2022 IEEE International Conference on
Systems, Man, and Cybernetics (SMC)

PL-TD3: A Dynamic Path Planning Algorithm of Mobile Robot

Yijian Tan[1,2], Yunhan Lin[1,2,3,*], Tong Liu[1,2] and Huasong Min[3]

2022 IEEE International Conference on Systems, Man, and Cybernetics (SMC) | 978-1-6654-5258-8/22/$31.00 ©2022 IEEE | DOI: 10.1109/SMC53654.2022.9945119

***Abstract*—In this paper, Prioritized Experience Replay (PER) strategy and Long Short Term Memory (LSTM) neural network are introduced to the path planning process of mobile robots, which solves the problems of slow convergence and inaccurate perception of dynamic obstacles with the Twin Delayed Deep Deterministic Policy Gradient (TD3) algorithm. We dubbed this new method as PL-TD3. Firstly, we improve the convergence speed of the algorithm by introducing PER strategy. Secondly, we use LSTM neural network to achieve the improvement of the algorithm for dynamic obstacle perception. In order to verify the method of this paper, we design static environment, dynamic environment and adaptability to dynamic experiments to compare and analyze the methods before and after improvement. The experimental results show that PL-TD3 outperforms TD3 in terms of execution time and execution path length in all environments.**

***Index Terms*—reinforcement learning, TD3, mobile robot, path planning**

I. Introduction

With the rapid development of modern machinery manufacturing and computer technology, intelligent mobile robots can perform complex and dangerous tasks in various occasions instead of humans, and the path planning technology of mobile robots is the primary condition for mobile robots to accomplish their target tasks [1], [2], [3]. The path planning of a mobile robot is to find a collision-free path from the starting position to the target position in an environment with obstacles, and also needs to satisfy the conditions that the path is as smooth as possible, the path length is as short as possible, and the time spent is as low as possible.

Existing robot path planning methods can be divided into traditional methods and machine learning based methods. Traditional path planning methods rely on high-precision real-time maps. Machine learning-based methods are less map-dependent because the robot can learn from the acquired experience and adapt its obstacle avoidance strategy adaptively.

The traditional approach consists of a global planner and a local planner, where the global planner is responsible for generating a series of feasible path points based on known map information and selecting the optimal path based on different optimization schemes [4]. In [5], Belanov et al. used a heuristic mechanism for reverse search that allows path planning in unknown environments with variable starting position and fixed target position, and named D* Lite. In [6], Liu et al. integrate the Jump-A* and Dynamic Window Approach (DWA) [7] algorithms to optimize the number of iterations using the jump-point search method and the distance evaluation function to obtain the global information of the paths, but the dynamic narrow space will generate more collision candidate paths. The local planner differs from the global planner in that the local planner is responsible for developing specific action strategies in the local environment based on the environmental information gathered by the sensors carried by the robot. The principle of the DWA algorithm is to sample multiple sets of data in the robot's velocity space and simulate the robot's path at these speeds for a certain period of time, and select the speed corresponding to the optimal path to drive the robot path. In [8], Min et al. combined a case-base reasoning approach with an improved APF method to obtain optimal paths and higher efficiency of the algorithm. Since the global planner and the local planner are independent of each other in traditional path planning methods, they need to be designed separately, which makes it difficult to apply traditional path planning methods to complex and dynamic scenarios [4].

Methods based on machine learning can be divided into methods based on supervised learning and methods based on reinforcement learning. The supervised learning-based approach works with a given data set from which the agent can quickly learn how to plan an efficient path [9]. Motion Planning network (MPNet) [10] is a general near-optimal heuristic algorithm for path planning in visible and invisible environments using neural networks and recursively calls itself to generate connectable paths in both directions. In [11], Khan et al. used the substitution invariance of Graph neural networks (GNNs) to robustly encode the topology of the planning space for path planning. In [12], Yu et al. reduced the number of collisions checks by training GNN models for path exploration and path smoothing given a random geometry map generated by batch sampling. However, in supervised learning-based approaches, it is time consuming collect a large number of data sets that match the environment for the training of the network.

