Local Path Planning of Mobile Robots Based on the Improved SAC Algorithm

Ruihong Zhou[1], Caihong Li[2]*, Guosheng Zhang[3], Yaoyu Zhang[4], Jiajun Liu[5]
School of Computer Science and Technology, Shandong University of Technology, Zibo 255049, China[1, 2, 3, 4]
Faculty of Business, Lingnan University, Hong Kong[5]

Abstract—This paper proposes a new EP-PER-SAC algorithm to solve the problems of slow training speed and low learning efficiency of the SAC (Soft Actor Critic) algorithm in the local path planning of mobile robots by introducing the Priority Experience Replay (PER) strategy and Experience Pool (EP) adjustment technique. This algorithm replaces equal probability random sampling with sampling based on the priority experience to increase the frequency of extracting important samples, thereby improves the stability and convergence speed of model training. On this basis, it requires to continuously monitor the learning progress and exploration rate changes of the robot to dynamically adjust the experience pool, so the robot can adapt effectively to the environment changes and the storage requirements and learning efficiency of the algorithm are balanced. Then, the algorithm's reward and punishment function is improved to reduce the blindness of algorithm training. Finally, experiments are conducted under different obstacle environments to verify the feasibility of the algorithm based on ROS (Robot Operating System) simulation platform and real environment. The results show that the improved EP-PER-SAC algorithm has a shorter path length and faster model convergence speed than the original SAC algorithm and PER-SAC algorithm.

Keywords—Mobile robots; local path planning; reinforcement learning; SAC algorithm; priority experience replay; experience pool adjustment; Robot Operating System (ROS)

I. INTRODUCTION

The ability of mobile robots to plan their paths is a critical task in the field of robotics. The robot explores an optimal or sub-optimal safe path from the starting point to the end point in workplace according to the given requirements [1]. The common traditional path planning algorithms mainly include A* algorithm [2-3], Artificial Potential Field method [4-5], Dijkstra algorithm [6], Genetic algorithm [7], Fuzzy Control algorithm [8], and Ant Colony algorithm [9-10]. These algorithms rely on maps and environmental models during the path planning process and are prone to falling into local minima when dealing with complex environments. With the development of computer science and artificial intelligence, intelligent algorithms have received widespread attention due to vast database and powerful computing capability to perform various tasks. Reinforcement Learning algorithm [11-12] is a typical example. It learns the optimal policy through interaction with the environment, thus can overcome the difficulties associated with map modeling. The Deep Reinforcement Learning algorithm [13-14] further combine Deep Learning [15] with Reinforcement Learning. It has the ability to learn and make decisions in complex environments, and has also achieved remarkable results of dealing with the path planning of mobile robots.

Typical algorithms of Deep Reinforcement Learning include DDPG (Deep Deterministic Policy Gradient) algorithm [16-17], TD3 (Twin Delayed Deep Deterministic policy gradient) [18] and SAC algorithm. Silver et al. proposed the DPG (Deterministic Policy Gradient) algorithm, which updated the value function to address Reinforcement Learning problems in a continuous action space [19]. Lillicrap et al. extended the DPG algorithm by incorporating the principles of Deep Q-learning and introduced the DDPG algorithm, which can effectively deal with high-dimensional continuous action [20]. Fujimoto et al. improved the DDPG algorithm by employing two separate Q-network to evaluate action values and proposed the TD3 algorithm. This improved method can reduce the overestimation of action values and enhance training stability [21]. Haarnoja et al. introduced the SAC algorithm, which used dual Q-network and incorporated the principle of maximum entropy. It maximized entropy to increase the exploratory capability of the algorithm [22]. Yuxiang Z et al. proposed the SAC algorithm combining with the Artificial Potential Field method. The self attention mechanism had been introduced into the Actor network of the SAC algorithm in response to the high dimensionality and complexity of environmental state in 3D environmental space. It improved the convergence speed and success rate of the algorithm, but the hyper-parameters can also be adjusted to improve the algorithm performance [23].

This paper presents an EP-PER-SAC algorithm to solve the shortcomings of the SAC algorithm, such as long training time and wasted effective experience. The improved algorithm uses the fully connected neural network in which the obstacle information is detected by the ten radar sensors of robot, where the angle and the distance between the robot and the target point are as inputs of the network, the angular velocity and linear velocity of the robot as outputs. Combined with the preferential experience playback mechanism, the samples with high priority are preferentially selected. The experience pool is dynamically adjusted according to the learning progress and changes of the exploration rate, thus balancing the efficiency of exploration and exploitation of samples. A detailed reward and punishment function is designed to enable robots to more easily obtain effective feedback from environmental exploration, which can solve the issue of reward sparsity and enhance the sample utilization rate and learning efficiency of the algorithm.

II. SAC ALGORITHM

The SAC algorithm is a Deep Reinforcement Learning algorithm that maximizes policy entropy [24]. It can be used to address the problem within continuous action space. The SAC algorithm consists of an Actor network, two Critic networks and two Target Critic networks. Fig. 1 depicts the flowchart of SAC algorithm.

