TD-error is substituted into the set of probability formulas for preferential sampling, which is Equation (18) where p_i is a priority indicator, $P(i)$ represents the sampling probability, and ε is a constant that prevents the sampling probability from being zero. β_1 is a hyperparameter used to adjust the weight of the priority term $P(i)$, which affects the probability of a sample being selected. By adjusting the value of β_1, it is possible to control the balance between high-priority samples and low-priority samples in the training process.

$$ISW = \left(\frac{1}{N} \times \frac{1}{P(i)}\right)^{\beta_2} \tag{19}$$

The importance sampling weight (ISW) is incorporated through Equation (19), where β_2 is a hyper-parameter used to adjust the weight of high-priority samples in the update.

$$\begin{aligned} J_Q(\theta) &= ISW \times E_{(s_t,a_t)\sim D}[\tfrac{1}{2}(Q_\theta(s_t,a_t) - \hat{Q}(s_t,a_t))^2] \\ &= \left(\tfrac{1}{N} \times \tfrac{1}{P(i)}\right)^{\beta_2} \times E_{(s_t,a_t)\sim D}[\tfrac{1}{2}(Q_\theta(s_t,a_t) - \hat{Q}(s_t,a_t))^2] \end{aligned} \tag{20}$$

Finally, the ISW is incorporated into the loss function of the Q-network in the traditional SAC algorithm, i.e., Equation (13), to form Equation (20). Through the above steps, the samples with high priority can be avoided from being over-learned and the loss function of the Q-network has to be changed to Equation (20).

$$\begin{cases} |\delta|_{new} = RW \times \frac{(abs(T_d(Q_1)) + abs(T_d(Q_2)))}{2} \\ p_{i(new)} = |\delta|_{new} + \varepsilon \\ P(i)_{new} = \frac{{p_i}^{\beta_1}}{\sum_j {p_j}^{\beta_1}} \end{cases} \tag{21}$$

Time-based priority sampling, i.e., emphasizing recent experience sampling, allows the most recent samples to have a higher probability of being sampled to be captured by assigning a sampling priority recent weight. The addition of a recent weight (RW) to the original TD-error is used to adjust the TD-error priority. In this way, the algorithm ensures that the most recent experience will have a higher probability of being resampled, thus accelerating the integration of the most recent information in the learning process. The new sampling probability is reflected in Equation (21).

Equation (21) is a fusion sampling probability that combines a TD-error priority-based playback mechanism with an experience pool theory that emphasizes recent experience. The fusion mechanism of experience pooling enhances the efficiency of utilizing the experience pool, improves the agent's ability to screen for valid samples, and speeds up the convergence of the agent.

3.2. OU Noise in Select Action

For the problem of insufficient continuous exploration ability of the agent, the traditional way to solve the problem is by sampling the greedy algorithm. However, since this paper targets the path planning in continuous action space, the traditional greedy algorithm is not suitable to be applied to the continuous action space. The greedy problem exists in continuous action space, where the possibilities of actions are infinite, and it is not possible to simply enumerate all actions as in discrete space. In contrast, OU noise is a first-order Gaussian Markov process with fixed mean and variance, and is time-dependent; i.e., subsequent noise values are correlated with noise values at the previous moment. The formula for this noise is shown in Equation (22).

$$dx = theta * (mu - x) * dt + sigma * dw \tag{22}$$

Adding the upper OU noise at the time of the agent's action selection can improve the continuous exploration ability of the agent where dx is the difference between the

noise value at the next step and the current noise value, *theta* is the regression coefficient, which indicates the rate at which the noise value regresses to the long-run mean, *mu* is the long-run mean, *x* is the current noise value, *dt* is the time step, *sigma* is the standard deviation of the noise, and *dw* is the random variable that conforms to a standard normal distribution.

After adding OU noise, the agent's action selection process is shown in Figure 5. The noise vector is updated; i.e., the noise vector is updated according to Equation (18) when selecting an action. The OU noise can be superimposed on the action generated by the Gaussian distribution at each action selection.

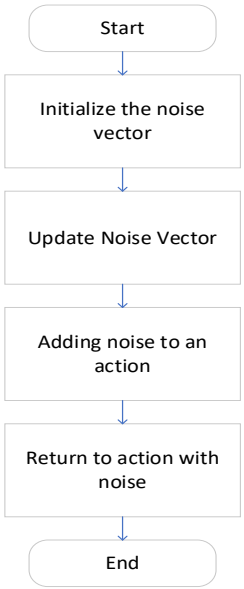


Figure 5. Agent's action selection flow.

3.3. Expert Experience

The SAC neural network part is where the agent acquires data when interacting with the environment and trains the data to obtain a better representation to achieve the desired strategy and value. The features and amount of training data are the most important factors that determine the performance of the neural network model [30]. In order to optimize the characteristics and quantity of training data, the expert experience is added to guide the agent's behavior, which helps the agent choose the appropriate action, accelerate the training process of the agent, and optimize the training data.

Obstacle Avoidance Expert experience: In the early stage of reinforcement learning, the probability of the agent colliding with an obstacle is higher because the agent does not have enough experience but explores on its own, which can lead to a negative impact on the features learned by the agent if there are more data from the collision, while data without collision can lead to the features learned by the agent being too narrow, resulting in insufficient generalization performance. Therefore, the obstacle avoidance expert experience is designed for the agent to help the agent's learning. Meanwhile, in order to improve the exploration efficiency of the agent in each round, the agent will not stop exploring when it encounters an obstacle, but will store all the data in this round into the experience replay.

Among the work in this paper, we classify the types of agent–obstacle collisions into three categories: tangent, intersection, and inclusion. The details are shown in Figure 6 below. In order to ensure that the experience replay can learn enough collision experience, we incorporate the expert experience into the agent according to certain requirements, and this experiment stipulates that when the agent has the same collision type in Figure 6 within five iterations, the expert experience will be imported to start avoiding collisions. The specific operation is as follows: when the agent detects that the distance between an obstacle and the agent is less than the set safety threshold, the agent is forced to give the

agent an action that is contrary to the direction of the obstacle, and when the agent detects that the distance between the agent and the obstacle is greater than the set safety threshold, the agent continues to explore freely.

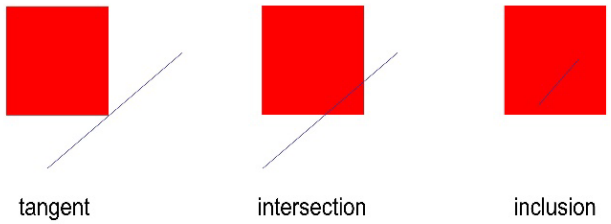


Figure 6. Types of agent collision with obstacles.

Through the above mechanism, the experience replay data of the agent can be optimized to improve the learning efficiency and training speed of the agent.

Make a normal line along the direction of Action and make symmetry of θ of Action with respect to the normal line to generate a new θ′ to replace Action. The specific operation is shown in Figure 7 below.

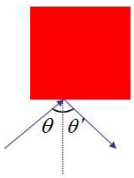


Figure 7. Demonstration of expert experience in obstacle avoidance.

Expert experience in path planning: The agent's exploration in the environment is mainly based on free exploration in the early stage, but the free exploration time is too long, which will lead to the inefficiency of the agent in completing the task. In order to improve the agent's path-planning ability, expert experience is added to help the agent learn actions closer to the endpoint.

In this paper, expert experience has been combined with an artificial potential field approach to integrate the ideas of gravitational and repulsive forces into agent training. The angle and distance between the agent and the target point are used as the base knowledge of the expert experience. The angle is defined as the absolute value of the angle between the line segment connecting the agent to the endpoint and the direction of movement of the agent, with the counterclockwise angle being positive and the clockwise angle being negative. The angle ϕ can be from 0 to 180 degrees. The distance is defined as the distance between the agent and the endpoint. With the definition of angle and distance, positive incentives and negative penalties are designed. From the size of the angle, we can ascertain the direction relationship between the robot and the end point; if the angle value is less than 90 degrees, it represents that the agent is moving toward the endpoint; if the angle value is more than 90 degrees, it represents that the agent is moving away from the endpoint. The distance is added on the basis of the angle if the angle is less than 90 degrees, but the agent may follow the angle to move and not touch the endpoint, so the concept of distance is added to help the agent close to the endpoint; when the distance is far away, a certain punishment is given, and when the distance is close, a certain reward is given. The expert experience through the angle and distance provides good help for the training of the agent, which accelerates the convergence speed of the agent and the stability of the training.

$$U_{att}(q,\phi) = \begin{cases} \frac{1}{2}k_{\text{att}}\rho^2(q) \ + k_{\text{att}}\frac{90}{\phi} & \rho(q) \le \rho^*_{goal}(q), \phi < 90 \\ \frac{1}{2}k_{\text{att}}\rho^2(q) + k_{\text{att}}\frac{\phi}{90} - k_{\text{att}} & \rho(q) \le \rho^*_{goal}(q), \phi \ge 90 \\ \rho^*_{goal}(q)k_{\text{att}}\rho(q) - \frac{1}{2}k_{\text{att}}{\rho^*_{goal}}^2(q) + k_{\text{att}}\frac{90}{\phi} & \rho(q) > \rho^*_{goal}(q), \phi < 90 \\ \rho^*_{goal}(q)k_{\text{att}}\rho(q) - \frac{1}{2}k_{\text{att}}{\rho^*_{goal}}^2(q) + k_{\text{att}}\frac{\phi}{90} - k_{\text{att}} & \rho(q) > \rho^*_{goal}(q), \phi \ge 90 \end{cases} \tag{23}$$

Combining the above experience with the artificial potential field method gives Equation (23). k_{att} is the gain coefficient, a positive constant. $\rho(q)$ represents the Euclidean distance between the current position and the end position. $\rho^*_{goal}(q)$ represents the Euclidean distance between the start position and the end position. Equation (23) is determined based on two parameters. The first parameter is the distance $\rho(q)$ moved by the agent compared with the distance $\rho^*_{goal}(q)$ between the start point and the target point. The second parameter is based on the defined angle ϕ in relation to 90°. If $\rho(q) \leq \rho^*_{goal}(q)$ and $\phi < 90$ then it is executed $\frac{1}{2}k_{\text{att}}\rho^2(q) + k_{\text{att}}\frac{90}{\phi}$. If $\rho(q) \leq \rho^*_{goal}(q)$ and $\phi \geq 90$, then it is executed $\frac{1}{2}k_{\text{att}}\rho^2(q) + k_{\text{att}}\frac{\phi}{90} - k_{\text{att}}$. If $\rho(q) > \rho^*_{goal}(q)$ and $\phi < 90$, then it is executed $\rho^*_{goal}(q)k_{\text{att}}\rho(q) - \frac{1}{2}k_{\text{att}}{\rho^*_{goal}}^2(q) + k_{\text{att}}\frac{90}{\phi}$. If $\rho(q) > \rho^*_{goal}(q)$ and $\phi \geq 90$, then it is executed $\rho^*_{goal}(q)k_{\text{att}}\rho(q) - \frac{1}{2}k_{\text{att}}{\rho^*_{goal}}^2(q) + k_{\text{att}}\frac{\phi}{90} - k_{\text{att}}$. The specific value is calculated by Equation (23).

The overall flow of the DPEOU-SAC algorithm is shown in Figure 8. Innovations are made on the basis of the SAC algorithm. OU noise and expert experience in obstacle avoidance are added in the action selection part. Expert experience in path planning based on the artificial potential field method is added in the reward function part. Double-priority sampling is used for the experience pool part, and the network gradient updating part is improved.