In contrast to supervised learning approaches, reinforcement-based learning approaches can learn from both the exploration and the rewards obtained. In [13], Xin et al. first applied Deep Q Network (DQN) to the path planning of mobile robots. In [14], Schaul et al. proposed a Prioritized Experience Replay (PER) strategy, which improved the experience replay method of the DQN algorithm by assigning weights to experiences and prioritizing the replay of experiences with high weights to speed up the convergence of the model. However, in environments where dynamic obstacles exist, the above reinforcement learning algorithms need to highly discretize the action space to achieve obstacle avoidance, which

*Coresponding author. E-mail: yhlin@wust.edu.cn. This work is supported by National Natural Science Foundation of China (Grant No.: 62073249).

[1]College of Computer Science and Technology, Wuhan University of Science and Technology, Wuhan 430065, China.

[2]Hubei Province Key Laboratory of Intelligent Information Processing and Real-time Industrial System, Wuhan 430065, China.

[3]Institute of Robotics and Intelligent Systems, Wuhan University of Science and Technology, Wuhan 430081, China.

can result in dimensional disasters in the action space. Therefore, continuous reinforcement learning algorithms will be more adaptive in dynamic environments. In [15], Yu et al. experimentally demonstrated that the execution path of the Deep Deterministic Policy Gradient (DDPG) algorithm has shorter path length and smoother path than the execution path of DQN and its improved algorithms. In [16], Dong et al. used prior knowledge and an adaptive exploration strategy based on greedy strategy to improve the DDPG algorithm and improve the exploration efficiency. In [17], Zhao et al. added a BatchNorm layer to the policy network of DDPG to normalize the input states and proposed the TPR-DDPG algorithm to enable the robot to reach the target position quickly in complex environments. However, all the above DDPG-based algorithms suffer from an overestimation bias of Q value. In [18], Wang et al. proposed a globally guided reinforcement learning method applied to dynamic path planning for robots, which combines a novel reward structure to solve the problem of unnecessary detours when robots encounter dynamic obstacles. However, its algorithm is slow to converge and inefficient in complex environments. Among the reinforcement learning algorithms, the Twin Delayed Deep Deterministic Policy Gradient (TD3) [19] is a reinforcement learning algorithm with superior performance. But in complex dynamic environments, the TD3 algorithm still suffers from slow convergence and inaccurate perception of dynamic obstacles. In this paper, we improve the TD3, firstly, by using PER strategy, which improves the convergence speed of the algorithm; secondly, at the network layer, we combine the Long Short Term Memory (LSTM) [20] network with ordinary neural network, which enhance the agents perception ability for the dynamic environment. The main contributions of this paper are summarized as follows:

- In this paper, we propose a PL-TD3-based path planning algorithm for mobile robots, which improves the convergence speed of the algorithm and the perception of dynamic obstacles by combining the TD3 algorithm with LSTM unit and PER strategy.
- We design the reward function to improve the algorithm for dynamic obstacle perception, which contains of three parts: the angle between the robot forward and the target position, the distance between the robot's current position and the target position, and the shortest distance between the robot and the obstacle.
- In order to verify the superiority of the PL-TD3 algorithm, we designed static environment, dynamic environment and adaptability to dynamic experiments to conduct comparative analysis of our proposed method.

The overall structure of this paper is as follows: In section 2, the PL-TD3 algorithm is proposed. Section 3 is designed to verify the effectiveness of PL-TD3 through simulation experiments. Section 4 is the conclusion of this paper.

II. The PL-TD3 Algorithm

The PL-TD3 model consists of three main parts: the structure of the PL-TD3 algorithm, the sampling strategy of the experience pool and the design of the reward function.

A. The structure of the PL-TD3

In the experience playback of the TD3 algorithm, the select probability of each experience is equal, but the importance of each experience is not the same. The PER strategy can set the probability of experience being played back according to the Temporal Difference error (TD-error) of the experience, so we use the PER strategy to increase the probability of experience with large TD-error being played back to improve the convergence speed of the algorithm. As we known, increasing the probability of playback for experiences with large TD-error improves the convergence rate of the model. When a robot is in a dynamic environment, the typical neural network can only process individual inputs, which cause to there is no correlation between its preceding and following inputs. The value of the hidden layer of the recurrent neural network depends not only on the current input but also on the value of the last hidden layer. Therefore, using the continuous state acquired by the robot in a dynamic environment as input enables the robot to perceive the environment more adequately. However, ordinary recurrent neural networks face the problem of gradient disappearance or gradient explosion. LSTM controls the transmission state by gating the state and solves the gradient disappearance or gradient explosion problem by remembering the information that requires long time to remember and forgetting the unimportant information. Therefore, in this paper, LSTM is added to the network model of TD3 algorithm and PER strategy is adopted in the empirical playback. The overall structure of the algorithm is shown in Fig. 1.