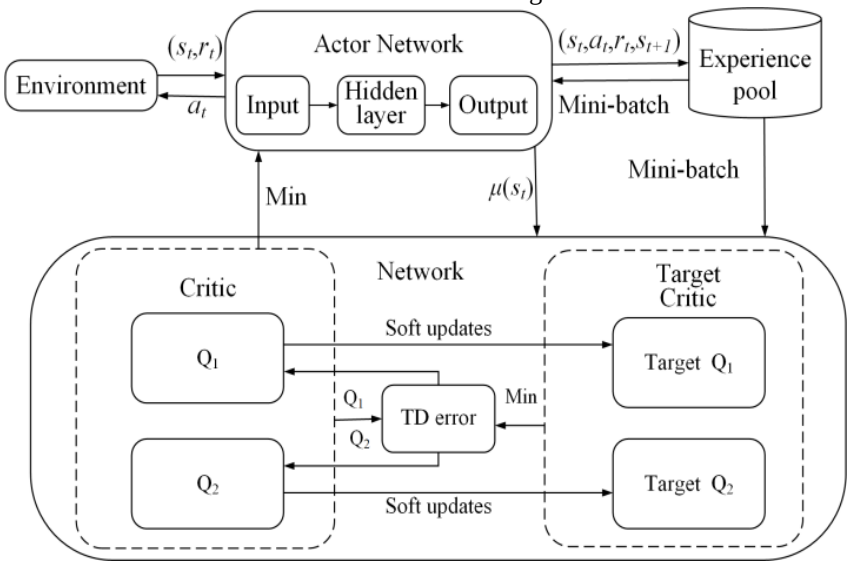


Fig. 1. The flowchart of SAC algorithm.

The SAC algorithm obtains the maximum expected reward value by training effective samples, while satisfying the maximization of entropy value. The algorithm can be represented as:

$$J(\pi) = \sum_{t=0}^{T} E_{(s_t, a_t)\sim\rho_\beta}\left[r(s_t, a_t) + \alpha H(\pi(\cdot \mid s_t))\right] \quad (1)$$

In Eq. (1), E is a reward expectation, (st,at)~ρβ is the state distribution related to strategy, r(st,at) is the return value obtained by executing the action at, α is a parameter that controls entropy regularization, H is the entropy in the state st which can be expressed as:

$$H(\pi(\cdot \mid s_t)) = E_{a_t}[-\log \pi(\cdot \mid s)] \quad (2)$$

The algorithm selects the initial state st, and obtains the action probability π(at|st) after passing through the Actor network. Then, the algorithm obtains the action at according to probability sampling and applies it to the environment to generate a set of empirical tuples (st, at, st+1, rt+1).

The input of the Actor network is the state st, and the output is the action probability π(at|st). The changes in loss during training can be expressed as:

$$L_\pi(\varepsilon) = \frac{1}{|B|} \sum_{(s_t, a_t, s_{t+1}, r_{t+1})} E_{a_t'}[q(s_t, a_t') - \ln \pi_\theta(a_t' \mid s_t)] \quad (3)$$

In Eq. (3), B represents the experience pool, represents the possible actions predicted by the Actor network again.

The Critic networks are used to evaluate the expected return on a given state and action under the current strategy. The Target Critic networks are used to provide stable target value estimates. In order to accurately evaluate the state-action function Q(st,at) (abbreviated as Q-value), the SAC algorithm combines the maximum entropy principle and uses the smaller values output by two Critic networks for estimation. The Q-value is as follows:

$$Q(s_t, a_t) = r(s_t, a_t) + \min_{i=1,2} Q_i(s_{t+1}, a_{t+1}) - \alpha \log(\pi_\phi(a_{t+1}, s_{t+1})) \quad (4)$$

In Eq. (4), α is a temperature parameter used to regulate the importance of entropy.

The loss function of the Critic networks are:

$$L_c(\theta_i) = E_{(s_t, a_t)}[(Q_{\theta_i}(s_t, a_t) - Q(s_t, a_t))] \quad (5)$$

The SAC algorithm uses gradient descent and ascent methods to update the parameters of the strategy and value network, while updating the target networks and clearing the current gradient information to prepare for the next round of training.

III. IMPROVE THE SAC ALGORITHM

This research proposes the EP-PER-SAC algorithm, which uses neural network to predict the state and the action of robot. The improved algorithm extracts samples with higher priority multiple times to increase the utilization of effective samples. It dynamically adjusts the experience pool based on training progress and performance to adapt to different training stages. This approach can balance the need of exploration and utilization, while improving the training effectiveness of the algorithm. The EP-PER-SAC algorithm includes the improvement of state and action space design, network structure, and design of the reward and punishment function.

A. State and Action Space Design

The state space refers to the set including all possible states which may occur in an environment. Inputting this information into the network, the robot execute an action based on the

current state to accumulate more rewards and optimize its strategy.

Radar sensors are mounted on a two wheel differential drive robot to detect the obstacles within a range of 180° in front, which are returned with every 20° a set, for a total of 10 sets. Fig. 2 shows the radar detection structure.

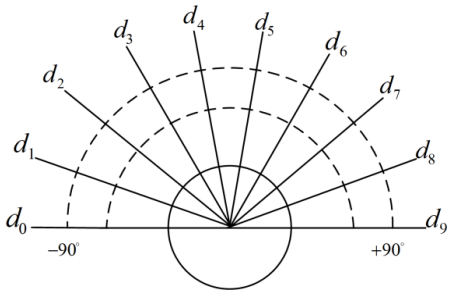


Fig. 2. Radar detection structure.

The robot state space S consists of the distance information to the nearest obstacles from ten direction sensors dk (k=0~9), the angular angle between the robot and the target point θi and the distance between the robot and the target point Di:

$$S = (d_0 \sim d_9, \theta_i, D_i) \tag{6}$$

The action space A is used for exploring and executing various actions within a certain range which includes the robot's angular velocity $\omega_i \in [\omega_{\min}, \omega_{\max}]$ and linear velocity $\nu_i \in [\nu_{\min}, \nu_{\max}]$, where $[\omega_{\min}, \omega_{\max}] \subseteq [-2,2]$ with the unit rad/s, $[\nu_{\min}, \nu_{\max}] \subseteq [0,0.34]$ with the unit m/s. The robot's action space A is defined as:

$$A = (\omega_i, \nu_i) \tag{7}$$

B. Priority Experience Replay

The SAC algorithm uses a random sampling method during sampling, which cannot ensure repeated sampling of important samples. Priority Experience Replay technology assigns different priorities to each sample based on the TD error value, and samples with higher priorities are more likely to be extracted. The framework for prioritizing experience replay is shown in Fig. 3.