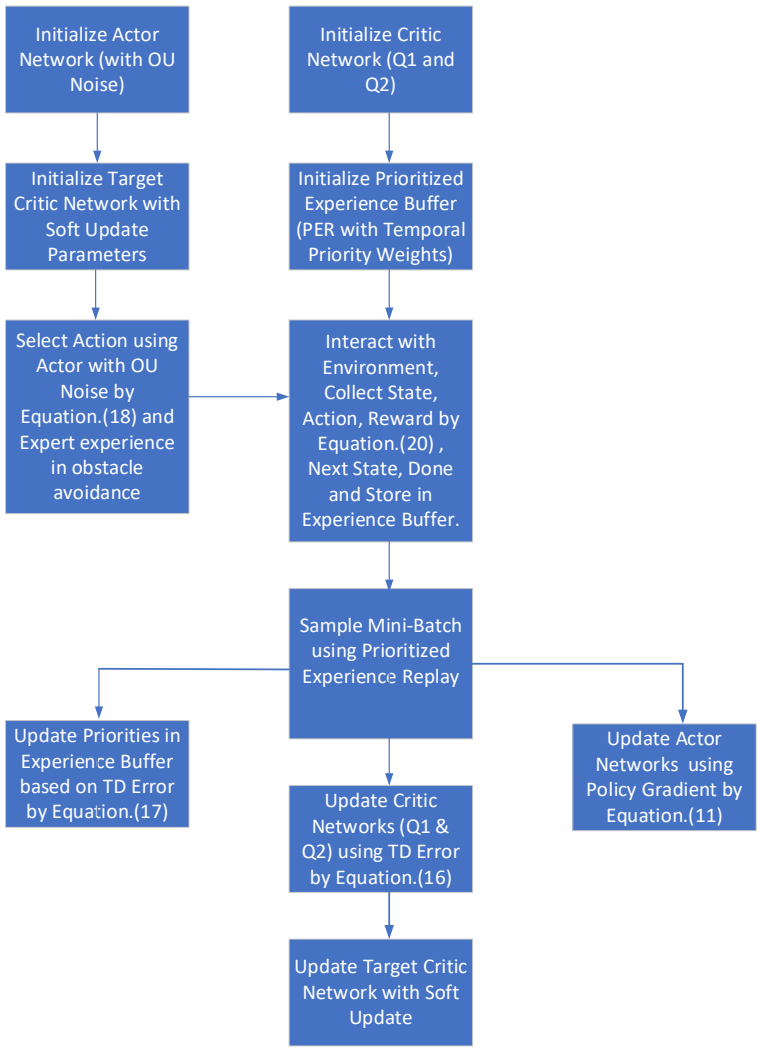


Figure 8. DPEOU-SAC algorithm flowchart.

4. Simulations and Analysis

In order to verify the effectiveness of the DPEOU-SAC algorithm as stated above, two 20×20 map grid virtual static environments in the gym environment are established in Python, which is shown in Figure 9. The red circles and rectangles in Figure 9 represent static obstacles. The generalization of the algorithm is supposed to be verified with different situations, no matter in a simple environment or complex environment. On the basis of the first static simulation experiment, we have also added some dynamic obstacles moving in different directions, as shown in Figure 10. The red circles and rectangles in Figure 10 represent static obstacles. The blue rectangle in Figure 10a is a dynamic obstacle moving along the y-axis with a moving speed of 2 m/s at the starting coordinates $(-1, -1)$, the blue rectangle in Figure 10b is a dynamic obstacle moving along the x-axis with a moving speed of 2 m/s at the starting coordinates $(-1, 0)$, and the green rectangle is a dynamic obstacle moving along the y-axis with a moving speed of 2 m/s at the starting coordinates $(0, -1)$. The green rectangle is a dynamic obstacle moving along the y-axis with a speed of 2 m/s at the start coordinate $(0, -1)$. The starting point of each of the above four maps is denoted by $\times$, and the ending point is denoted by $\bullet$. The coordinates of the starting point of all maps are $[-10, -9]$ and the coordinates of the ending point are $[9, 10]$. They represent different situations where the robot meets dynamic obstacles to verify the dynamic obstacle avoidance performance of the DPEOU-SAC.

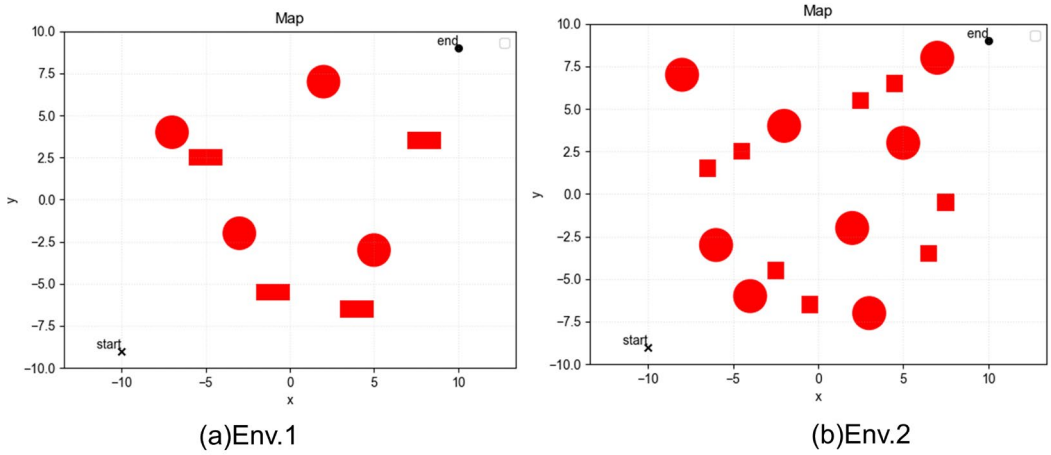


Figure 9. Virtual static environmental models: (**a**) simple static environmental model 1 (Env.1); (**b**) complex static environmental model 2 (Env.2).

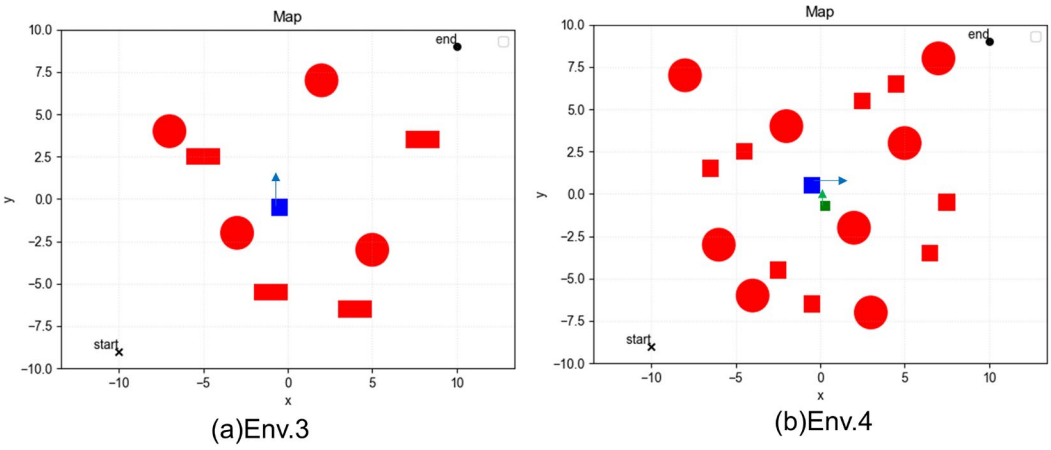


Figure 10. Virtual dynamic environmental models: (**a**) simple dynamic environmental model 3 (Env.3); (**b**) complex dynamic environmental model 4 (Env.4).

All the experiments were performed on the same computer. The parameters of each experiment are recorded in Table 1. All experiments were run based on the following configuration: RTX3060 32G for GPU information, i5-12600KF for CPU information, Python 3.9, and PyTorch 2.11. The simulation environment is OpenAI's gym.

Table 1. Initial parameter of simulation in all environments.

Parameter Name	Value
Max_steps	200
Gamma	0.99
Alpha	0.2
Memory_Size	20,000
Lr_critic	1×10^{-3}
Lr_actor	1×10^{-3}
Lr_alpha	1×10^{-3}
Buffer_alpha_init	0.6
Buffer_alpha_end	0.4
Buffer_beta	0.4
Recentness_weight	0.1
Batch_size	128

Since SAC is an algorithm based on continuous action space, this experiment sets the action range of the agent as shown below:

Action: agent movement range $v \in [-1, 1]$, agent movement angle range $\theta \in [0, 360]$.

Reward function is an important factor used to help the agent to train and reach the goal and has the function of allowing the agent to reinforce the desired behavior and punish wrong behavior [31]. In this paper, the traditional reward function is combined with the APF-based expert experience proposed by Equation (23) to form a reward function as shown in Equation (24).

$$r = -d - \frac{d\theta all}{num\_pos} - num\_theta - num\_crash - num\_over + U_{att}(q, \phi) \tag{24}$$

where d represents the total length, $d\theta all$ represents the total angular change, and $\frac{d\theta all}{num\_pos}$ is used to compute the average angular change per navigational point, which is used to measure the energy change. num_theta represents the number of unreasonable corners. num_crash represents the number of collisions, and num_over represents the number of states that are out of constraint. $U_{att}(q, \phi)$ is the expert experience based on the artificial potential field method derived from Equation (23).

Some targeted terms will appear in the next experiments, so let us explain them here first. The first term is "unsmooth converges". In order to make the experiment closer to reality and ensure that the path planned by the agent is smooth enough, we set up "unsmooth converges" to observe whether the path generated by the agent is smooth or not. We set the angle change between the current segment of the agent's path and the previous segment of the path to be ≤ 45 degrees. When the angle is > 45 degrees, the generated path is considered to be unsmooth. The number of unsmoothed converges per turn for AGENT can be obtained by num_theta. The number of "unsmooth converges" to zero means that during the training period of the agent, there will be no unsmoothness in the path planning of the agent after that round.

The second term is "collisions converge". To ensure that the path planned by the agent is safe, i.e., the agent must avoid obstacles while planning the path, we set "collisions converge" to observe whether the path generated by the agent is safe or not. The number of collisions converge that occurs in each turn of the agent can be obtained by num_crash. The number of "collisions converge" to zero means that during the training period of the agent, there will be no more collisions when the agent is planning the path after that round.

4.1. Static Obstacle Avoidance Simulation Experiments

It is assumed that the mobile robot is considered to be a point, and it can only move within a two-dimensional grid area. The experimental environments of the static obstacle avoidance simulation experiments are shown in Figure 9, and Env.1 and Env.2 are Figure 9a,b, respectively. Env.1 has a fewer number of obstacles and sparser locations compared to Env.2, and the above two environments are used as control groups. In this paper, seven sets of comparison experiments are established in two different map environment models. The static obstacle avoidance simulation experiments actually test the static properties of the path planning algorithm. When only the convergence speed of the algorithm is considered, the path length of the planned path, the path smoothness, and the path safety are used as the measurement criteria. The conventional SAC, DDPG, TD3, and the modified PER-SAC, DPE-SAC, and DPEOU-SAC are trained in Env.1 and Env.2, respectively, and the performance of the algorithms is evaluated by observing the training results.

In Env.1, the number of iterations of the algorithms is set to 800. The trajectory diagrams of the seven algorithms are shown in Figure 11. Figure 11a–f shows the static path planning and static obstacle avoidance effects of DDPG, TD3, SAC, OU-SAC, PER-SAC, and DPE-SAC in Env.1, respectively. Figure 11e shows the static path planning and static obstacle avoidance effect of the proposed algorithms in this paper. DPEOU-SAC with static path planning and static obstacle avoidance effects is shown in Env.1.

Seven different algorithms are run in Env.1 to obtain the specific performance index, as shown in Table 2. From Table 2, the number of collisions and the number of unsmooth converges to zero can be observed, with the DPEOU-SAC proposed in this paper converging to zero the fastest, in which the number of rounds of collision convergence to zero is 38, and the number of rounds of unsmooth converges to zero is 49, far faster than the other algorithms. When the number of collisions converges to zero and the number of unsmooth converges to zero at the same time, the agent is considered to be able to avoid obstacles normally and plan a feasible path. The traditional DDPG, TD3, and SAC algorithms converge at 513, 672, and 445 rounds, respectively, which are slower than the convergence ability of the DPEOU-SAC proposed in this paper. The improved SACs such as OU-SAC, PER-SAC, and DPE-SAC converge at 209, 118, and 93 rounds, respectively, compared to the original SAC, but the convergence speed is still slower compared to the DPEOU-SAC proposed in this paper.