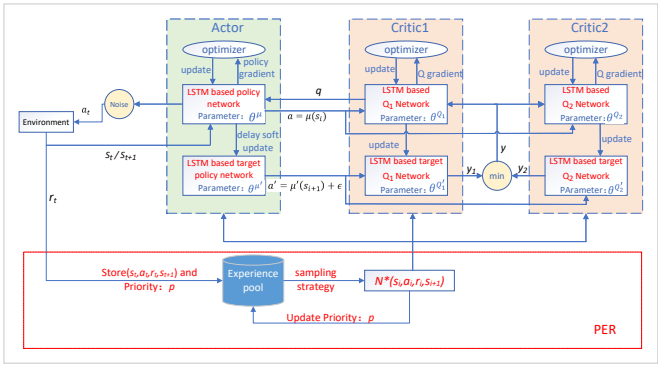


Fig. 1. The structure of PL-TD3 algorithm.

Where s_t, a_t, r_t, s_{t+1} represent the current state, the action, the reward obtained and the new state after the execution of the action, respectively. s_t consists of four parts, the first part is the LiDAR data, the second part is the distance between the current position of the robot and the target position, the third part is the distance between the robot and the nearest obstacle, and the last part is the angle between the robot directly in front and the nearest obstacle. a_t is the prediction of the Actor policy network based on the current state of the robot, s_t, which controls the angular velocity of the robot at a frequency of 20 Hz. s_{t+1} is the new state reached by the robot after executing action a_t. r_t is the reward obtained by the robot for performing action a_t. In order to ensure that new experiences are added to the experience pool are played back at least once, new experiences added to the pool have the highest priority.

In the iterative process, when the number of experiences in the experience pool is greater than N, the algorithm continuously uses the PER strategy to select N experiences from the experience pool, calculates their TD-error, and updates their priorities in the experience pool according to the TD-error. These experiences are then used to update Actor Network and Critic Network. The update method of the Critic network is shown in (1) - (4).

$$a' = \mu'_{\theta^{\mu'}}(s_{i+1}) + \epsilon \tag{1}$$

$$y = r + \gamma \min\{Q_{\theta^{Q'}1}(s_{i+1,a'}), Q_{\theta^{Q'}2}(s_{i+1,a'})\} \tag{2}$$

$$\theta^{Q_1} \leftarrow argmin_{\theta^{Q_1}} N^{-1} \sum (y - Q_{\theta^{Q_1}}(s_i, a_i))^2 \tag{3}$$

$$\theta^{Q_2} \leftarrow argmin_{\theta^{Q_2}} N^{-1} \sum (y - Q_{\theta^{Q_2}}(s_i, a_i))^2 \tag{4}$$

where Q represents the Critic network, and μ represents the Actor network; θ^Q is the parameter of the Critic network, and θ^μ is the parameter of the Actor network; γ is the discount rate; ϵ is the Gaussian noise added to the action space. The Actor network adopts a delayed update method, that is, after the Critic network is updated three times, the Actor network is updated once. The Actor network update method is shown in (5).

$$\nabla_{\theta^\mu} J(\theta^\mu) = N^{-1} \sum \nabla_a Q_{\theta^{q_1}}(s_i, a)\,|_{a=\mu_{\theta^\mu}(s_i)} \nabla_{\theta^\mu} \mu_{\theta^\mu}(s_i) \tag{5}$$

Once the Actor network is updated, all target networks are updated by soft update. All the target networks update method is shown in (6) and (7).

$$\theta^{Q'_j} \leftarrow \tau\theta^{Q_j} + (1-\tau)\theta^{Q'_j} \tag{6}$$

$$\theta^{\mu'} \leftarrow \tau\theta^{\mu} + (1-\tau)\theta^{\mu'} \tag{7}$$

where $j = 1, 2$; τ is the coefficient of the soft update.