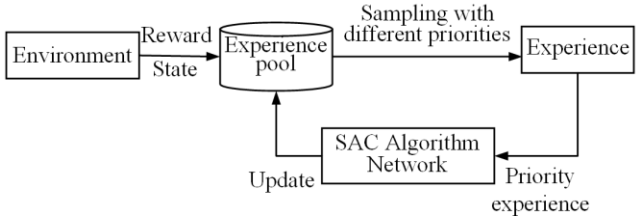


Fig. 3. Priority experience replay framework.

The TD error of samples is commonly used to measure the discrepancy between the actual value and the predicted value. When the TD error value is large, it may indicate that the training performance at that particular state is poor. The learning efficiency can be improved by increasing the probability of samples with large errors being extracted and training them for multiple times. TD error δ_i is defined as:

$$\delta_i = r(s_t, a_t) + \max_{a_{t+1}} \gamma Q_{\text{target}}(s_{t+1}, a_{t+1}) - Q(s_t, a_t) \tag{8}$$

In Eq. (8), γ represents the discount factor, $Q(s_t,a_t)$ and $Q_{\text{target}}(s_{t+1},a_{t+1})$ represent the states value of the critic networks and the target critic networks, respectively.

The probability of extracting samples from the experience pool can be expressed as:

$$p_i = \frac{\delta_i^{\alpha}}{\sum_{i=1}^{n} \delta_i^{\alpha}} \tag{9}$$

In Eq. (9), α parameter is used to control the sample priority.

During the calculation of TD error in the SAC algorithm, the impacts of Critic networks, Target Critic networks and Actor network on the algorithm are different. The addition of balance parameters to the algorithm can improve the influence of the strategy network on the total error. The TD comprehensive error with balance parameters η, σ and φ is:

$$\delta_i = |\eta \cdot TD(Q)| + |\sigma \cdot TD(Q_{\text{target}})| + |\varphi \cdot TD(\pi)| \tag{10}$$

C. Dynamic Adjustment of Experience Pool

The experience pool is used to store data generated by the interaction between robot and the environment. The EP-PER-SAC algorithm dynamically adjusts the capacity of the experience pool according to the progress of training and the change of exploration rate. This improved method can optimize the efficiency and quality of sample utilization, and save memory resource. The adjustment framework of experience pool is shown in Fig. 4.

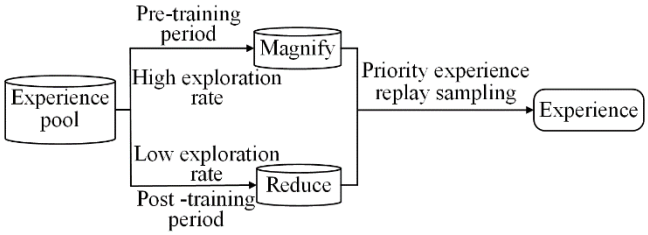


Fig. 4. The adjustment framework of experience pool.

During the training process, the change of the robot's strategy leads to a variation of the data distribution in the experience pool. The algorithm dynamically adjusts the experience pool can ensure that the data in it matches the current strategy more closely, and can improve the stability of training. In the early stage of training, the exploration rate is higher than the exploration utilization rate, and increasing the experience pool can quickly accumulate experience. In the later stage of training, the exploration rate decreases, and the robot needs more refined optimization. Reducing the experience pool can avoid data over-fitting and improve the effectiveness of the algorithm.

D. Network Structure

The network input of EP-PER-SAC algorithm includes obstacle detection data d_k from ten directions of the radar,

angle θ_i and distance D_i between the robot and the target point. The network output of the algorithm includes the angular velocity ω_i and the linear velocity v_i of the robot. Fig. 5 shows the network input and output of the EP-PER-SAC algorithm.

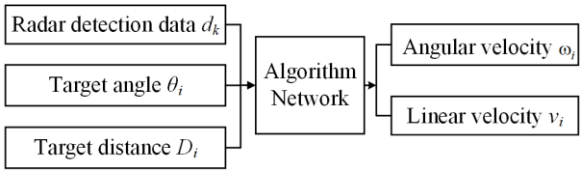


Fig. 5. Network input and output of the EP-PER-SAC algorithm.

Fig. 6 shows the network structure of the algorithm. The algorithm inputs action and state information into the network. The hidden layer consists of three fully connected layers, and each fully connected layer contains 512 neuronal nodes. The activation function in the algorithm network is used to perform nonlinear transformation to improve the learning ability. Then, the algorithm samples to obtain specific actions in the continuous action space. Finally, the network maps these action values to angular velocity ω_i and linear velocity v_i, and sends them to the robot.

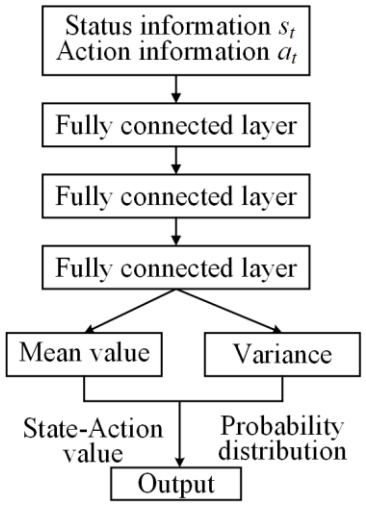


Fig. 6. The network structure of EP-PER-SAC algorithm.

E. Design of Reward and Punishment Functions

The reward and punishment function guides robot to perform appropriate actions in the environment to achieve specific goals which play a crucial role in the success of the algorithm.

