PATH OPTIMIZATION FOR DYNAMIC OBSTACLE AVOIDANCE OF PIONEER ROBOT BASED ON DDPG SMART AGENT

Qingchun ZHENG[1,2], Zhi PENG[1,2,3], Peihao ZHU[1,2*], Yangyang ZHAO[4], Wenpeng MA[1,2]

To improve the path efficiency for dynamic obstacle avoidance algorithms of Pioneer robots, this paper proposes an approach to create deep deterministic policy gradient (DDPG) smart agents with recurrent neural networks (RNN) to optimize the path strategy. This approach is realized by training RNN with the backpropagation (BP) algorithm to train the agent. First, the SolidWorks model and obstacle avoidance model of the Pioneer robot is established. The LIDAR sensor is used to obtain environmental information. Further, deep reinforcement learning obstacle avoidance algorithm based on RNN is presented. The dynamic obstacle avoidance path of the Pioneer robot is further simulated in two different virtual robot experiment platform (V-REP) scenarios. The simulation results show that the proposed algorithm shortens the training time by 26.3% compared with the DDPG algorithm. Our proposed scheme can improve the path efficiency for dynamic obstacle avoidance of the Pioneer robot.

Keywords: recurrent neural network, deep deterministic policy gradient, mobile robot, pioneer, deep reinforcement learning

1. Introduction

With the continuous development of science and technology in the world, mobile robot (MR) [1] technology is widely used in the robot industry, which is one of the key research contents in the field of robotics. Navigation and obstacle avoidance have become key technologies for mobile robots to deal with complex working environments. However, in dynamic environments with more complex requirements, obstacle avoidance alone is no longer sufficient for the work of mobile robots.

[1] Tianjin Key Laboratory for Advanced Mechatronic System Design and Intelligent Control, Tianjin University of Technology, Tianjin, 300384, China.
[2] National Demonstration Center for Experimental Mechanical and Electrical Engineering Education (Tianjin University of Technology).
[3] School of Mechanical Engineering, Tianjin University of Technology, Tianjin, 300384, China.
[4] School of Computer Science and Engineering, Tianjin University of Technology, Tianjin, 300384, China.
* E-mail: zhupeihao_ch@163.com

In complex obstacle environments, mobile robot faces a large number of external obstacles and disturbances. It is crucial to identify these disturbances, quickly determine the obstacle location, and adopt an optimal obstacle avoidance strategy. Mobile robots need to optimize the obstacle avoidance algorithm and obstacle avoidance strategy when avoiding obstacles in front of them [2]. In addition, the highly dynamic environment is prone to incorrect data correlation, which leads to robot positioning errors or positioning failures [3] and affects the normal operation of mobile robots. Therefore, improving the accuracy and localization robustness of the mobile robot obstacle avoidance algorithm is the most important task to improve the obstacle avoidance capability of mobile robots.

Traditional path optimization obstacle avoidance algorithms include A-star, rapidly exploring random trees (RRT), and artificial potential field (APF). Liu et al. [4] proposed a dynamic fusion pathfinding algorithm based on the Delaunay triangulation algorithm and improved A-star, which improves the success rate of path planning for mobile robots under complex obstacles. This method applies to simple maps and increases consumption when the number of path nodes increases. Song et al. [5] proposed an improved smooth rapidly exploring random tree algorithm to remove redundant nodes and generate smooth paths, which can effectively shorten the path length of global path planning. This method is suitable for simple path planning, while complex path needs to combine with other algorithms to train the dataset. Rostami et al. [6] proposed an improved artificial potential field method to avoid collision between mobile robots and fixed obstacles. This method is at the local minimum value and the target is inaccessible when the obstacle is near the target. Wu et al. [7] proposed an algorithm based on beetle antennae search and the APF algorithm, which accelerates the convergence speed and avoids the local minimum problem of the APF algorithm.

In addition to traditional obstacle avoidance algorithms, reinforcement learning methods are also the focus of research. Not only improve the accuracy of mobile robot obstacle avoidance algorithms but also advance the current mobile robot obstacle avoidance technology. Choi et al. [8] proposed a framework for reinforcement learning in decentralized collision avoidance where each agent independently makes its decision without communication with others. Peng et al. [9] proposed a novel method based on the multistep update method and double deep Q-network to improve autonomous navigation. This method adopts terminal and non-terminal rewards to train the mobile robot navigation, which improves the learning speed and reduces the learning time. Cheng et al. [10] trained nonholonomic wheeled mobile robots to reach the followed path and avoid obstacles based on a deep deterministic policy gradient. This method needs to optimize the strategy and design the reward function.