B. The sampling strategy of the experience pool

PL-TD3 adopts weight-based priority experience replay based on the experience replay mechanism of TD3. The importance of experience is judged by the TD-error of experience in the priority experience replay mechanism, which is the difference between the action estimate and the output value of the current value function. the TD-error is calculated as shown in (8).

$$\delta_t = r_t - Q_{\theta^{Q_1}}(s_t, a_t) + \gamma \min\left\{Q_{\theta^{Q'_1}}\left(s_{t+1}, \mu'_{\theta^{\mu'}}(s_{t+1})\right), Q_{\theta^{Q'_2}}\left(s_{t+1}, \mu'_{\theta^{\mu'}}(s_{t+1})\right)\right\} \tag{8}$$

Where s_t is the state of the robot at time t; a_t is the action performed by the robot in state s_t; s_{t+1} is the new state reached after the robot performs the action; r_t is the reward obtained by the robot performing the action; γ is the discount factor. When the TD-error is larger, the more inaccurate the output of the current value function is, i.e., the higher the availability of that experience. To ensure that the new sample with unknown TD-error is played back at least once, it needs to be placed first, and then the sample with the largest TD-error is played back each time. Setting the experience with a TD-error of 0 as the lowest priority not only ensures that the priority of the experience is proportional to its TD-error, but also ensures that the probability of all samples will not decrease to 0.

C. The design of the reward function

The reward function is the benchmark to evaluate robot actions in reinforcement learning. The reward function of this paper is designed as follows: If the robot reaches the target position, give the maximum positive reward R_{reach}; If the robot collides with an obstacle, give the largest negative reward $R_{collide}$; If the robot neither collides nor reaches the target area, its reward consists of the robot's yaw angle reward R_1 multiplied by the distance reward R_2 between the robot and the target point plus the minimum distance reward R_3 between the robot and the obstacle. The calculation of the reward value is shown in (9). When the yaw angle of the robot is quite different from the direction of the target position, even if the distance between the robot and the target position is very close, the robot will move in the opposite direction to the target position, and the robot needs to obtain a small negative reward. Therefore, using the product of R_1 and R_2 as part of the total reward not only can decide whether the reward is positive or negative, but also can adjust the size of the reward according to the distance.

$$Reward = \begin{cases} R_{reach}, & \text{reach the target position} \\ R_{collide}, & \text{collision with obstacles} \\ R_1 R_2 + R_3, & \text{other} \end{cases} \tag{9}$$

The calculation of the yaw angle reward R_1 of the robot is shown in (10).

$$R_1 = 4 * (1 - abs(\alpha)) \tag{10}$$

Where α is the radian representation of the angle between the forward direction of the robot and the target position, and its value range is $[-\pi, \pi]$.

The calculation of the distance reward R_2 between the robot and the target position is shown in (11).

$$R_2 = 2^{\frac{d_{t-1}}{d_t}} \tag{11}$$

Where d_{t-1} and d_t represent the distance between the previous robot and the target position and the distance between the current robot and the target position, respectively.

The calculated representation of the minimum distance R_3 between the robot and the obstacle is shown in (12).

$$R_3 = \begin{cases} -5, & min_{obs} < 0.5 \\ 0, & min_{obs} \geq 0.5 \end{cases} \tag{12}$$

Where min_{obs} is the shortest distance between the current robot and the obstacle. When the robot is close to the obstacle, it is easy to collide. Therefore, it is necessary to give a negative reward to the robot to keep the robot at a safe distance from obstacles. However, if the given negative reward is too large, it will affect the robot's obstacle avoidance strategy, which makes the robot unable to pass through the narrow area. Through experimental testing in environments with dynamic obstacles and narrow intersections, -5 is a reasonable value.

III. EXPERIMENT

In order to verify the method of this paper, we built a 3D map simulation model using Robot Operating System (ROS) in Ubuntu 16.04 with AMD R5 3600X, NVIDIA

GTX1650 and 16G running memory, and designed static environment, dynamic environment and adaptability to dynamic experiments to compare and analyze the method before and after the improvement. When a robot reaches the target position the reward setting is $R_{reach} = 1000$, and when a robot collides with an obstacle the reward setting is $R_{collide} = -800$. In the training process, in order to ensure the diversity and the complexity of the environment, the target position is randomly generated. When the robot reaches the target position, a new target position is immediately generated randomly until the robot collides with an obstacle or reaches the maximum number of steps to move, the robot resets to the center of the map and starts a new iteration.