In the design of the reward and punishment function, the reward and punishment R_1 shows the distance between the robot and the environment. It controls the distance between the robot and the obstacle environment, and rewards robots to approach the target or avoid obstacles for completing path planning quickly and accurately. The reward and punishment R_2 shows the angle between the robot and the target point. It adjusts the angle between the robot and the target direction, encourages the robot to choose the path with the minimum angle, reduces unnecessary movement, and enhances the target orientation property. The total reward R consists of reward R_1 and reward R_2:

$$R_1 = \begin{cases} r_g, & d_i < c_d \\ r_c, & \min_k < c_o \end{cases}$$

$$R_2 = \begin{cases} C, & -\frac{1}{4}\pi \le \theta \le \frac{1}{4}\pi \\ -C, & \theta < -\frac{1}{4}\pi \quad or \quad \theta > \frac{1}{4}\pi \end{cases}$$

$$R = R_1 + R_2 \tag{11}$$

Where,

r_g is the reward value of the robot to reach the target,

r_c is the penalty value of the robot to collide with obstacles,

d_i is the distance from the robot to the target point at this time,

c_d is the minimum range threshold for reaching the target,

$\min_k$ is the minimum value detected by radar,

c_o is the minimum safe distance from the obstacle environment,

C is a positive integer,

θ is the angle value between the robot and the target point.

IV. Experimental Test and Result Analysis

The improved algorithm is first tested in a simulation environment under ROS platform. After the algorithm converges, it is loaded into the robot for real environment experiments. This research uses Gazebo on the ROS platform to build the robot running environments of obstacle-free, discrete obstacles, 1-shaped obstacle, U-shaped obstacle, and mixed obstacles, respectively. The feasibility of the designed EP-PER-SAC algorithm is verified by model training, and compared with the original SAC algorithm and PER-SAC algorithm (the SAC algorithm combined with Preferential Empirical Replay). The path planned is projected to Rviz in which the blue square represents the target point, gray circles denotes obstacles, and green line means the path trajectory. The parameters of the experimental model are shown in Table I.

TABLE I. Experimental Model Parameters

Parameter	Initial value
Attenuation degree factor	0.99
Maximum number of steps per training round	1000
Number of samples per round	256
Strategy network learning rate	0.0003
Q-network learning rate	0.0003

A. Obstacle-free Environment Simulation

Fig. 7 shows the obstacle-free simulation model of 4m×4m built in Gazebo, with the starting point of the robot set to (-1,0) and the target point set to (1,0).

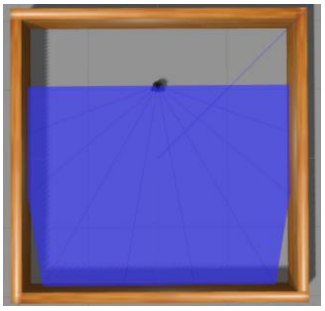


Fig. 7. Obstacle-free environment in Gazebo.

Fig. 8 shows the results of 300 rounds of simulation training on three different algorithms in Gazebo, and records the average reward and punishment return value for each round of the training. The horizontal axis represents the number of training rounds, and the vertical axis denotes the average reward for each round. The results in the graph shows that the average return value of the EP-PER-SAC algorithm significantly increases after 35 rounds, with a faster convergence speed than the original SAC algorithm and PER-SAC algorithm, and tends to stabilize after 150 rounds.

The paths planned by the converged models of the three algorithms in Rviz are drawn in Fig. 9 (a), (b), and (c), respectively. The EP-PER-SAC algorithm has 90 steps in the path, 94 steps in the PER-SAC algorithm, and 95 steps in the original SAC algorithm. The improved algorithm has a slightly shorter path than the other two algorithms.

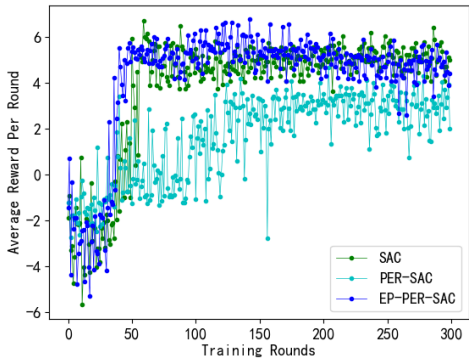
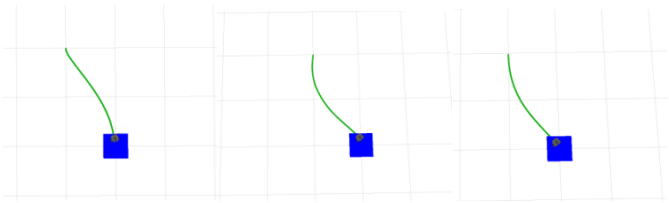


Fig. 8. Comparison of the average reward in the obstacle-free environment of the three methods.

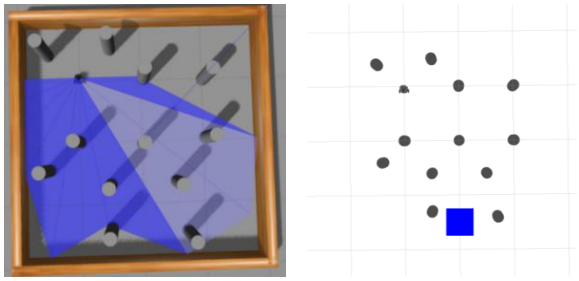


(a) EP-PER-SAC (b) PER-SAC (c) SAC

Fig. 9. Path planning of the three algorithms in an obstacle-free environment.

B. Discrete Obstacles Environment Simulation

Fig. 10(a) and (b) shows the Gazebo discrete obstacles environment and the Rviz projection, respectively.



(a) Gazebo environment (b) Rviz projection

Fig. 10. Discrete obstacles simulation environment.