As can be seen from the above research, the performance of path optimization obstacle avoidance algorithms based on deep reinforcement learning outperforms the traditional method. DDPG algorithm is a deterministic strategy with limited exploration ability. Therefore, this paper proposes a method to create DDPG smart agents with recurrent neural networks to optimize the path strategy. The RNN is trained with a BP algorithm to minimize the loss to solve the gradient disappearance before the RNN update. The main contributions of this paper can be described as follows:

(1) We propose an approach to create DDPG smart agents with recurrent neural networks, which is based on a backpropagation algorithm to train RNN to train the agent. In dynamic and complex environments, the smart agent is used to learn how to avoid obstacles and reach the target point efficiently.

(2) Our proposed algorithm reduces the training time by 26.3% compared with the DDPG algorithm.

(3) Path optimization is integrated with reinforcement learning-based obstacle avoidance to solve the problem of not finding paths in dynamic environments, thus improving path efficiency.

The main content of the work is described as follows. Section 2 presents the Pioneer robot model and obstacle avoidance model. Section 3 depicts the dynamic obstacle avoidance algorithm based on deep reinforcement learning. In Section 4, the dynamic obstacle avoidance path of the Pioneer robot is simulated. In Section 5, we summarize our research progress.

2. Pioneer robot modeling

2.1 Pioneer robot model

The research object of mobile robots in this paper is the Pioneer3-DX robot. Its SolidWorks model is shown in Figure 1. Pioneer3-DX is a two-wheel differential drive mobile robot, which is suitable for all-terrain operation and experiments. The robot has a total weight of 12kg and can carry 12kg on tiles and floors. It can easily work on grass, dirt, asphalt, and other complex terrains. The Pioneer3-DX is equipped with batteries, wheel encoders, 19cm tires, 16 anti-collision sonars, a microcontroller with ARCOS firmware, and a mobile robot software development kit. Its versatility and reliability make it the research platform of choice for advanced intelligent robots.

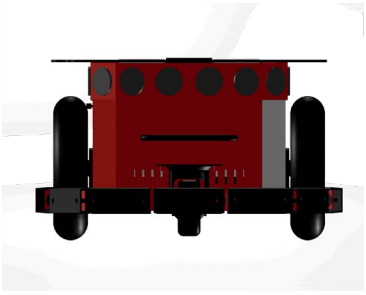


Fig. 1. Pioneer robot model

2.2 Pioneer robot obstacle avoidance model

The purpose of obstacle avoidance [11] is to enable the Pioneer robot to detect obstacles in the known environment map, avoid obstacles in time, and finally reach the target destination. The regular curve of the mobile robot path tracking is shown in Equation (1).

$$Path = \left\{ X_r(k) \in \square^{\,n} \middle| X_r(k) = p(\theta_r(k)) \right\} \tag{1}$$

Where, $X_r(k)$ is the reference position of the mobile robot at k time, $p(\theta_r(k))$ is the path at k time, and $\theta_r(k)$ is the path parameter at k time.

The necessary condition of obstacle avoidance is environment perception. Obstacle avoidance in an unknown environment requires sensors to obtain the surrounding environment information, including the shape and position of obstacles. Therefore, sensor technology plays a key role in obstacle avoidance of mobile robots. In this paper, the LIDAR sensor is used to obtain environmental information, and the laser is used to measure the distance between Pioneer robots and obstacles [12].

$$d = \frac{ct}{2} \tag{2}$$

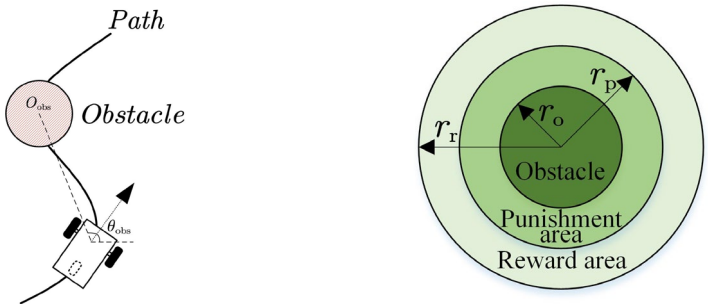
Where d is the distance, c is the speed of light, and t is the time interval from the transmission to reception.