A. The definition of static environment and dynamic environment

The static environment is shown in Fig. 2. The black car in the middle of map is the Turtlebot robot, and the obstacles are composed of 13 brown-yellow walls.

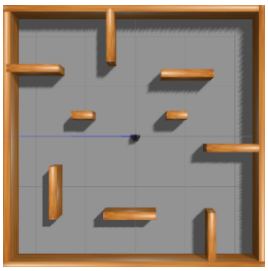


Fig. 2. Static environment. The obstacles are composed of 13 brown-yellow walls.

The dynamic environment is shown in Fig. 3, in which the white columns represent movable pedestrian, and the red path is the path that simulates the pedestrian's dynamic obstacle action.

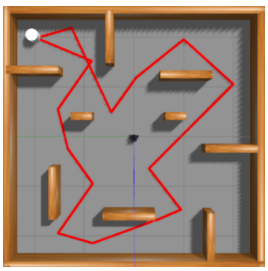


Fig. 3. Dynamic environment. The white columns represent movable pedestrian, and the red path is the path of pedestrian.

B. Static environment

Fig. 4 shows the reward values for PL-TD3 and TD3 training. In terms of convergence speed, PL-TD3 only takes 250 rounds to converge, while TD3 algorithm needs 600 rounds to converge. From the perspective of maximum reward, the average maximum reward value of PL-TD3 is 500 higher than that of TD3, which accounts for the fact that the path performed by the PL-TD3 algorithm is shorter than the path performed by the TD3 algorithm.

In this environment, we test both algorithms starting from the position of the robot and ending with the red square. If

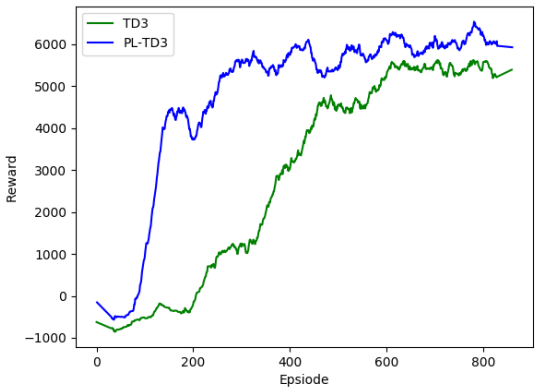


Fig. 4. Comparison of training rewards between PL-TD3 and TD3 in the static environment.

the starting position and the target position of the robot are connected in a straight line and pass through two obstacles, it is regarded as normally complicated, and if it passes through three obstacles, it is regarded as very complicated. The comparison of the path length and execution time of the two algorithms is shown in Table I, and the path is shown in Fig. 5. The experimental results show that the paths executed by the robot can all bypass the obstacles when close to them, because PL-TD3 has a more complete perception of the environment. In contrast, TD3 needs to be farther away from the obstacle to perform obstacle avoidance, which means that the paths executed by the PL-TD3 algorithm are shorter than those of TD3.

TABLE I
PATH LENGTH AND EXECUTION TIME OF PL-TD3 AND TD3 IN THE STATIC ENVIRONMENT

Test scene in Fig. 5	TD3(path length/time)	PL-TD3(path length/time)
a	4.15m/27.69s	3.66m/24.43s
b	3.78m/25.24s	3.65m/24.25s
c	5.53m/36.89s	5.43m/35.91s
d	4.98m/33.25s	4.82m/32.17s

C. Dynamic Environment

Fig. 6 shows the reward values for PL-TD3, PER+TD3 and TD3 training. PL-TD3 and PER+TD3 only need 700 rounds to reach the maximum convergence value of TD3, which proves that the PER strategy increases the convergence speed of the algorithm. The average maximum reward value of the PL-TD3 algorithm is 200 higher than that of PER+TD3 and 600 higher than that of TD3. It means that the LSTM unit enhances the agent's perception of the environment and the analysis ability of the neural network, enabling the robot to avoid obstacles with better strategies.