In this test, the three algorithms are trained for 800 rounds, respectively, and the average reward and punishment return value of each round is shown in Fig. 11. The EP-PER-SAC algorithm has a higher return value than the original SAC algorithm after 150 rounds. While compared to the PER-SAC algorithm, the average return of the EP-PER-SAC algorithm fluctuates less per round, and the probability of the robot reaching the target point is higher.

The path planning results of the three algorithms in the discrete obstacles environment are shown in Fig. 12(a), (b) and (c), respectively. The starting position of the robot is (-1, -1), and the target point is (1,1). The EP-PER-SAC algorithm takes 102 steps to plan the path, the PER-SAC algorithm is 115 steps, and the SAC algorithm owns 123 steps. The improved algorithm has a smoother path and shorter length.

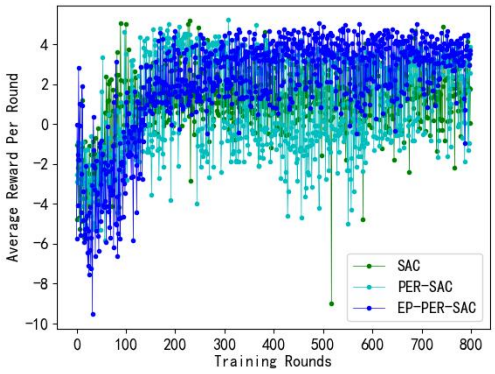
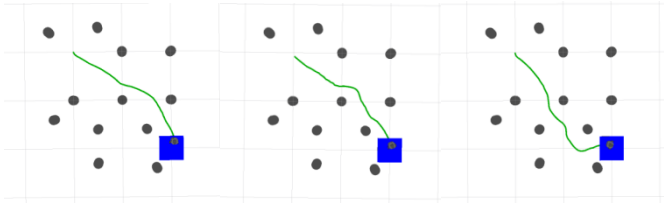


Fig. 11. Comparison of the average reward in the discrete obstacles environment of the three methods.

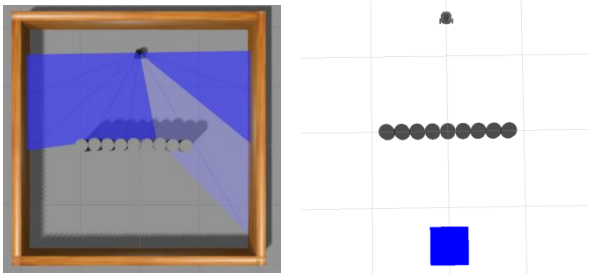


(a) EP-PER-SAC (b) PER-SAC (c) SAC

Fig. 12. Path planning of the three algorithms in a discrete obstacles environment.

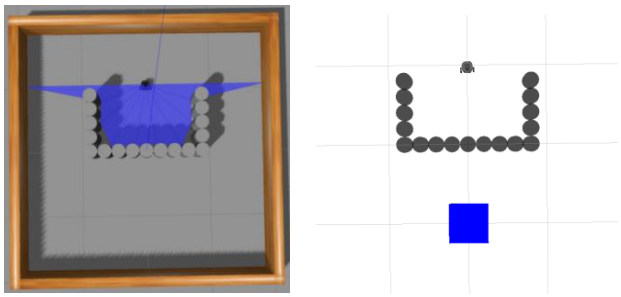
C. Special Obstacles Environment Simulation

Fig. 13 and 14 represent two special obstacles environment, 1-shaped and U-shaped, respectively. The starting point in the 1-shaped environment is (-1.5,0), and the target point is (1.5,0). In the U-shaped environment, the starting point is set to (-1,0) and the ending point is set to (1,0).



(a) Gazebo environment (b) Rviz projection

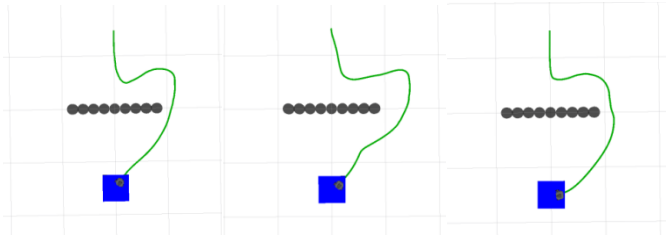
Fig. 13. The 1-shaped obstacle simulation environment.



(a) Gazebo environment (b) Rviz projection

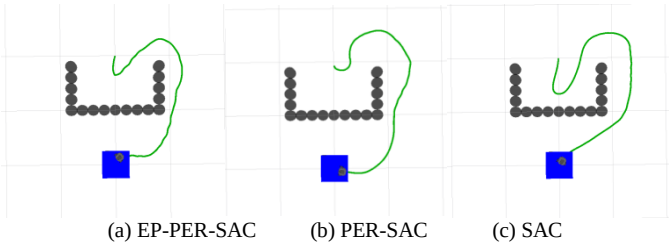
Fig. 14. The U-shaped obstacle simulation environment.

Fig. 15 and 16 show the planned paths of the three algorithms in the 1-shaped and U-shaped environment, respectively. In the 1-shaped obstacle environment, the EP-PER-SAC algorithm, the PER-SAC algorithm and the SAC algorithm take 138 steps, 142 steps and 146 steps, respectively. While in the U-shaped environment, the planned path of the three algorithms owns 131 steps, 140 steps and 143 steps, respectively. The improved algorithm can navigate around the special obstacles environment faster and the planned path length is shorter in both environments.



(a) EP-PER-SAC (b) PER-SAC (c) SAC

Fig. 15. Path planning of the three algorithms in the 1-shaped obstacles environment.