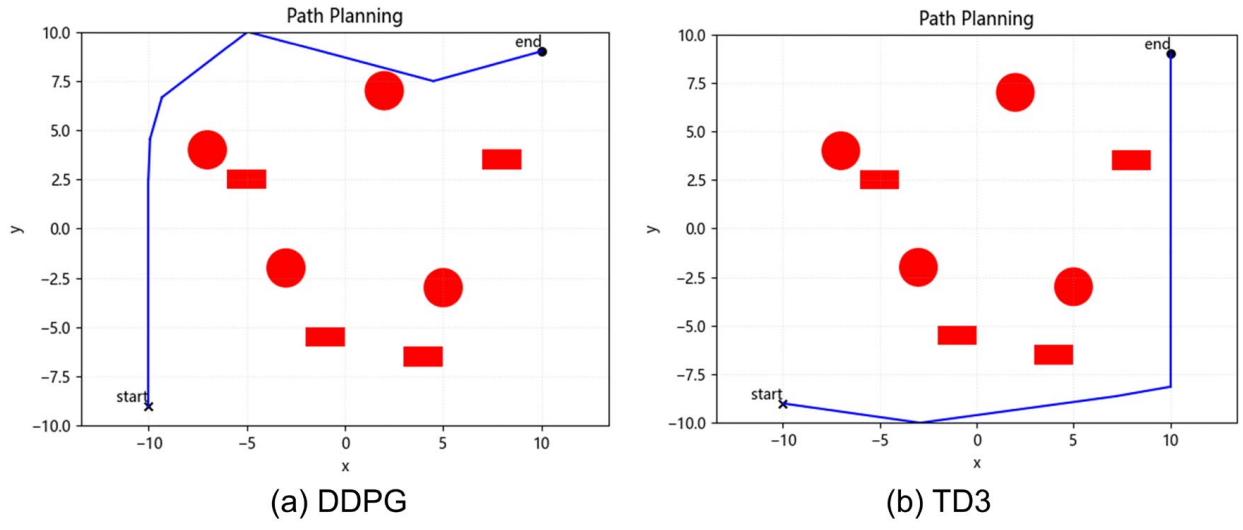


Figure 11. *Cont.*

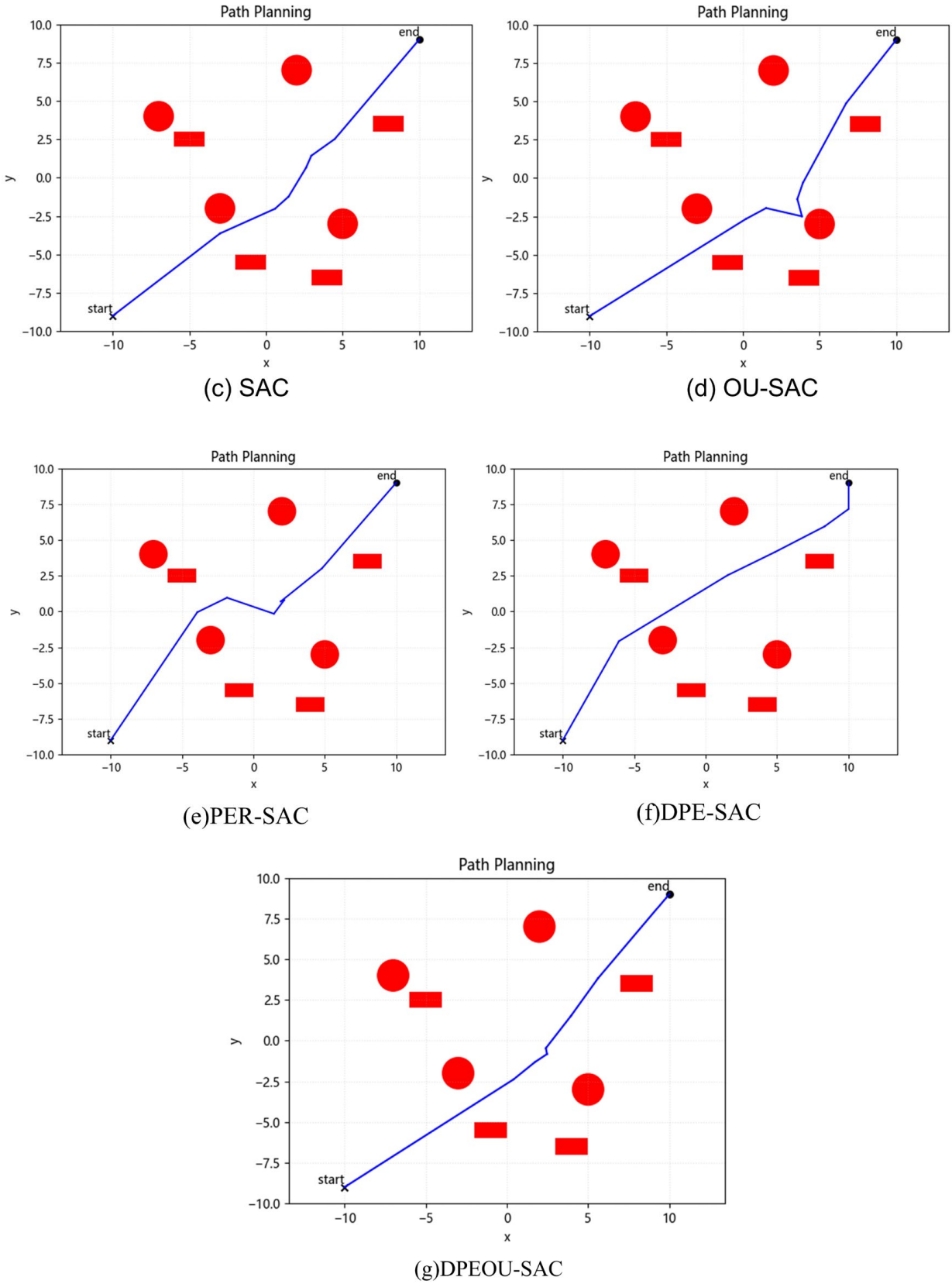


Figure 11. Different algorithms' trajectories in Env.1.

Table 2. Different algorithms' results in Env.1.

Algorithm Indicator	DDPG	TD3	SAC	OU-SAC	PER-SAC	DPE-SAC	DPEOU-SAC
collisions converge to 0	434	672	445	209	112	93	38
unsmooth converges to 0	513	672	445	209	118	93	49
length	37.46	36.72	27.47	27.91	29.14	28.34	27.31

The length in Table 2 represents the length of the paths planned by each algorithm in Figure 11, where the path lengths planned by DDPG, TD3, and SAC algorithms are 37.46, 36.72, and 27.47. The SAC algorithm plans shorter paths than the DDPG and TD3. Improved SACs such as OU-SAC, PER-SAC, DPE-SAC, and DPEOU-SAC plan out path lengths of 27.91, 27.14, 28.34, and 27.31, respectively. It can be observed that the DPEOU-SAC algorithm plans out the shortest paths.

$$ipc = \frac{Con2 - Con1}{Con2} \tag{25}$$

In order to reflect the performance of the improved SAC algorithm more intuitively, this paper proposes Equation (25), in which the ipc metric is used to measure the percentage improvement in the convergence speed of the improved SAC algorithm relative to the original SAC algorithm. $Con2$ is the number of convergence rounds of the original SAC algorithm. $Con1$ is the number of convergence rounds of the improved SAC algorithm. First, the difference between the number of converged rounds of the original SAC algorithm and the improved SAC algorithm is calculated, i.e., $Con2 - Con1$. Then, the difference in the number of converged rounds of the original SAC algorithm $Con2$ is divided to obtain the percentage of improvement. By comparing the number of convergence rounds of the improved algorithm with that of the original algorithm, the degree of improvement in convergence speed of the improved algorithm can be visualized. Specifically, the larger the ipc value, the more significant the improvement in the convergence speed of the improved algorithm. The percentage of improved performance obtained by Equation (25) is shown in Table 3. From Table 3, it can be observed that the level of improvement in the convergence performance of the DPEOU-SAC algorithm is much higher than the other algorithms. The superiority of the DPEOU-SAC algorithm is proved.

Table 3. Convergence improvement metrics for the improved SAC algorithm in Env.1.

Algorithm Indicator	OU-SAC	PER-SAC	DPE-SAC	DPEOU-SAC
ipc	53.03%	74.83%	79.10%	88.99%

In Figure 12, all the algorithms converge in the value of total reward convergence, but the DDPG, TD3, and SAC algorithms fluctuate in the value of convergence and converge in a higher number of rounds. The DPE-SAC algorithm converges in a smaller value, but since the convergence is slower than the DPEOU-SAC algorithm, the DPEOU-SAC outperforms the DPE-SAC algorithm.

Among Env.1, DPEOU-SAC has the fastest convergence speed and the shortest planned path. In order to further verify the performance of the proposed DPEOU-SAC algorithm, a second set of static obstacle avoidance simulation experiments is carried out by selecting Env.2. The obstacle distribution of Env.2 is more complex. Seven algorithms are applied to Env.2 for training to observe the training results.

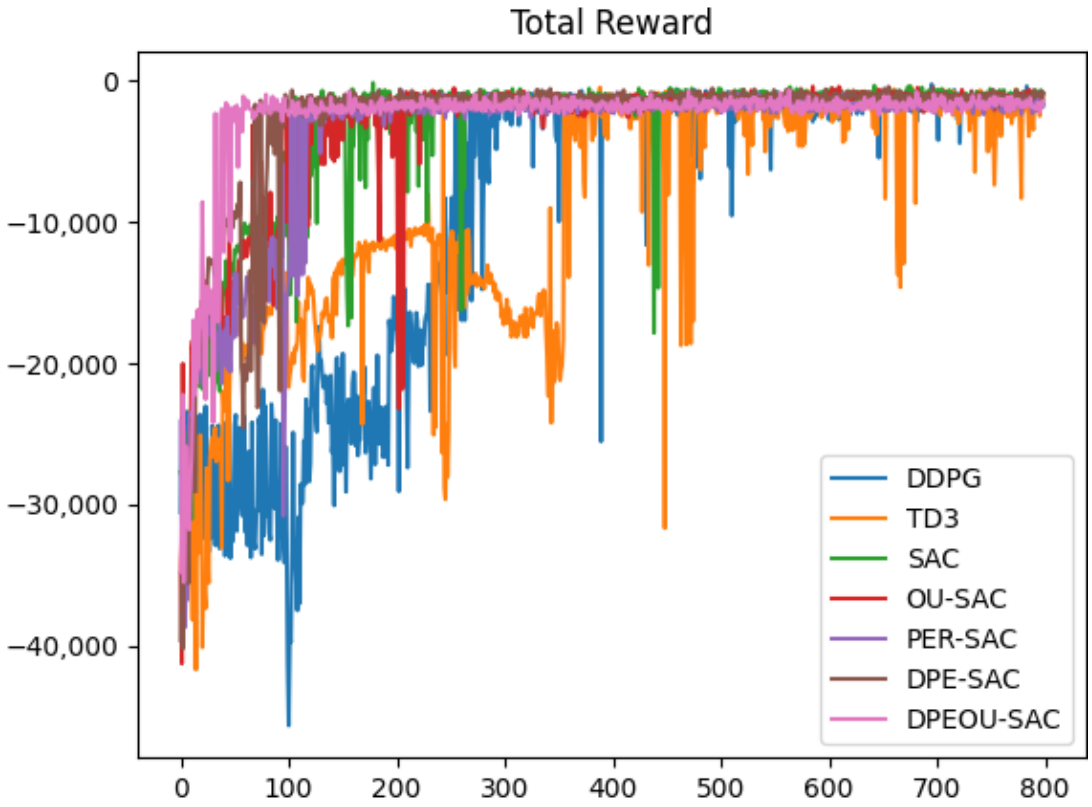


Figure 12. Different algorithms' total rewards in environment 1.