We designed two test experiments, obstacle avoidance in a wide area and obstacle avoidance at a narrow area. The robot position in Fig 7 is the starting position, the red rectangle is the target position, and the white ellipse area is the area where the robot meets the pedestrian for the test. Finally, the average path length and the average execution time of the robot obtained from the experiment are shown in Table II.

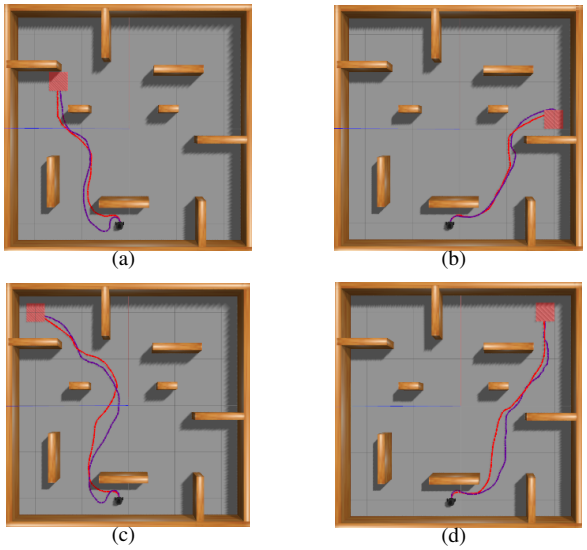


Fig. 5. The path of PL-TD3 and TD3 in the static environment. The robot position is the starting position, and the red rectangle is the target position. The red path is generated by PL-TD3 and the purple path is generated by TD3. (a) Normally complexity 1. (b) Normally complexity 2. (c) Very complexity 1. (d) Very complexity 2.

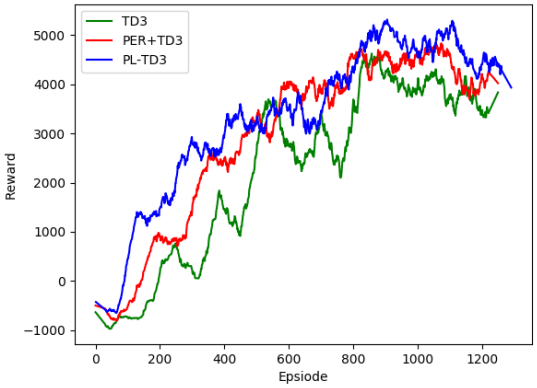


Fig. 6. Comparison of training rewards of PL-TD3, PER+TD3, and TD3 in the dynamic environment.

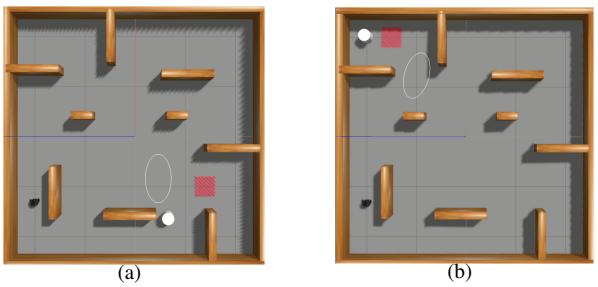


Fig. 7. Dynamic test environment. The robot position is the starting position, and the red rectangle is the target position. (a) Robot meets obstacles in wide area. (b) Robot meets obstacles in narrow area.

Fig. 8 shows the obstacle avoidance paths of the two algorithms when encountering pedestrian in different areas. It can be seen that TD3 does not perceive the state of the environment sufficiently, and TD3 will choose to move towards an open area when meeting the pedestrian, and then continue to move towards the target position after the pedestrian passes by while PL-TD3 will stay close to the pedestrian and set off toward the target position. The results show that the dynamic obstacle avoidance capability of PL-TD3 is stronger than that of TD3 in this environment.