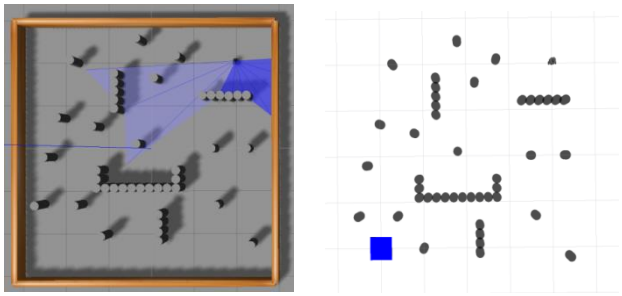


(a) EP-PER-SAC (b) PER-SAC (c) SAC

Fig. 16. Path planning of the three algorithms in the U-shaped obstacles environment.

D. Simulation of Mixed Obstacles Environment

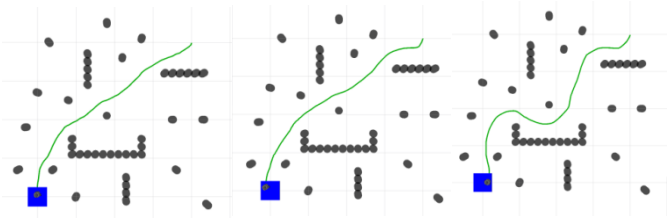
Fig. 17 represents a mixed obstacles environment of 6m×6m in Gazebo. The environment consists of discrete obstacles, 1-shaped and U-shaped obstacles, with the starting point set to (-2, 2), and the target point being (2,-2).



(a) Gazebo environment (b) Rviz projection

Fig. 17. The mixed obstacles environment 1.

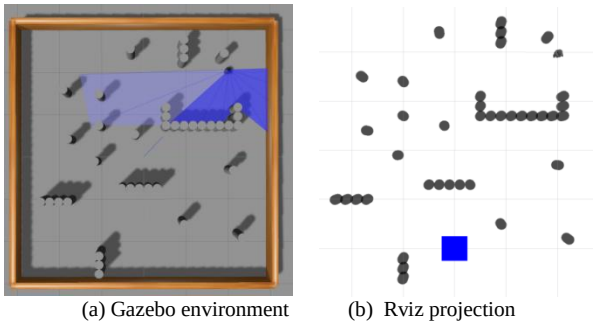
Fig. 18 represents the planned paths of the three algorithms in the mixed obstacles environment 1. Compared with the original SAC algorithm and the PER-SAC algorithm, the EP-PER-SAC algorithm has a shorter path length and it is easier to pass through complex obstacles, which has a significant effect. The planning steps of the EP-PER-SAC algorithm is 194 steps, while PER-SAC algorithm is 217 steps and the original algorithm is 244 steps.



(a) EP-PER-SAC (b) PER-SAC (c) SAC

Fig. 18. Path planning of the three algorithms in the mixed obstacles environment 1.

In order to further verify the feasibility and universality of the convergence algorithm, the algorithm is tested again in a mixed obstacles environment 2, as shown in Fig. 19. The starting point is set to (-2,2), and the target point is set to (2,0)

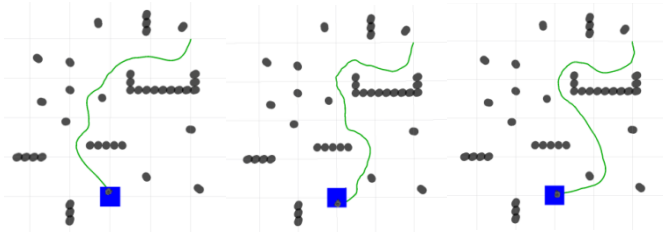


(a) Gazebo environment (b) Rviz projection

Fig. 19. The mixed obstacles environment 2.

The converged model is loaded into the mixed obstacles environment 2, and the path results planned by the three algorithms are shown in Fig. 20. The path steps planned by the EP-PER-SAC algorithm is 199 steps, while the other two

algorithms use 238 and 286 steps by the PER-SAC algorithm and SAC algorithm, respectively. The EP-PER-SAC algorithm still has a shorter path length, which can better avoid obstacles and verify the effectiveness of the algorithm.



(a) EP-PER-SAC (b) PER-SAC (c) SAC

Fig. 20. Path planning of the three algorithms in the mixed obstacles environment 2.

E. Comparison of Experimental Results

Through the experimental results in the aforementioned simulation environments, the EP-PER-SAC algorithm is compared with the SAC algorithm and the PER-SAC algorithm. The feasibility and performance advantages of the designed EP-PER-SAC algorithm have been verified. In different simulation environments, the paths planned by the EP-PER-SAC algorithm are smoother and converge faster than those planned by the SAC algorithm and the PER-SAC algorithm.

The number of steps taken by the three algorithms in different simulation environments are compared in the bar chart, as shown in Fig. 21. From the graph, it can be seen more clearly that the EP-PER-SAC algorithm has fewer steps than the SAC algorithm and the PER-SAC algorithm in any obstacles environment.