In Env.2, because DDPG and TD3 did not converge in 800 rounds, the number of iterations was set to 1200. The trajectory diagrams of the seven algorithms are shown in Figure 13. Figure 13a–f show the static path planning and static obstacle avoidance effects of DDPG, TD3, SAC, OU-SAC, PER-SAC, and DPE-SAC in Env.2, respectively. Figure 13e shows the static path planning and static obstacle avoidance effect of the proposed algorithms in this paper. DPEOU-SAC with static path planning and static obstacle avoidance effects is shown in Env.2.

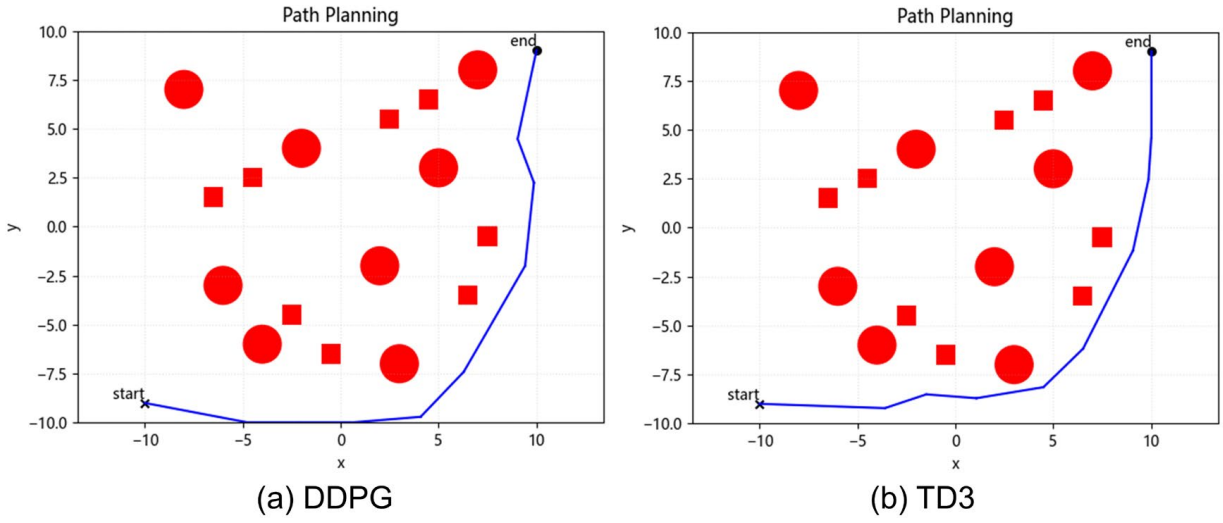


Figure 13. *Cont.*

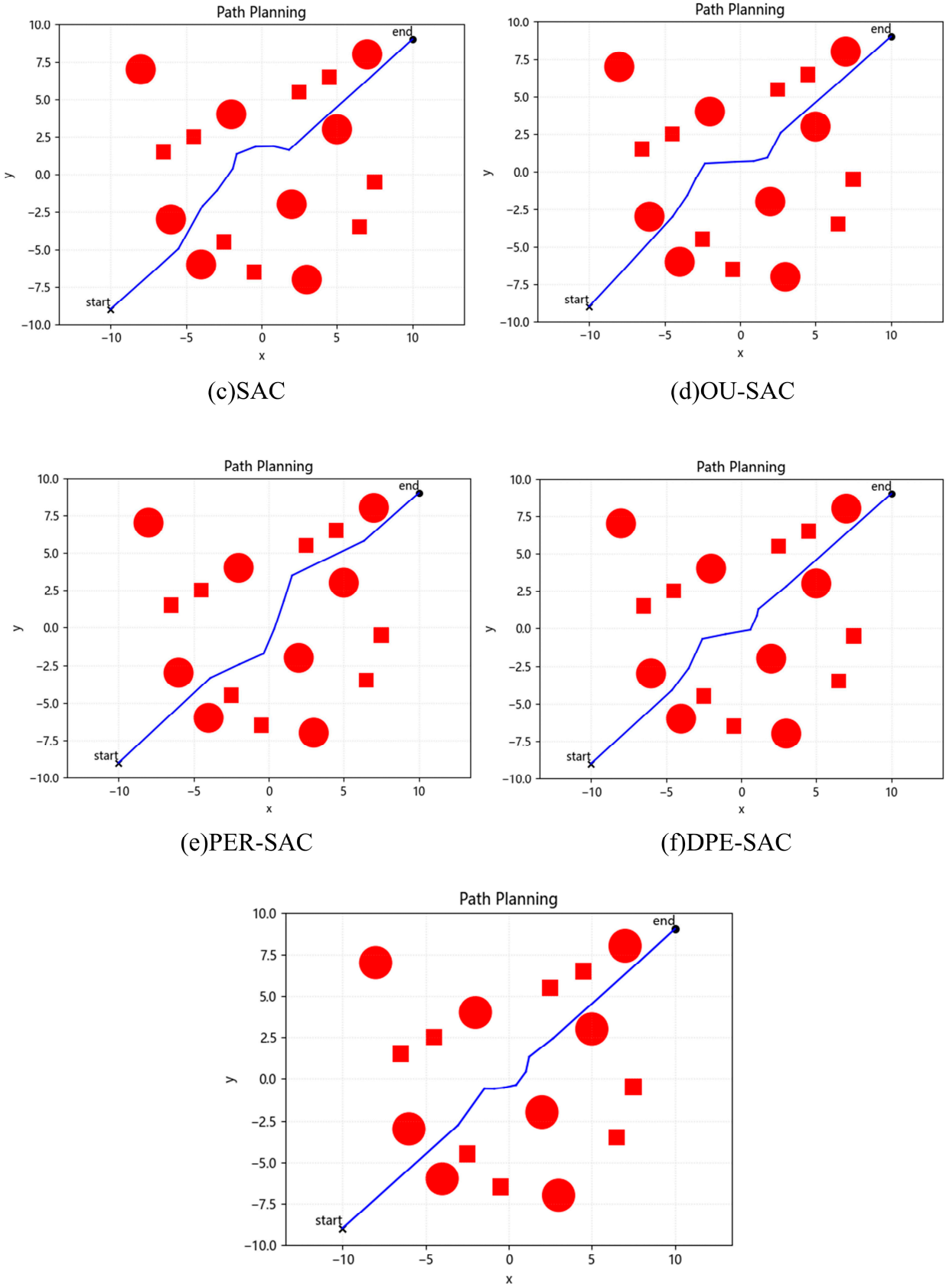


Figure 13. Different algorithms' trajectories in Env.2.

Seven different algorithms are run in Env.2 to obtain the specific performance index, as shown in Table 4. From Table 4, the number of collisions and the number of unsmooth converges to zero can be observed, with the DPEOU-SAC proposed in this paper converging to zero the fastest, in which the number of rounds of collision convergence to zero is 297, and the number of rounds of unsmooth converges to zero is 297, far faster than the other algorithms. When the number of collisions converges to zero and the number of unsmooth converges to zero at the same time, the agent is considered to be able to avoid obstacles normally and plan a feasible path. The traditional DDPG, TD3, and SAC algorithms converge at 965, 807, and 534 rounds, respectively, which are slower than the convergence ability of the DPEOU-SAC proposed in this paper. The improved SAC such as OU-SAC, PER-SAC, and DPE-SAC converge at 439, 317, and 301 rounds, respectively, compared to the original SAC, but the convergence speed is still slower compared to the DPEOU-SAC proposed in this paper.

Table 4. Different algorithms' results in Env.2.

Algorithm Indicator	**DDPG**	**TD3**	**SAC**	**OU-SAC**	**PER-SAC**	**DPE-SAC**	**DPEOU-SAC**
collisions converge to 0	815	807	534	439	317	300	297
unsmooth converges to 0	965	807	534	439	312	301	297
length	35.50	35.36	27.95	28.81	27.94	27.71	27.66

The length in Table 4 represents the length of the paths planned by each algorithm in Figure 13, where the path lengths planned by DDPG, TD3, and SAC algorithms are 35.50, 35.36, and 27.95. The SAC algorithm plans shorter paths than the DDPG and TD3. Improved SACs such as OU-SAC, PER-SAC, DPE-SAC, and DPEOU-SAC plan out path lengths of 27.66, 27.94, 27.71, and 28.81, respectively. It can be observed that the DPEOU-SAC algorithm plans out the shortest paths.

The percentage of improved performance obtained by Equation (25) is shown in Table 5. From Table 5, it can be observed that the level of improvement in the convergence performance of the DPEOU-SAC algorithm is much higher than the other algorithms; the superiority of the DPEOU-SAC algorithm is proved.

Table 5. Convergence improvement metrics for the improved SAC algorithm in Env.2.

Algorithm Indicator	**OU-SAC**	**PER-SAC**	**DPE-SAC**	**DPEOU-SAC**
ipc	17.79%	40.64%	43.82%	44.38%

In Figure 14, the curve of DPEOU-SAC enters the convergence state earlier than the curves of other algorithms, and the fluctuation value is smaller, which represents that the algorithm training is more stable. The DPE-SAC algorithm is the first to enter the first plateau, but still slower than the DPEOU-SAC algorithm to enter convergence. The DDPG algorithm is not stable enough, and the training plateau will fluctuate again after a period of time, while the TD3 algorithm needs a longer training time to converge. The SAC, OU-SAC, and PER-SAC effect compared to DDPG and TD3 shows significant improvement, but compared to the DPEOU-SAC proposed in this paper, it does not improve enough.

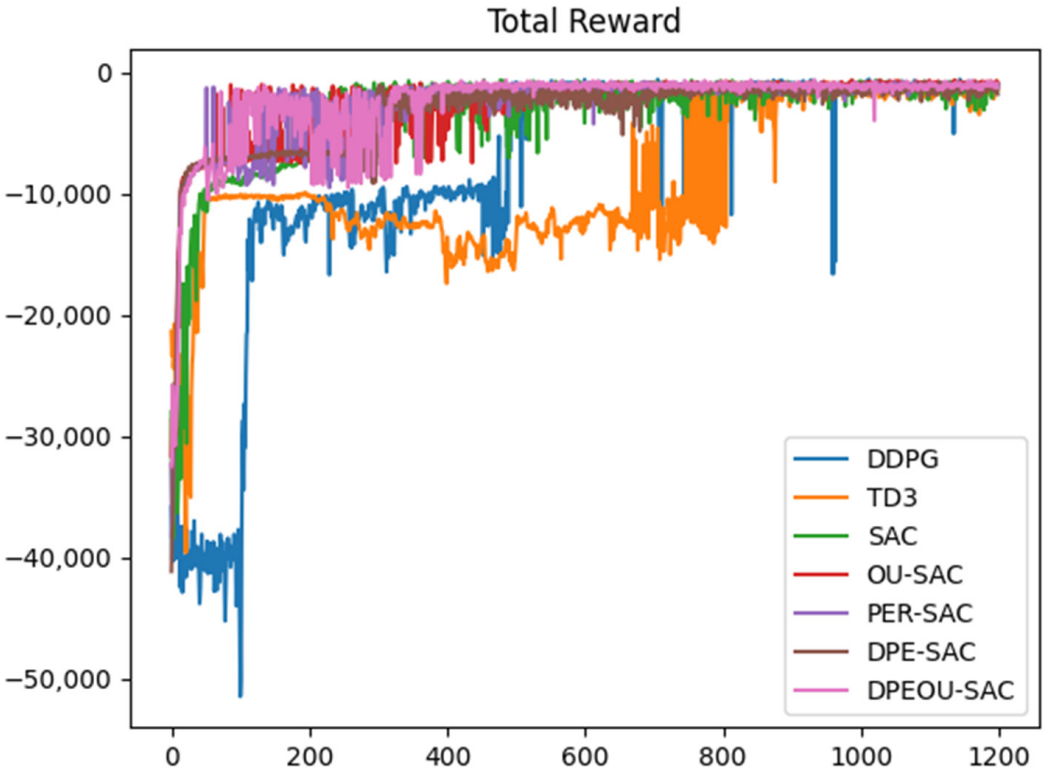


Figure 14. Different algorithms' total rewards in Env.2.