TABLE II
PATH LENGTH AND EXECUTION TIME OF PL-TD3 AND TD3 IN THE DYNAMIC ENVIRONMENT

Area where obstacles meet	TD3(path length/time)	PL-TD3(path length/time)
Wide area	6.32m/37.12s	5.13m/30.14s
Narrow area	5.40m/31.71s	4.54m/26.77s

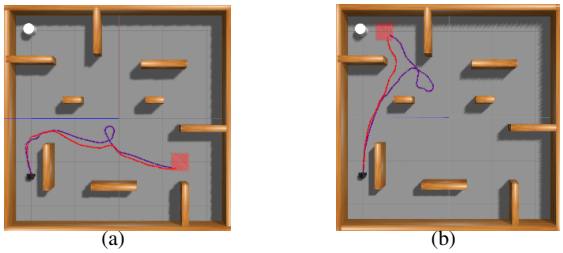


Fig. 8. The path of PL-TD3 and TD3 in dynamic environment. The robot position is the starting position, and the red rectangle is the target position. The red path is generated by PL-TD3 and the purple path is generated by TD3. (a) Robot meets obstacles in wide area. (b) Robot meets obstacles in narrow area.

D. Adaptability to dynamic experiments

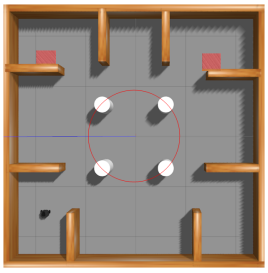


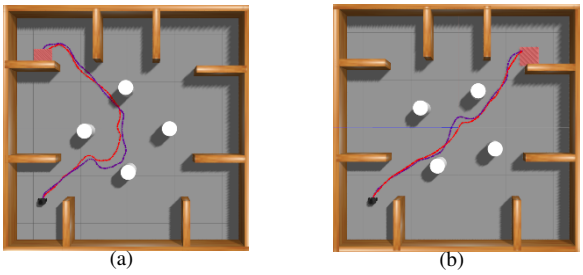
Fig. 9. Adaptability to dynamic experiments. The red circle in the map is the path of the obstacles. The robot position is the starting position, and the red rectangle is the target position.

We test the adaptability of the algorithm in the dynamic experiments as shown in Fig. 9. The robot is initially located in the room in the lower left corner, and needs to pass through the area where the pedestrians are walking simulated by four white columns to reach the rooms in the upper left corner and the upper right corner, respectively. The red circle in the map is the path of the simulated pedestrians movement. The path execution length, execution time and success rate of reaching the target position for both algorithms are shown in Table III, and the execution path is shown in Fig. 10. Because PL-TD3 processes environmental information more

adequately, not only its path length shorter than TD3 and execution time is significantly less, but also its success rate of reaching the target position is significantly higher than that of TD3.

TABLE III
PATH LENGTH, EXECUTION TIME AND SUCCESS RATE OF REACHING THE TARGET POSITION OF PL-TD3 AND TD3 IN THE ADAPTABILITY TO DYNAMIC EXPERIMENTS

Test experiment obstacles meet	TD3(path length/time/ success rate)	PL-TD3(path length/time/ success rate)
Arrive in the upper left room	5.78m/33.97s/86%	5.27m/31.04s/92%
Arrive in the upper right room	5.37m/31.61s/84%	4.55m/26.81s/92%



(a) (b)

Fig. 10. The path of PL-TD3 and TD3 in the adaptability to dynamic experiments. The robot position is the starting position, and the red rectangle is the target position. The red path is generated by PL-TD3 and the purple path is generated by TD3. (a) Target position on the upper left corner. (b) Target position on the upper right corner.

IV. CONCLUSION

In this paper, we propose a PL-TD3-based path planning method, which improves the convergence speed of the algorithm and the perception ability for dynamic environment by introducing PER strategy and combining with LSTM neural network. The static environment, dynamic environment and adaptability to dynamic experiments are designed to compare and analyze the methods before and after the improvement.

The experimental results show that: in the static environment, PL-TD3 has a 5% reduction in execution path length and 4% faster robot execution time compared to TD3; in the dynamic environment, the execution path length of PL-TD3 is reduced by 17% compared to TD3, and robot execution time is accelerated by 17%; in the adaptability to dynamic experiments, PL-TD3 showed a 12% reduction in execution path length over TD3, a 10% speedup in robot execution time, and an 11% increase in success rate. All the above experimental prove that the algorithm based on our PL-TD3 path planning method has good performance.

For future work, we will consider combining LIDAR and depth cameras to perceive obstacle categories, and combine social etiquette to apply different obstacle avoidance rules for different categories of obstacles, which will eventually be applied to real scenarios.

REFERENCES

[1] X. Xuesu, L. Bo, W. Garrett and S. Peter, "Motion control for mobile robot navigation using machine learning: a survey," arXiv preprint arXiv:2011.13112, 2020.