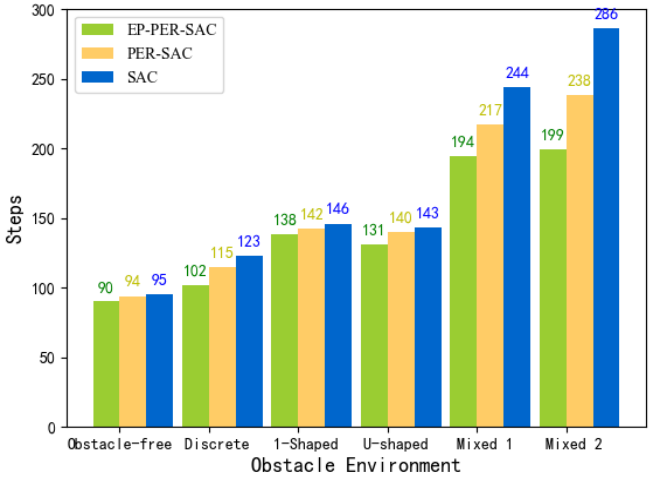
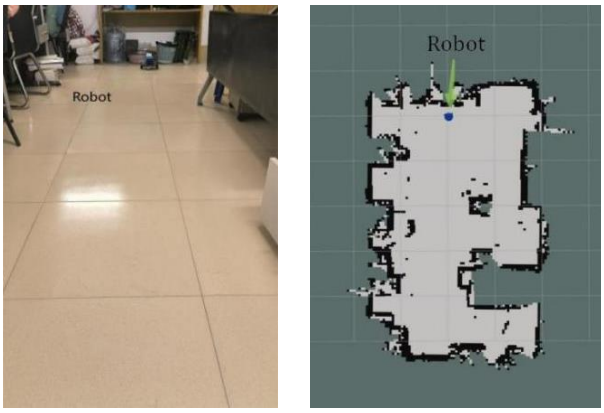


Fig. 21. Comparison of path lengths of the three algorithms in different running environments.

F. Experiments in Real Environment

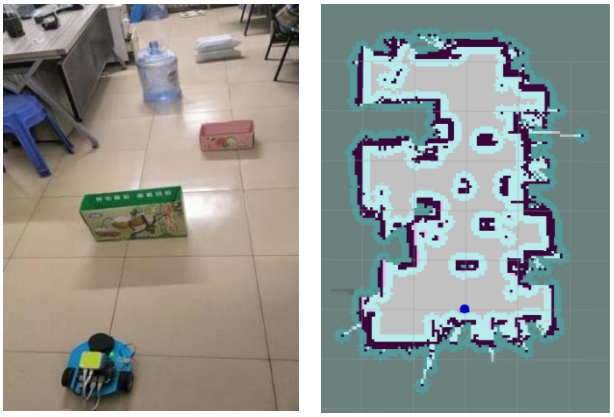
This research applies three algorithms to mobile robot based on the ROS platform, and performs path planning tasks in a real environment to verify the performance of the improved algorithm. The Gmapping algorithm is used to construct a two-dimensional laboratory environment map. In order to test the real-time planning ability of the algorithm, temporary unknown obstacles are added in laboratory environment. Fig. 22 shows the laboratory environment and laboratory map model.



(a) Laboratory environment (b) Laboratory map model

Fig. 22. Laboratory environment.

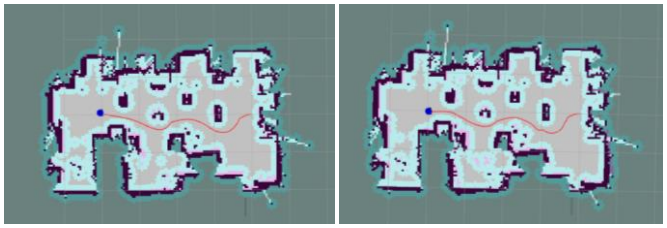
Fig. 23 shows the laboratory environment with temporary obstacles and laboratory map model with temporary obstacles. The radar sensors detect obstacles information and provide real-time feedback, and achieve self-localization through the AMCL (Adaptive Monte Carlo Localization) module.



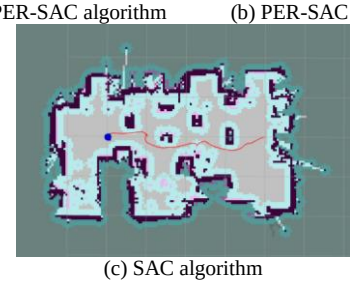
(a) Laboratory environment (b) Laboratory map model

Fig. 23. Laboratory environment with temporary obstacles.

The robot plans a collision-free path from the starting point to the target point, and visualizes the paths planned by the three algorithms in Rviz to verify the real-time obstacle avoidance ability of the algorithms. Fig. 24(a), (b) and (c) show the planned path by the EP-PER-SAC algorithm, PER-SAC algorithm, and the SAC algorithm, respectively. In the real environment, the path lengths of the three algorithms are 5.562 meters, 6.159 meters, and 6.965 meters, respectively. Compared with PER-SAC and SAC algorithms, EP-PER-SAC algorithm has a shorter path length and smoother path during obstacle avoidance. The experiments indicate that the path planned by the proposed algorithm is feasible and effective in both the simulation environment and the real environment, and the performance is better than the original SAC algorithm and PER-SAC algorithm.



(a) EP-PER-SAC algorithm (b) PER-SAC algorithm



(c) SAC algorithm

Fig. 24. Paths planned by the three algorithms in laboratory environment.

V. Conclusion

This research improves the original SAC algorithm and proposes the EP-PER-SAC algorithm based on the Priority Experience Replay and dynamic adjustment of experience pool. Simulation comparisons and real environment experiments are made with the original SAC and PER-SAC algorithms in specific environments to verify that the improved algorithm can reach the target point faster and more efficiently. The improved EP-PER-SAC algorithm has the following characteristics:

1) The Priority Experience Replay technology is added to improve the sampling probability of important samples, and improve the convergence speed and effectiveness of the algorithm.

2) According to the comparison of exploration rate and exploration-utilization rate in the training process, the experience pool is dynamically adjusted to avoid over-fitting of the model and improve the stability of the algorithm.

The improved algorithm has certain practical applications in the field of mobile robots, such as autonomous vehicles, warehouse automation, and service robots working in the unknown environment. We can download the improved algorithm to the controller of a mobile robot. According to the instructions of the algorithm, the robot can perceive the environment in real time, autonomously avoid obstacles and reach the designated target through the optimal path, completing the given transportation task.