Based on the discussions above, the research results from simulation experiments conducted in Env.1 and Env.2 indicate that, regardless of changes in obstacle distribution, the DPEOU-SAC algorithm can safely and effectively plan a smooth global optimal path in both Env.1 and Env.2. The data from Tables 2–5 demonstrate the feasibility and superiority of the DPEOU-SAC algorithm in static obstacle avoidance path planning. The DPEOU-SAC algorithm can enhance the efficiency of agent training, enabling the agent to plan an appropriate optimal path more quickly.

Path planning is not just about planning in static environments, but also about the presence of dynamic obstacles. Therefore, two experimental environments with dynamic obstacles, Figure 10a,b, were designed for testing. The overall number of obstacles in Figure 10a is less than that in Figure 10b. Figure 10a represents Env.3, and Figure 10b represents Env.4. The superiority of DPEOU-SAC is verified by observing the paths generated by the seven algorithms in Env.3 and Env.4, as well as their performance metrics. The details are given in Section 4.2. Dynamic Obstacle Avoidance Experiment.

4.2. Dynamic Obstacle Avoidance Experiment

The total number of episodes of the algorithm in Env.3 is 800. By adding a collision penalty and an unsmooth penalty to the reward mechanism, the agent learns intelligent obstacle avoidance, which is shown in Figure 15. Figure 15, it can be observed that the agent avoids moving obstacles.

In the simulation experiments in this paper, the agent is a mass by default. In Env.3, Figure 15 shows how the agent avoids dynamic obstacles after training with DPEOU-SAC. Figure 15a shows the moving direction of the dynamic obstacle and the position at the current moment, and the moving speed of the dynamic obstacle is 2 m/s. Figure 15b shows when the dynamic obstacle is about to contact the agent, and the agent makes evasive action in the direction of about 40 degrees to the right. Figure 15c shows the position of the agent in the next moment, and it can be seen that the position of the agent is 15 degrees to the upper right of the dynamic obstacle, which means that the agent has safely avoided the dynamic obstacle.

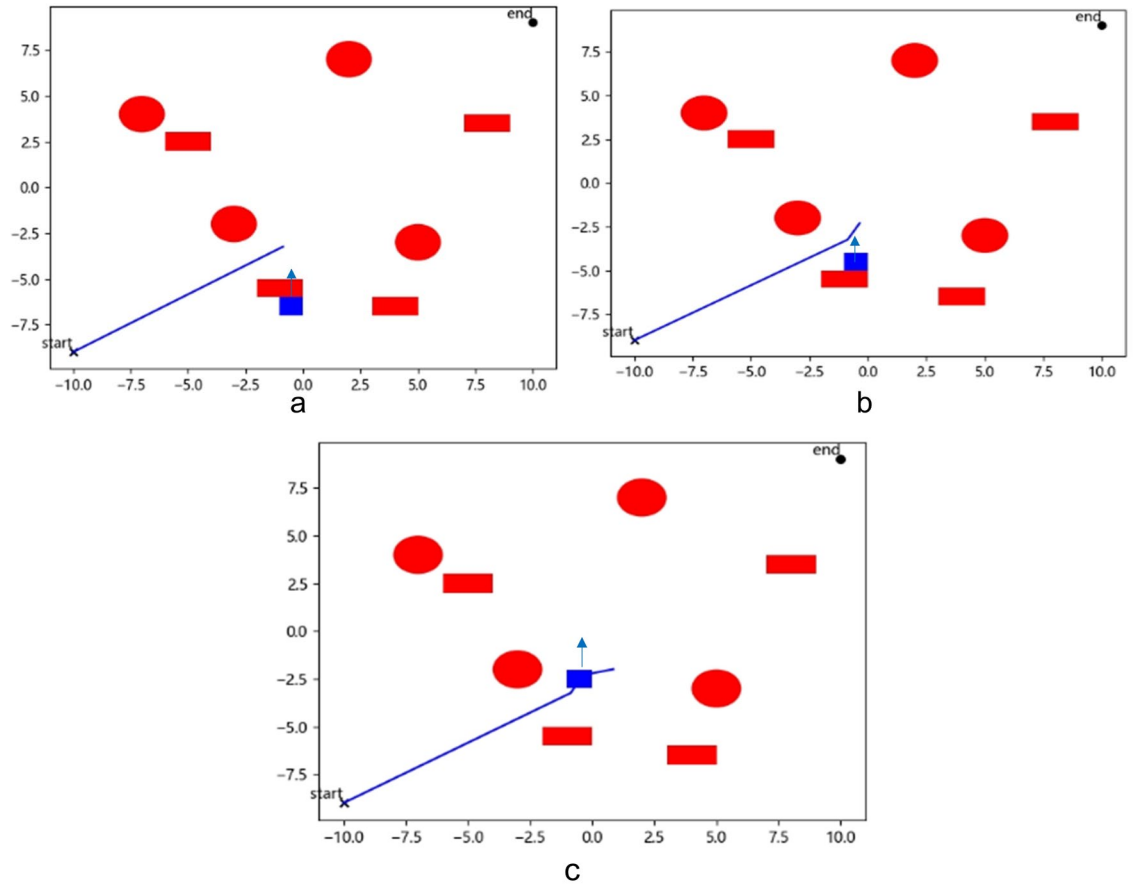


Figure 15. Local dynamic obstacle avoidance effect of DPEOU-SAC in Env.3.

The trajectory diagrams of the seven algorithms are shown in Figure 16. Figure 16a–f show the static path planning and static obstacle avoidance effects of DDPG, TD3, SAC, OU-SAC, PER-SAC, and DPE-SAC in Env.3, respectively. Figure 16e shows the static path planning and static obstacle avoidance effect of the proposed algorithms in this paper. DPEOU-SAC with static path planning and static obstacle avoidance effects is shown in Env.3.

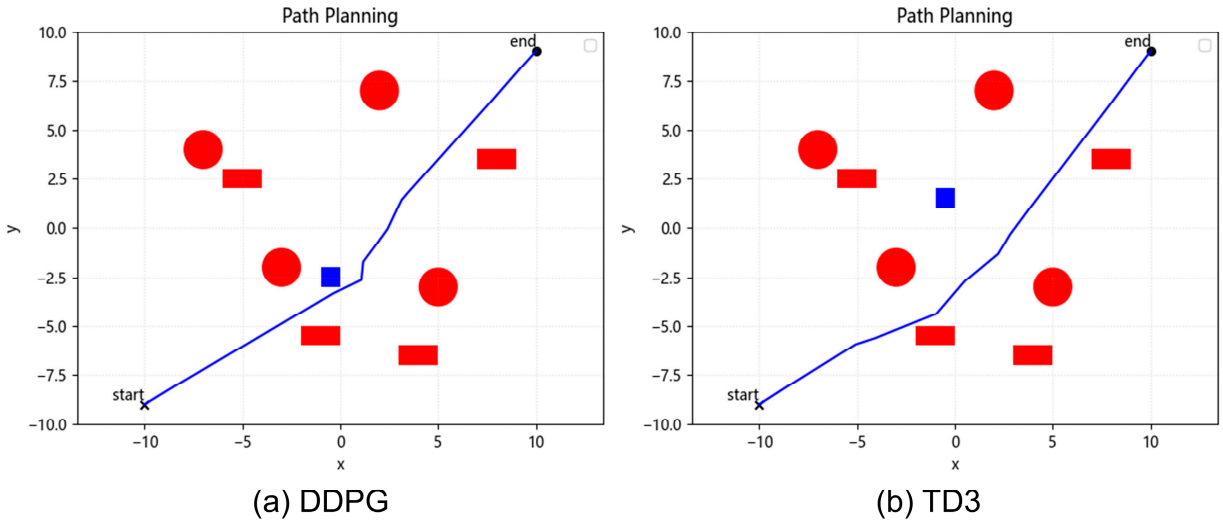


Figure 16. *Cont.*

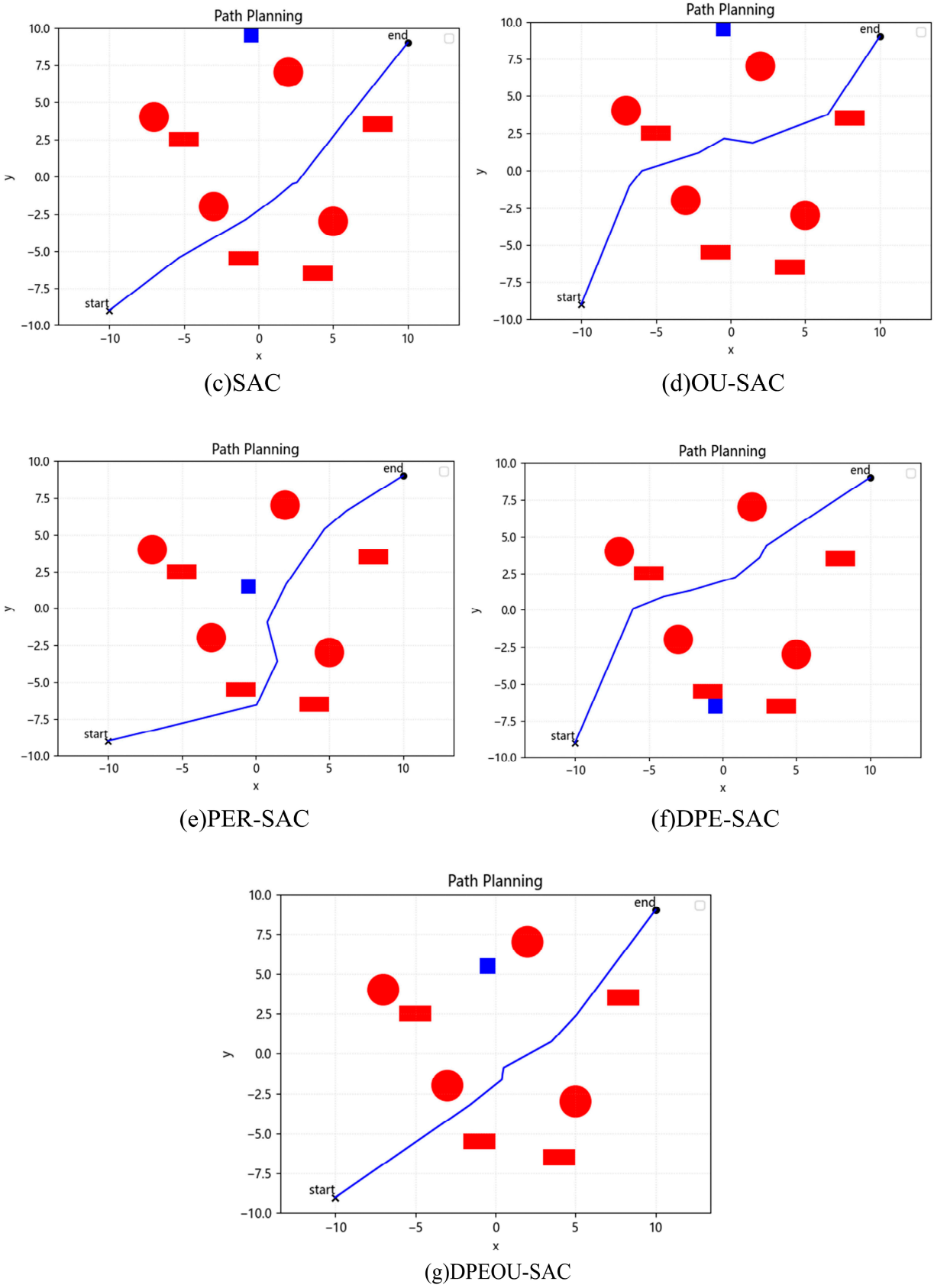


Figure 16. Different algorithms' trajectories in Env.3.