The obstacle avoidance diagram of the Pioneer robot is shown in Figure 2(a). When the mobile robot moves on the path and recognizes obstacles ahead, it would judge whether the next action is to stop or turn to stay away from obstacles. The composition of the obstacle avoidance area of the Pioneer robot is shown in Figure 2(b). When the robot is away from obstacles, the robot's state is safe. When the mobile robot approaches the obstacle and enters the reward area, the robot's state is considered relatively safe. When the robot enters the punishment area, the robot would impact the obstacle. The obstacle avoidance model of the Pioneer robot is defined as follows:

$$d_{obs} = \begin{cases} d_o, & if\ d_o < r_r \\ 0, & if\ d_o \geq r_r \end{cases} \tag{3}$$

$$\theta_{obs} = \begin{cases} |\theta_{obs} - \theta|, & if\ d_o < \theta_e \\ 0, & if\ d_o \geq \theta_e \end{cases} \tag{4}$$

Where, d_{obs} and θ_{obs} are the state parameters for obstacle avoidance, d_{obs} represent the distance between the obstacle center and the Pioneer robot center.



(a) Obstacle avoidance diagram (b) Obstacle avoidance area composition diagram

Fig. 2. Pioneer robot obstacle avoidance diagram

3. Obstacle avoidance algorithm based on DRL

3.1 Smart agent

Deep reinforcement learning (DRL) is an intelligent algorithm that combines deep learning (DL) and reinforcement learning (RL). In this paper, we use recurrent neural networks to create DDPG smart agents approach to optimize path strategy. Our work includes the use of RNN-based DDPG, which is used to learn how to avoid obstacles and reach the target point efficiently. The main structure of the smart agent is shown in Figure 3. The agent is used to receive observations and rewards from the environment, and then send actions to the environment. The actor is a policy network in the agent. The reward function is to maximize the cumulative reward to improve path efficiency and obstacle avoidance efficiency.

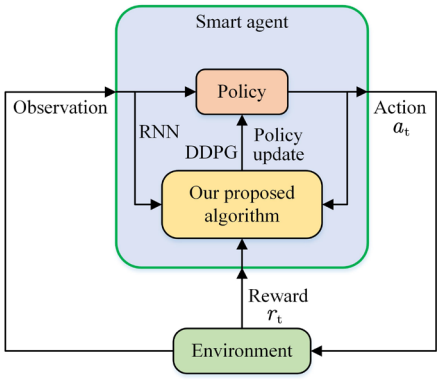
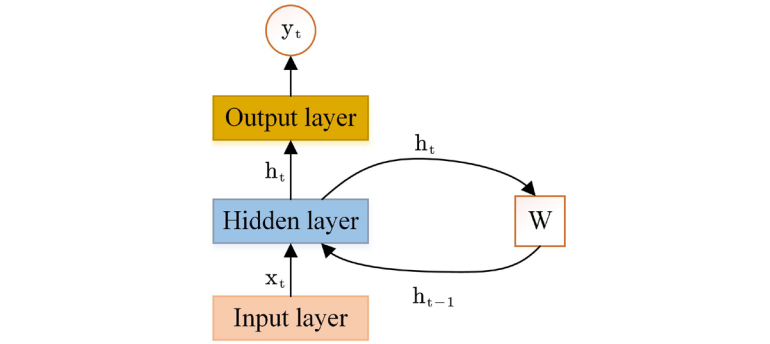


Fig. 3. Smart agent structure

The recurrent neural network is a kind of neural network to process sequence data, which can process serial data at multiple time steps. As shown in Figure 4(a), RNN consists of an input layer, a hidden layer, and an output layer [13]. The RNN cycle layer has a memory function and can extract time series information through parameter sharing at different moments. Therefore, the BP algorithm is used to train the RNN to minimize the loss to solve the gradient disappearance before the RNN update. The loss function is shown in Equation (7). Gradient disappearance means that the gradient in the hidden layer of the RNN is updated without using the previous information, and the previous gradient disappears due to the long distance. The RNN cycle structure is shown in Figure 4(b), where multiple RNN layers are the same layer and the same output is copied as its input.



(a) RNN cycle layer

(b) RNN cycle structure

Fig. 4. Recurrent neural network

$$y_t = \text{softmax}(h_t W + b) \tag{5}$$

$$h_t = \tanh(x_t W + h_{t-1} W + b) \tag{6}$$

Where, h_t is the state information stored in the memory at the current moment, and h_{t-1} is the state information stored in the memory at the last moment. x_t is the input feature at the current moment, y_t is the output feature at the current moment, W is the weight matrix, t is the time step, and b is the bias. tanh is the activation function, and softmax is the normalized exponential function of the fully connected network.

$$E(\theta) = \frac{1}{2}\sum_{i=1}^{2}(y_i - \hat{y}_i)^2 \tag{7}$$

where, E is the expected value of the loss function. θ is the set of parameters. y is the actual value, and $\hat{y}$ is the predicted value.