However, the improved algorithm is relatively simple and rigid in evaluating the effect of adjusting the method in the experience rules, which may lead to bias in the information obtained from training. The next step of research will consider conducting more complex and diverse evaluations of the adjustment performance in the experience rules, which can enhance the algorithm's adaptability to more complex environments and enable robots to better perform path planning.

Acknowledgment

This work was supported by the Natural Science Foundation of Shandong Province, China (Nos. ZR2023MF015 and ZR2021MF072) and the National Natural Science Foundation of China (Nos. 61973184 and 61473179).

References

[1] Z. S. Wu and W. P. Fu, "A review of path planning method for mobile robot," Advanced Materials Research, vol. 1030, pp. 1588-1591, 2014.

[2] Z. Deng and D. Wang, "Research on parking path planing based on a-star algorithm." Journal of New Media, vol. 5, no. 1, 2023.

[3] H. Jiang and Y. Sun, "Research on global path planning of electric disinfection vehicle based on improved a* algorithm," Energy Reports, vol. 7, pp. 1270-1279, 2021.

[4] P. Wang, S. Gao, L. Li, B. Sun and S. Cheng, "Obstacle avoidance path planning design for autonomous driving vehicles based on an improved artificial potential field algorithm," Energies, vol. 12, no. 12, p. 2342, 2019.

[5] T. Gao, J. Wang, Z. Wang, W. Chen, G. Chen and S. Zhang, "Research on path planning of mobile robot with a novel improved artificial potential field algorithm," Mathematical Problems in Engineering, vol. 2022, 2022.

[6] X. Li, "Path planning of intelligent mobile robot based on dijkstra algorithm," in Journal of Physics: Conference Series, vol. 2083, no. 4, IOP Publishing, 2021, p. 042034.

[7] P. G. Luan and N. T. Thinh, "Hybrid genetic algorithm based smooth global-path planning for a mobile robot," Mechanics Based Design of Structures and Machines, vol. 51, no. 3, pp. 1758-1774, 2023.

[8] H. Gao, S. Lu and T. Wang, "Motion path planning of 6-dof industrial robot based on fuzzy control algorithm," Journal of Intelligent & Fuzzy Systems, vol. 38, no. 4, pp. 3773-3782, 2020.

[9] Y. Tan, J. Ouyang, Z. Zhang, Y. Lao and P. Wen, "Path planning for spot welding robots based on improved ant colony algorithm," Robotica, vol. 41, no. 3, pp. 926-938, 2023.

[10] K. Shi, L. Huang, D. Jiang, Y. Sun, X. Tong, Y. Xie and Z. Fang, "Path planning optimization of intelligent vehicle based on improved genetic and ant colony hybrid algorithm," Frontiers in Bioengineering and Biotechnology, vol. 10, p. 905983, 2022.

[11] W. Zhang and G. Wang, "Reinforcement learning-based continuous action space path planning method for mobile robots," Journal of Robotics, vol. 2022, 2022.

[12] Y. Xu, "Research on reinforcement learning algorithm for path planning of multiple mobile robots," in Journal of Physics: Conference Series, vol. 1915, no. 4. IOP Publishing, 2021, p. 042022.

[13] H. Meng and H. Zhang, "Mobile robot path planning method based on deep reinforcement learning algorithm," Journal of Circuits, Systems and Computers, vol. 31, no. 15, p. 2250258, 2022.

[14] W. Lan, X. Jin, X. Chang, T. Wang, H. Zhou, W. Tian and L. Zhou, "Path planning for underwater gliders in time-varying ocean current using deep reinforcement learning," Ocean Engineering, vol. 262, p. 112226, 2022.

[15] P. Wang, J. Qin, J. Li, M. Wu, S. Zhou and L. Feng, "Optimal transshipment route planning method based on deep learning for multimodal transport scenarios," Electronics, vol. 12, no. 2, p. 417, 2023.

[16] H. Gong, P. Wang, C. Ni and N. Cheng, "Efficient path planning for mobile robot based on deep deterministic policy gradient," Sensors, vol. 22, no. 9, p. 3579, 2022.

[17] P. Li, X. Ding, H. Sun, S. Zhao and R. Cajo, "Research on dynamic path planning of mobile robot based on improved ddpg algorithm," Mobile Information Systems, vol. 2021, pp. 1-10, 2021.

[18] D. Zahng, Z. Xuan, Y. Zhang, J. Yao, X. Li and X. Li, "Path planning of unmanned aerial vehicle in complex environments based on state-detection twin delayed deep deterministic policy gradient," Machines, vol. 11, no. 1, p. 108, 2023.

[19] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra and M. Riedmiller, "Deterministic policy gradient algorithms," in International conference on machine learning, Pmlr, 2014, pp. 387-395.

[20] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver and D. Wierstra, "Continuous control with deep reinforcement learning," US Patent, vol. 15, no. 217,758, 2020.

[21] S. Fujimoto, H. Hoof, and D. Meger, "Addressing function approximation error in actor-critic methods," in International conference on machine learning. PMLR, 2018, pp. 1587-1596.

[22] T. Haarnoja, A. Zhou, P. Abbeel and S. Levine, "Soft actor-critic: off-policy maximum entropy deep reinforcement learning with a stochastic actor," in International conference on machine learning. PMLR, 2018, pp. 1861-1870.

[23] Y. Zhou, J. Shu, H. Hao, H. Song and X. Lai, "Uav 3d online track planning based on improved sac algorithm," Journal of the Brazilian Society of Mechanical Sciences and Engineering, vol. 46, no. 1, p. 12, 2024.

[24] T. Haarnoja, A. Zhou, P. Abbeel and S. Levine, "Soft actor-critic: off-policy maximum entropy deep reinforcement learning with a stochastic actor," in International conference on machine learning. PMLR, 2018, pp. 1861-1870.