Seven different algorithms are run in Env.3 to obtain the specific performance index, as shown in Table 6. From Table 6, the number of collisions and the number of unsmooth

converges to zero can be observed, with the DPEOU-SAC proposed in this paper converging to zero the fastest, in which the number of rounds of collision convergence to zero is 59, and the number of rounds of unsmooth converges to zero is 59, far faster than the other algorithms. When the number of collisions converges to zero and the number of unsmooth converges to zero at the same time, the agent is considered to be able to avoid obstacles normally and plan a feasible path. The traditional DDPG, TD3, and SAC algorithms converge at 369, 549, and 299 rounds, respectively, which are slower than the convergence ability of the DPEOU-SAC proposed in this paper. The improved SACs such as OU-SAC, PER-SAC, and DPE-SAC converge at 254, 116, and 87 rounds, respectively, compared to the original SAC, but the convergence speed is still slower compared to the DPEOU-SAC proposed in this paper.

Table 6. Different algorithms' results in Env.3.

Algorithm Indicator	DDPG	TD3	SAC	OU-SAC	PER-SAC	DPE-SAC	DPEOU-SAC
collisions converges to 0	369	549	299	254	116	86	59
unsmooth converges to 0	368	513	296	253	116	87	59
length	28.19	27.62	27.29	29.49	30.39	28.45	27.24

The length in Table 6 represents the length of the paths planned by each algorithm in Figure 16, where the path lengths planned by DDPG, TD3, and SAC algorithms are 28.59, 27.62, and 27.29. The SAC algorithm plans shorter paths than the DDPG and TD3. Improved SACs such as OU-SAC, PER-SAC, DPE-SAC, and DPEOU-SAC plan out path lengths of 29.49, 30.39, 28.45, and 27.24, respectively. It can be observed that the DPEOU-SAC algorithm plans out the shortest paths.

From Table 7, it can be seen that the lifting efficiency of DPEOU-SAC with respect to SAC calculated by Equation (25) is 80.27%, while that of OU-SAC is 15.06%, that of PER-SAC is 61.20%, and that of DPE-SAC is 70.20%. It can be found that the algorithm proposed in this paper has obvious advantages over other improved algorithms.

Table 7. Convergence improvement metrics for the improved SAC algorithm in Env.3.

Algorithm Indicator	OU-SAC	PER-SAC	DPE-SAC	DPEOU-SAC
ipc	15.05%	61.20%	70.90%	80.27%

In Figure 17, it can be observed that DPEOU-SAC converges the earliest and the fluctuation after convergence is small. While DPE-SAC takes a slightly longer time to converge than DPEOU-SAC, but faster than the other algorithms. And the traditional DDPG, SAC, and TD3 take a longer time to converge.

In order to verify the generality of the DPEOU-SAC algorithm in dynamic obstacle avoidance, seven algorithms are trained in Env.4. Env.4 adds more static obstacles and new moving obstacles compared to Env.3, and the performance of the algorithms is judged by observing the experimental results.

The total number of episodes of the algorithm in Env.4 is 800.

By adding a collision penalty and an unsmooth penalty to the reward mechanism, the agent learns intelligent obstacle avoidance, which is shown in Figure 18. Figure 18, it can be observed that the agent avoids moving obstacles.

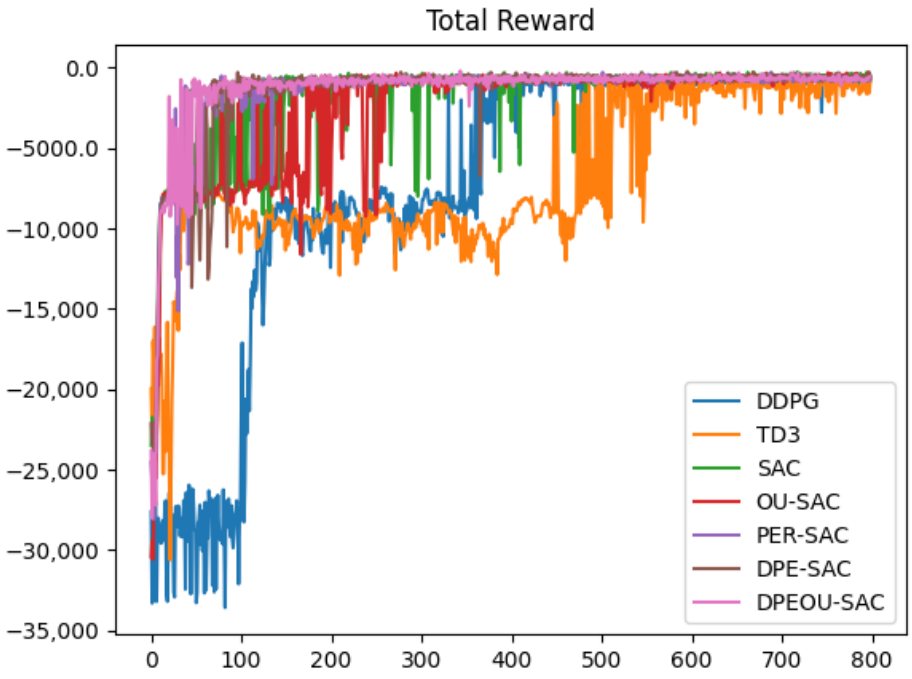


Figure 17. Different algorithms' total rewards in Env.3.

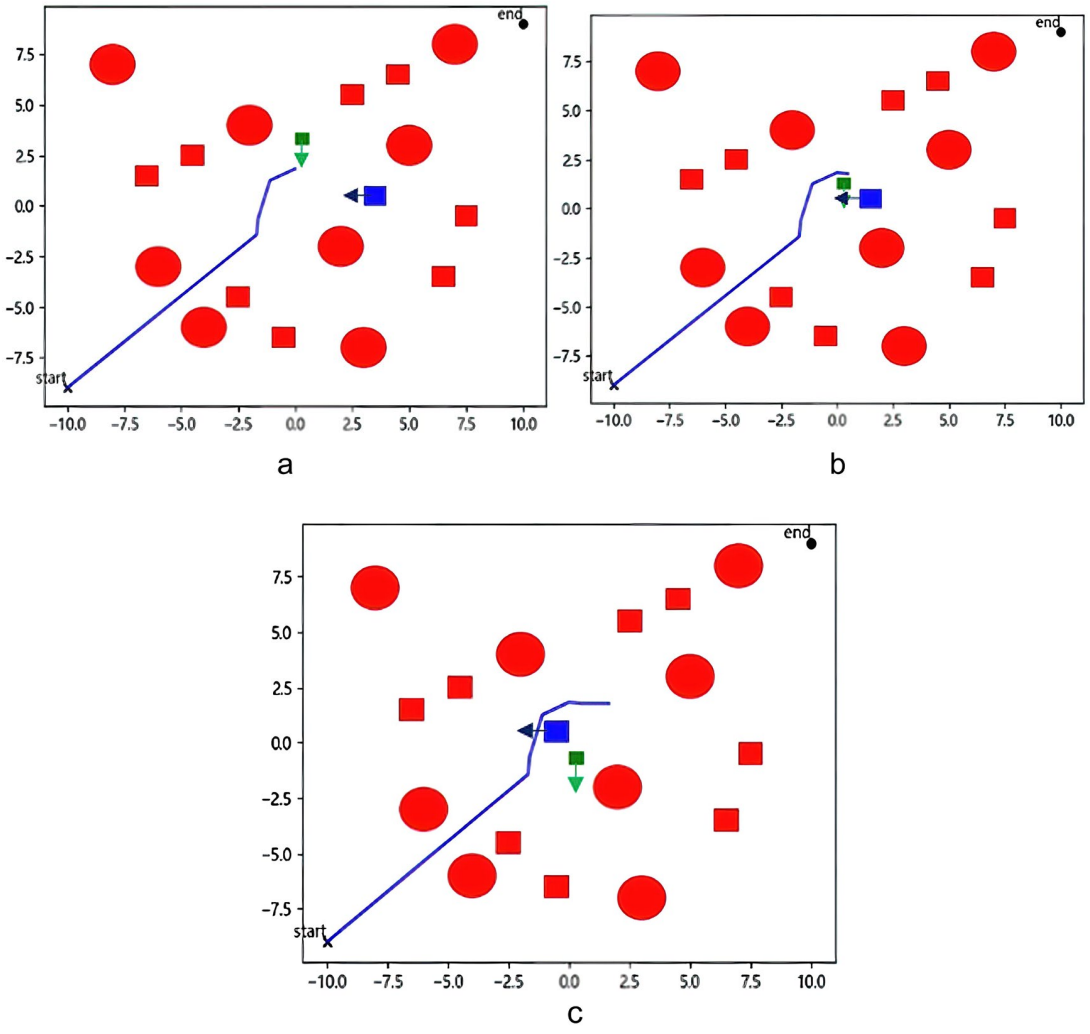


Figure 18. Local dynamic obstacle avoidance effect of DPEOU-SAC in Env.4.

In the simulation experiments in this paper, the agent is a mass by default. Figure 18 shows how the agent avoids dynamic obstacles after training with DPEOU-SAC in Env.4. Figure 18a shows a moving obstacle approaching the agent, Figure 18b shows the agent choosing to move 5 degrees down to the right in order to avoid the obstacle, avoiding the green obstacle, and Figure 18c shows the agent successfully avoiding two moving obstacles.

The trajectory diagrams of the seven algorithms are shown in Figure 19. Figure 19a–f show the static path planning and static obstacle avoidance effects of DDPG, TD3, SAC, OU-SAC, PER-SAC, and DPE-SAC in Env.4, respectively. Figure 19e shows the static path planning and static obstacle avoidance effect of the proposed algorithms in this paper. DPEOU-SAC with static path planning and static obstacle avoidance effects is shown in Env.4.

Seven different algorithms are run in Env.4 to obtain the specific performance index as shown in Table 8. From Table 8, the number of collisions and the number of unsmooth converges to zero can be observed, with the DPEOU-SAC proposed in this paper converging to zero the fastest, in which the number of rounds of collision convergence to zero is 245, and the number of rounds of unsmooth converges to zero is 52, far faster than the other algorithms. When the number of collisions converges to zero and the number of unsmooth converges to zero at the same time, the agent is considered to be able to avoid obstacles normally and plan a feasible path. The traditional DDPG, TD3, and SAC algorithms converge at 716, 719, and 640 rounds, respectively, which are slower than the convergence ability of the DPEOU-SAC proposed in this paper. The improved SACs such as OU-SAC, PER-SAC, and DPE-SAC converge at 487, 432, and 304 rounds, respectively, compared to the original SAC, but the convergence speed is still slower compared to the DPEOU-SAC proposed in this paper.