[2] C. Sun, W. Liu and L. Dong, "Reinforcement learning with task decomposition for cooperative multiagent systems," IEEE transactions on neural networks and learning systems, 2021, 32(5), pp. 2054-2065.

[3] S. Aradi, "Survey of Deep Reinforcement Learning for Motion Planning of Autonomous Vehicles," IEEE Transactions on Intelligent Transportation Systems, 2022, 23(2), pp. 740-759.

[4] D. Lu, Z. He, C. Song and C. Sun, "A review of mobile robot motion planning methods: from classical motion planning workflows to reinforcement learning-based architectures," arXiv preprint arXiv:2108.13619, 2021.

[5] D. Belanov, M. Mach, P. Sincak and K. Yoshida, "Path Planning on Robot Based on D* Lite Algorithm," 2018 World Symposium on Digital Intelligence for Systems and Machines (DISA), 2018, pp. 125-130.

[6] L. Liu et al., "Global Dynamic Path Planning Fusion Algorithm Combining Jump-A* Algorithm and Dynamic Window Approach," IEEE Access, 2021, 9, pp. 19632-19638.

[7] M. Seder and I. Petrovic, "Dynamic window based approach to mobile robot motion control in the presence of moving obstacles," IEEE International Conference on Robotics and Automation (ICRA), 2007, pp. 1986-1991.

[8] H. Min, Y. Lin, S. Wang, F. Wu and X. Shen, "Path planning of mobile robot by mixing experience with modified artificial potential field method," Advances in Mechanical Engineering, 2015, 7(12): 1687814015619276.

[9] Y. -H. Kim, J. -I. Jang and S. Yun, "End-to-end deep learning for autonomous navigation of mobile robot," IEEE International Conference on Consumer Electronics (ICCE), 2018, pp. 2158-4001.

[10] A. H. Qureshi, Y. Miao, A. Simeonov and M. C. Yip, "Motion Planning Networks: Bridging the Gap Between Learning-Based and Classical Motion Planners," IEEE Transactions on Robotics, 2021, 37(1), pp. 48-66.

[11] A. Khan, A. Ribeiro, V Kumar and A. G. Francis, "Graph neural networks for motion planning," arXiv preprint arXiv:2006.06248, 2020.

[12] C. Yu and S. Gao, "Reducing collision checking for sampling-based motion planning using Graph Neural Networks," Advances in Neural Information Processing Systems, 2021, 34(2021), pp. 4274-4289.

[13] J. Xin, H. Zhao, D. Liu and M. Li, "Application of deep reinforcement learning in mobile robot path planning," 2017 Chinese Automation Congress (CAC), 2017, pp. 7112-7116.

[14] T. Schaul, J. Quan, I. Antonoglou and D. Silver, "Prioritized Experience Replay," Proceedings of the IEEE International Conference on Learning Representations (ICLR), 2016, pp. 21-46.

[15] J. Yu, Y. Su and Y. Liao, "The path planning of mobile robot by neural networks and hierarchical reinforcement learning," Frontiers in Neurorobotics, 2020, 14(63), pp. 1-12.

[16] Y. Dong and X. Zou, "Mobile Robot Path Planning Based on Improved DDPG Reinforcement Learning Algorithm," International Conference on Software Engineering and Service Science (ICSESS), 2020, pp. 52-56.

[17] Y. Zhao, X. Wang, R. Wang, Y. Yang and F. Lv, "Path Planning for Mobile Robots Based on TPR-DDPG," International Joint Conference on Neural Networks (IJCNN), 2021, pp. 1-8.

[18] B. Wang, Z. Liu, Q. Li and A. Prorok, "Mobile Robot Path Planning in Dynamic Environments Through Globally Guided Reinforcement Learning," IEEE Robotics and Automation Letters, 2020, 5(4), pp. 6932-6939.

[19] S. Fujimoto, H. Hoof and D. Meger, "Addressing function approximation error in actor-critic methods," International Conference on Machine Learning (ICML), 2018, pp. 1587-1596.

[20] M. Sundermeyer, R. Schlter and H. Ney, "LSTM neural networks for language modeling," Thirteenth annual conference of the international speech communication association. 2012, pp. 1-6.