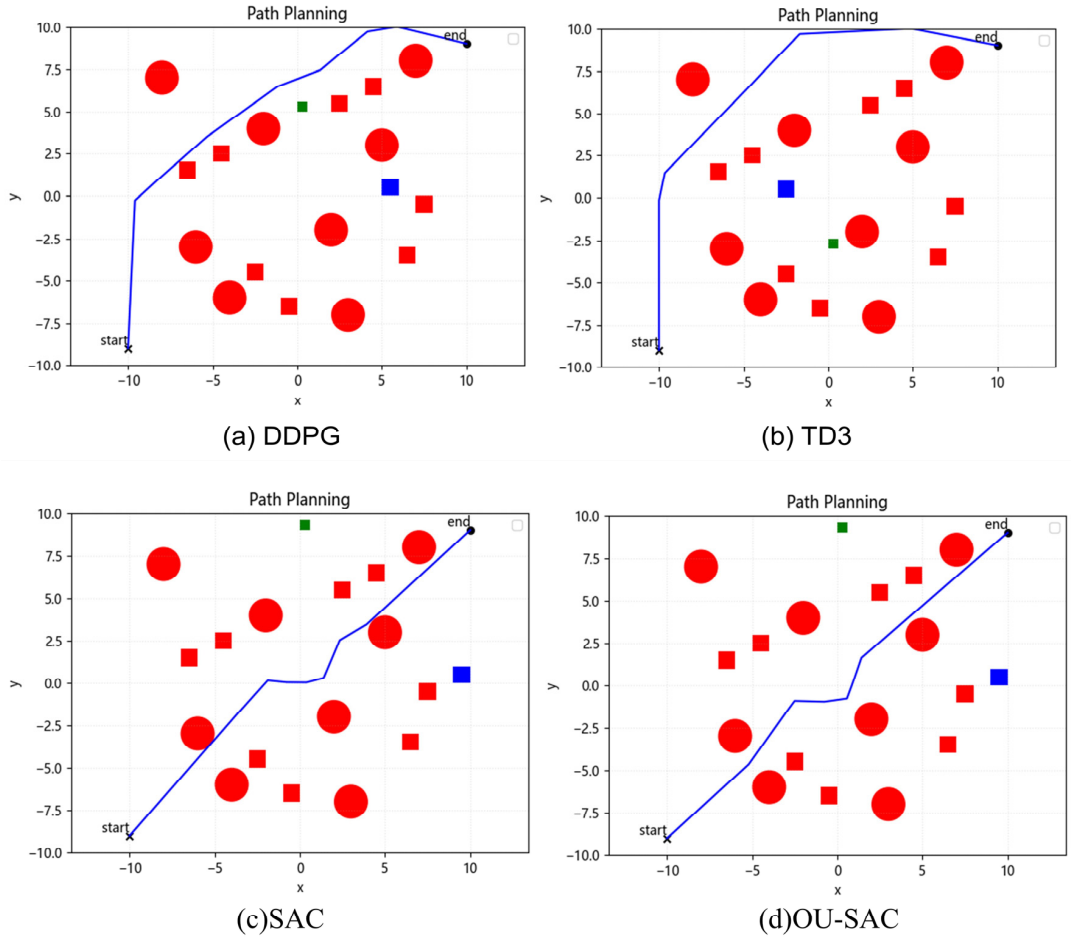
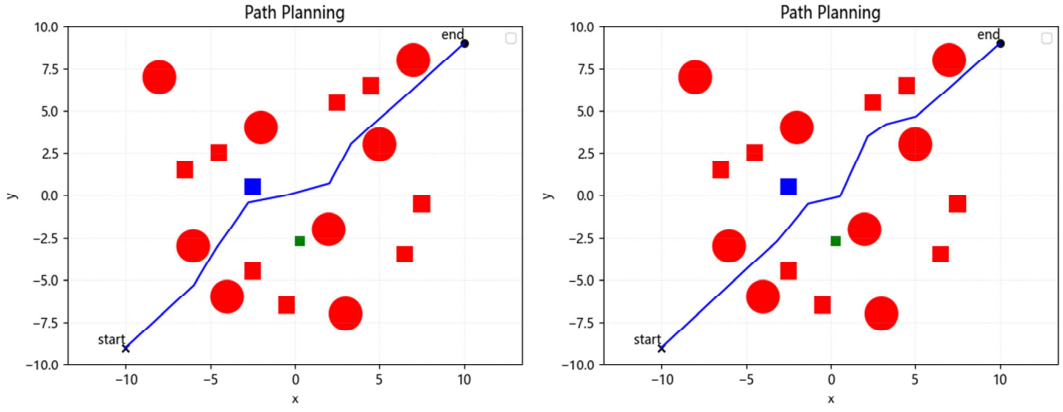
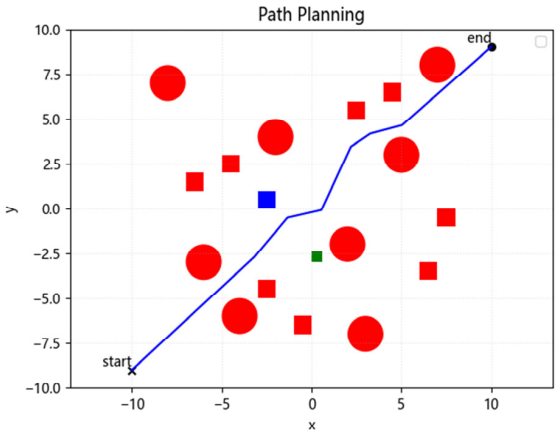


Figure 19. *Cont.*



(e)PER-SAC (f)DPE-SAC



(g)DPEOU-SAC

Figure 19. Different algorithms' trajectories in Env.4.

Table 8. Different algorithms' results in Env.4.

Algorithm Indicator	DDPG	TD3	SAC	OU-SAC	PER-SAC	DPE-SAC	DPEOU-SAC
collisions converges to 0	716	719	640	376	432	304	245
unsmooth converges to 0	716	426	640	487	432	303	52
length	32.16	33.57	27.98	28.04	27.92	27.89	27.64

The length in Table 8 represents the length of the paths planned by each algorithm in Figure 19, where the path lengths planned by the DDPG, TD3, and SAC algorithms are 35.16, 36.57, and 27.98. The SAC algorithm plans shorter paths than the DDPG and TD3. Improved SACs such as OU-SAC, PER-SAC, DPE-SAC, and DPEOU-SAC plan out path lengths of 28.04, 27.92, 27.89, and 27.64, respectively. It can be observed that the DPEOU-SAC algorithm plans out the shortest paths.

From Table 9, it can be seen that the lifting efficiency of DPEOU-SAC with respect to SAC calculated by Equation (25) is 61.72%, while that of OU-SAC is 23.90%, that of PER-SAC is 32.50%, and that of DPE-SAC is 52.50%. It can be found that the algorithm proposed in this paper has obvious advantages over other improved algorithms.

Table 9. Convergence improvement metrics for the improved SAC algorithm in Env.4.

Algorithm Indicator	OU-SAC	PER-SAC	DPE-SAC	DPEOU-SAC
ipc	23.90%	32.50%	52.50%	61.72%

In Figure 20, it can be observed that DPEOU-SAC converges the earliest and the fluctuation after convergence is small. While DPE-SAC takes a slightly longer time to converge than DPEOU-SAC, but faster than the other algorithms. And the traditional DDPG, SAC, and TD3 take a longer time to converge.

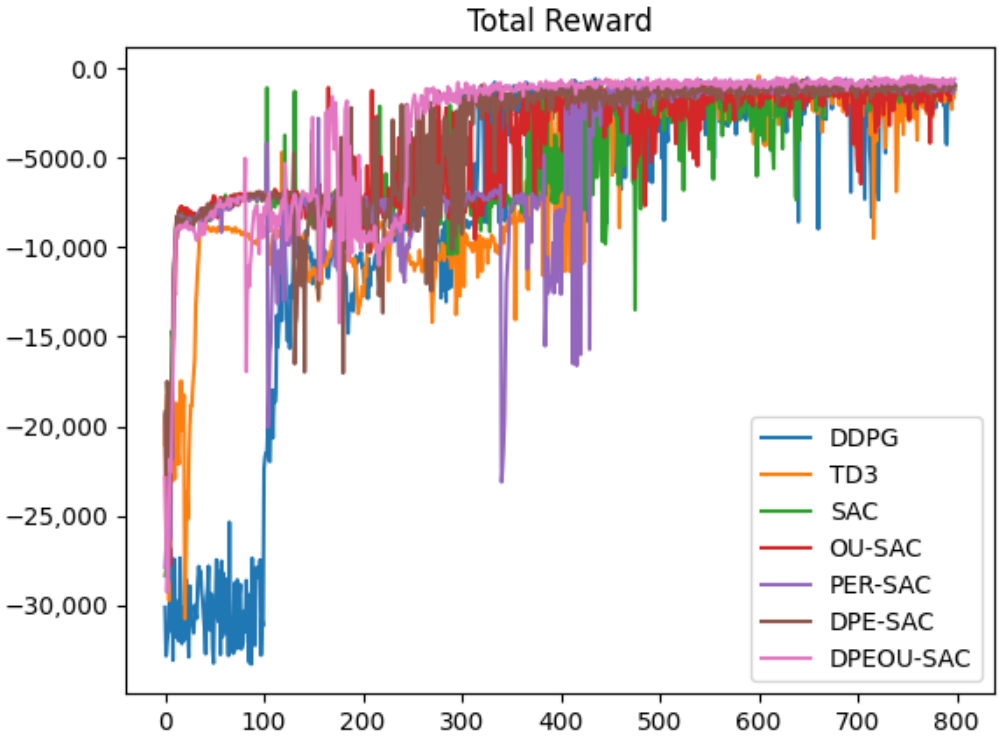


Figure 20. Different algorithms' total rewards in Env.4.

4.3. Impact of Parameter Variations on Algorithm Performance

The factors that have the greatest impact on the performance of the algorithm in DPEOU-SAC are Gamma, Lr_critic, and Lr_actor. Gamma is the γ of Equation (9), and Lr_critic and Lr_actor are the learning rates of the critic neural network and the actor neural network, respectively. In order to simplify the tuning process, this paper chooses to initialize Lr_critic and Lr_actor to the same value, and it is referred to as lr in the following experiments.

In order to verify the specific impact of Gamma and lr on the performance of the algorithm, we adjusted the parameters of the DPEOU-SAC algorithm and tested it in four environments, namely, Env.1, Env.2, Env.3, and Env.4, and discussed the sensitivity of the performance of the DPEOU-SAC algorithm to the parameter changes by observing the convergence speed and the distance of the planned path.

The following experiments are all results of 800 training sessions. The results of the completed tests are shown in Table 10 below.

Table 10. The results of the impact of parameter variations.

Env	Changed Parament	Fixed Parameter	DPEOU-SAC Convergence Round	DPEOU-SAC Length
Env.1	Gamma1 = 0.97	Lr = 1×10^{-3}	54	28.31
	Gamma2 = 0.9		75	27.64
	Gamma3 = 0.8		122	27.94
	Lr1 = 1×10^{-3}	Gamma = 0.95	95	29.32
	Lr2 = 5×10^{-4}		121	27.98
	Lr3 = 1×10^{-4}		170	27.88
	Gamma1 Lr1	None	84	27.66
	Gamma2 Lr2		78	27.83
	Gamma3 Lr3		157	29.73
Env.2	Gamma1 = 0.97	Lr = 1×10^{-3}	277	27.12
	Gamma2 = 0.9		349	28.49
	Gamma3 = 0.8		424	28.61
	Lr1 = 1×10^{-3}	Gamma = 0.95	442	27.84
	Lr2 = 5×10^{-4}		348	29.64
	Lr3 = 1×10^{-4}		398	28.96
	Gamma1 Lr1	None	337	27.82
	Gamma2 Lr2		412	27.29
	Gamma3 Lr3		433	27.56
Env.3	Gamma1 = 0.97	Lr = 1×10^{-3}	83	28.53
	Gamma2 = 0.9		107	27.53
	Gamma3 = 0.8		77	27.57
	Lr1 = 1×10^{-3}	Gamma = 0.95	112	27.44
	Lr2 = 5×10^{-4}		120	27.28
	Lr3 = 1×10^{-4}		144	27.06
	Gamma1 Lr1	None	86	27.55
	Gamma2 Lr2		63	28.25
	Gamma3 Lr3		100	29.62
Env.4	Gamma1 = 0.97	Lr = 1×10^{-3}	205	28.28
	Gamma2 = 0.9		228	27.31
	Gamma3 = 0.8		255	27.35
	Lr1 = 1×10^{-3}	Gamma = 0.95	280	27.51
	Lr2 = 5×10^{-4}		275	27.76
	Lr3 = 1×10^{-4}		312	27.21
	Gamma1 Lr1	None	235	28.22
	Gamma2 Lr2		287	28.30
	Gamma3 Lr3		310	27.53

As can be seen in Table 10, the variation in Gamma and Lr still has an impact on the performance of the algorithm. The overall trend in the four environments is that as Gamma decreases, the number of rounds required for agent convergence increases, indicating that Gamma has a significant impact on the algorithm's consideration of long-term gains, and that lower values of Gamma may result in the algorithm placing more emphasis on short-term rewards. And path planning requires the agent to pay more attention to long-term rewards, so lower Gamma will lead to a decrease in the performance of the algorithm. The effect of Lr is more random, in the case of the same Gamma, there is no situation that the larger the Lr, the better the performance of the algorithm, implying the need to carefully adjust the learning rate for specific tasks and environments. And when Gamma and Lr change at the same time, the performance of the algorithm will also fluctuate. However, all of the above algorithms were able to converge before 800 rounds, proving that the algorithm is robust. It also shows that for path planning using deep reinforcement learning algorithms, parameter variations cause fluctuations in the performance of the algorithm, and appropriate parameters can improve the performance of the algorithm.

5. Conclusions and Future Work

With the development of mobile robots, there is an increasing demand for mobile robot path-planning algorithms. In this paper, a mobile robot path planning method based on the improved SAC algorithm is proposed. The method optimizes the agent's exploration ability by combining OU noise; the two-factor-based priority sampling experience pool technique greatly improves the agent's sampling efficiency and learning efficiency; and the reward mechanism purposefully solves the reward sparsity problem. The algorithms are designed in the gym environment and applied to the environment for simulation experiments to prove the effectiveness of the algorithms and test the advantages and disadvantages of the algorithms.

In the simulation Env.1, the simulation results show that the OU-SAC algorithm improves the efficiency by 53.03% on the basis of SAC, the PER-SAC algorithm improves the efficiency by 74.83% on the basis of SAC, the DPE-SAC method improves the efficiency by 79.10% on the basis of SAC, and the DPEOU-SAC improves the efficiency by 88.99% on the basis of SAC. Simulation results in Env.2 show that the OU-SAC algorithm improves 15.05% efficiency on top of SAC, the PER-SAC algorithm improves 61.20% efficiency on top of SAC, the DPE-SAC method improves 70.09% efficiency on top of SAC, and the DPEOU-SAC improves 80.27% efficiency. Simulation results in Env.3 show that the OU-SAC algorithm improves 17.79% efficiency on top of SAC, the PER-SAC algorithm improves 40.64% efficiency on top of SAC, the DPE-SAC method improves 43.82% efficiency on top of SAC, and the DPEOU-SAC improves 44.38% efficiency. Simulation results in Env.4 show that the OU-SAC algorithm improves 23.90% efficiency on top of SAC, the PER-SAC algorithm improves 32.50% on top of SAC, the DPE-SAC method improves 52.50% on top of SAC, and the DPEOU-SAC improves 61.72% efficiency. When both static and dynamic characteristics of the algorithms are considered, the proposed DPEOU-SAC outperforms the proposed DDPG and six heuristics such as TD3, SAC, OU-SAC, PER-SAC, and DPEOU-SAC in the optimal path planning problem. It also verifies the sensitivity of the performance of the DPEOU-SAC algorithm to parameter variations and demonstrates that DPEOU-SAC is robust enough to cope with parameter variations. It is also shown that appropriate parameter settings of deep reinforcement learning algorithms can improve the performance of the algorithms.

The experimental results show that the proposed DPEOU-SAC not only solves the shortcomings of the standard SAC but also plans the globally optimal paths, and can smooth the paths in real time and avoid dynamic obstacles in real time. The proposed DPEOU-SAC is validated only on the simulation of the optimal path problem by a mobile robot.

The above work is part of the research that lays the foundation for work on reinforcement learning-based path planning for mobile robots. However, the above work is based on a simulation and future work will apply the simulated algorithms to real robots to observe the effect.

Author Contributions: J.C. (Jionghui Chen) was responsible for conceptualizing the study concept and design, obtaining data, analyzing and interpreting the data, and writing the manuscript. L.L. assisted in conceptualizing the study concept and design ideas and obtaining funding to help revise the manuscript. Y.Z., J.L., and J.C. (Jiayu Chen) assisted in analyzing and interpreting the data and writing the manuscript. D.H. provided ideas and funding. All authors have read and agreed to the published version of the manuscript.

Funding: The research in this paper was funded by the Fujian University Industry–University-Research Joint Innovation Project (2022H6005) and the Fujian University Industry–University Cooperation Science and Technology Program (2022N5020).

Data Availability Statement: Data is contained within the article.

Conflicts of Interest: We declare no conflicts of interest.

References

1. Wu, M.; Zhu, S.; Li, C.; Zhu, J.; Chen, Y.; Liu, X.; Liu, R. UAV-Mounted RIS-Aided Mobile Edge Computing System: A DDQN-Based Optimization Approach. *Drones* **2024**, *8*, 184. Available online: https://www.mdpi.com/2504-446X/8/5/184 (accessed on 11 May 2024). [CrossRef]
2. Alonso-Mora, J.; Baker, S.; Rus, D. Multi-robot formation control and object transport in dynamic environments via constrained optimization. *Int. J. Robot. Res.* **2017**, *36*, 1000–1021. [CrossRef]
3. Su, Z.; Li, M.; Zhang, G.; Yue, F. Modeling and solving the repair crew scheduling for the damaged road networks based on Q-learning. *Acta Autom. Sin.* **2020**, *46*, 1467–1478.
4. Zeng, C.; Yang, C.; Li, Q.; Dai, S. Research progress on human-robot skill transfer. *Acta Autom. Sin.* **2019**, *45*, 1813–1828.
5. 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: Cham, Switzerland, 2015; pp. 3–27.
6. Raja, P.; Pugazhenthi, S. Optimal path planning of mobile robots: A review. *Int. J. Phys. Sci.* **2012**, *7*, 1314–1320. [CrossRef]
7. Li, Q.; Geng, X. Research on Path Planning Method Based on Improved DQN Algorithm. *Comput. Eng.* **2023**, 1–11. [CrossRef]
8. Chong, W.; Yi, C.; Shu-Feng, W.; Chaohai, K. Heuristic dynamic path planning algorithm based on SALSTM-DDPG. *J. Phys. Conf. Ser.* **2023**, *2593*, 012008. [CrossRef]
9. Yang, L.; Bi, J.; Yuan, H. Intelligent Path Planning for Mobile Robots Based on SAC Algorithm. *J. Syst. Simul.* **2023**, *35*, 1726–1736.
10. Chen, P.; Pei, J.; Lu, W.; Li, M. A deep reinforcement learning based method for real-time path planning and dynamic obstacle avoidance. *Neurocomputing* **2022**, *497*, 64–75. [CrossRef]
11. Zhang, Y.; Chen, P. Path Planning of a Mobile Robot for a Dynamic Indoor Environment Based on an SAC-LSTM Algorithm. *Sensors* **2023**, *23*, 9802. Available online: https://www.mdpi.com/1424-8220/23/24/9802 (accessed on 21 January 2024). [CrossRef]
12. Wang, C.; Ross, K. Boosting soft actor-critic: Emphasizing recent experience without forgetting the past. *arXiv* **2019**, arXiv:1906.04009.
13. Peixoto, M.J.; Azim, A. Using time-correlated noise to encourage exploration and improve autonomous agents performance in Reinforcement Learning. *Procedia Comput. Sci.* **2021**, *191*, 85–92. [CrossRef]
14. Josef, S.; Degani, A. Deep Reinforcement Learning for Safe Local Planning of a Ground Vehicle in Unknown Rough Terrain. *IEEE Robot. Autom. Lett.* **2020**, *5*, 6748–6755. [CrossRef]
15. Liu, Y.; He, Q.; Wang, J.; Chen, T.; Jin, S.; Zhang, C.; Wang, Z. Convolutional Neural Network Based Unmanned Ground Vehicle Control via Deep Reinforcement Learning. In Proceedings of the 2022 4th International Conference on Control and Robotics (ICCR), Guangzhou, China, 2–4 December 2022; pp. 1–6. [CrossRef]
16. Chen, X.; Qi, Y.; Yin, Y.; Chen, Y.; Liu, L.; Chen, H. A Multi-Stage Deep Reinforcement Learning with Search-Based Optimization for Air–Ground Unmanned System Navigation. *Appl. Sci.* **2023**, *13*, 2244. Available online: https://www.mdpi.com/2076-3417/13/4/2244 (accessed on 20 January 2024). [CrossRef]
17. Lozano-Pérez, T.; Wesley, M.A. An algorithm for planning collision-free paths among polyhedral obstacles. *Commun. ACM* **1979**, *22*, 560–570. [CrossRef]
18. Gao, P.; Liu, Z.; Wu, Z.; Wang, D. A global path planning algorithm for robots using reinforcement learning. In Proceedings of the 2019 IEEE International Conference on Robotics and Biomimetics (ROBIO), Dali, China, 6–8 December 2019; IEEE: Piscataway, NJ, USA, 2019; pp. 1693–1698.
19. Xu, X.; Zeng, J.; Zhao, Y.; Lü, X. Research on global path planning algorithm for mobile robots based on improved A. *Expert Syst. Appl.* **2023**, *243*, 122922. [CrossRef]
20. Zong, C.; Han, X.; Zhang, D.; Liu, Y.; Zhao, W.; Sun, M. Research on local path planning based on improved RRT algorithm. *Proc. Inst. Mech. Eng. Part D J. Automob. Eng.* **2021**, *235*, 2086–2100. [CrossRef]
21. Szczepanski, R.; Tarczewski, T.; Erwinski, K. Energy efficient local path planning algorithm based on predictive artificial potential field. *IEEE Access* **2022**, *10*, 39729–39742. [CrossRef]
22. Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A.A.; Veness, J.; Bellemare, M.G.; Graves, A.; Riedmiller, M.; Fidjeland, A.K.; Ostrovski, G.; et al. Human-level control through deep reinforcement learning. *Nature* **2015**, *518*, 529–533. [CrossRef]
23. Wu, M.; Guo, K.; Li, X.; Lin, Z.; Wu, Y.; Tsiftsis, T.A.; Song, H. Deep Reinforcement Learning-based Energy Efficiency Optimization for RIS-aided Integrated Satellite-Aerial-Terrestrial Relay Networks. *IEEE Trans. Commun.* **2024**, 1. [CrossRef]
24. Sutton, R.S.; Barto, A.G. *Reinforcement Learning: An Introduction*; MIT Press: Cambridge, MA, USA, 2018.
25. Tang, J.; Liang, Y.; Li, K. Dynamic Scene Path Planning of UAVs Based on Deep Reinforcement Learning. *Drones* **2024**, *8*, 60. Available online: https://www.mdpi.com/2504-446X/8/2/60 (accessed on 11 May 2024). [CrossRef]
26. Haarnoja, T.; Zhou, A.; Abbeel, P.; Levine, S. Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor. In Proceedings of the 35th International Conference on Machine Learning, Proceedings of Machine Learning Research, Stockholm, Sweden, 10–15 July 2018; Available online: https://proceedings.mlr.press/v80/haarnoja18b.html (accessed on 12 July 2023).
27. Hessel, M.; Modayil, J.; Van Hasselt, H.; Schaul, T.; Ostrovski, G.; Dabney, W.; Horgan, D.; Piot, B.; Azar, M.; Silver, D. Rainbow: Combining improvements in deep reinforcement learning. In Proceedings of the AAAI conference on Artificial Intelligence, New Orleans, LA, USA, 2–7 February 2018; Volume 32.
28. Christodoulou, P. Soft actor-critic for discrete action settings. *arXiv* **2019**, arXiv:1910.07207.

29. Banerjee, C.; Chen, Z.; Noman, N. Improved soft actor-critic: Mixing prioritized off-policy samples with on-policy experiences. *IEEE Trans. Neural Netw. Learn. Syst.* **2022**, *35*, 3121–3129. [CrossRef] [PubMed]
30. Jiang, L.; Huang, H.; Ding, Z. Path planning for intelligent robots based on deep Q-learning with experience replay and heuristic knowledge. *IEEE/CAA J. Autom. Sin.* **2020**, *7*, 1179–1189. [CrossRef]
31. Zhang, G.; Wu, Q.; Cui, M.; Zhang, R. Securing UAV communications via joint trajectory and power control. *IEEE Trans. Wirel. Commun.* **2019**, *18*, 1376–1389. [CrossRef]