3.2 Our proposed algorithm

The structure of our proposed algorithm is shown in Figure 5, which first initializes actor network parameters and critic network parameters. Then select the action according to the strategy in the current state. The action is performed in the environment and transfers the state to obtain rewards [14]. Finally, the DDPG agent is created with recurrent neural networks. The Q-value function and actor network are approximated by RNN. The actor and critic networks are updated according to the formula. The DDPG algorithm can solve the continuous action control problem well [15]. When the action is continuous, the maximum gain is obtained while the specific action value can be output to obtain a deterministic strategy.

$$a_t = \mu(s_t|\theta^{\mu}) \tag{8}$$

$$R_t = \sum_{i=t}^{\infty} \gamma^{i-t} r_i \tag{9}$$

Where R is the reward function, γ is the discount rate, and r is the reward.

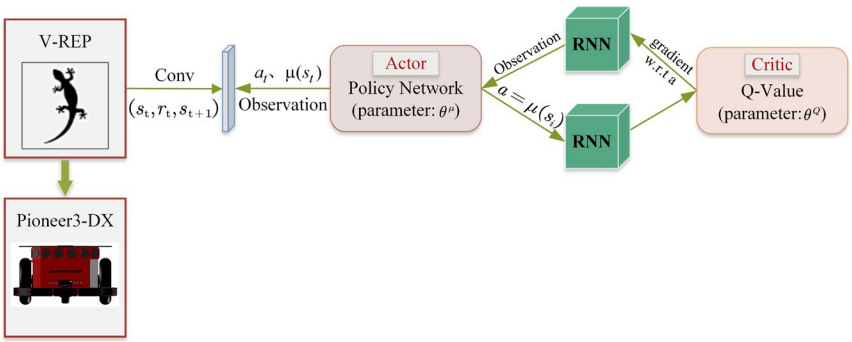


Fig. 5. Our proposed algorithm structure

First, a random batch is sampled from the replay buffer. The sampled data is (s, a, r, s'), and the action in the state s' is calculated using the target actor network as shown in Equation (10). The target value y is calculated using the target Critic network shown in Equation (11). The evaluation value q is calculated by using the Critic network shown in Equation (12). Minimize the difference between the evaluated value and the expected value by using the gradient descent method as shown in Equation (13).

$$a' = \mu'\left(s'\middle|\theta^{\mu'}\right) \tag{10}$$

$$y = r + \gamma Q'\left(s', a'\middle|\theta^{Q'}\right) \tag{11}$$

$$q = Q\left(s, a\middle|\theta^{Q}\right) \tag{12}$$

$$L = \left(y - q\right)^2 \tag{13}$$

Second, the soft target is used to update the target function and a learning rate is introduced. The old and new target network parameters are weighted mean. Then assign the value to the target network. The target actor network is obtained in Equation (14), and the target Critic network is obtained in Equation (15).

$$\theta^{\mu'} = \tau\theta^{\mu} + \left(1-\tau\right)\theta^{\mu'} \tag{14}$$

$$\theta^{Q'} = \tau\theta^{Q} + \left(1-\tau\right)\theta^{Q'} \tag{15}$$

Finally, due to the low exploration ability of the DDPG algorithm. The Ornstein Uhlenbeck (OU) noise is introduced to the output action. So that the Pioneer robot can explore the environment better. The stochastic differential equation of the OU process is shown in Equation (16).

$$dN_t = \theta\left(\mu - N_t\right)dt + \sigma dB_t \tag{16}$$

Where, μ is the mean value, σ and θ are parameters and both are greater than 0. B_t is the standard Brownian motion. N is the status. When θ is larger, the N_t larger and faster to approach the mean value. The covariance of the OU noise with Gaussian distribution is shown in equation (18).

$$N_t = \sigma\int_0^t e^{\theta(\tau-t)}dB_t \tag{17}$$

$$Cov(N_t, N_s) = \frac{\sigma^2}{2\theta}\left(e^{-\theta|t-s|} - e^{-\theta|t+s|}\right) \tag{18}$$

4. Simulation results and analysis

4.1 Simulation environment configuration

This simulation experiment is based on Python and virtual robot experimentation platforms (V-REP) to establish the simulation environment. The specific experimental configuration is shown in Table 1. V-REP is a dynamic simulation software, which is mainly used in the field of robot simulation modeling. A distributed control structure can be implemented using robot operating system (ROS) nodes and a remote application programming interface (API).

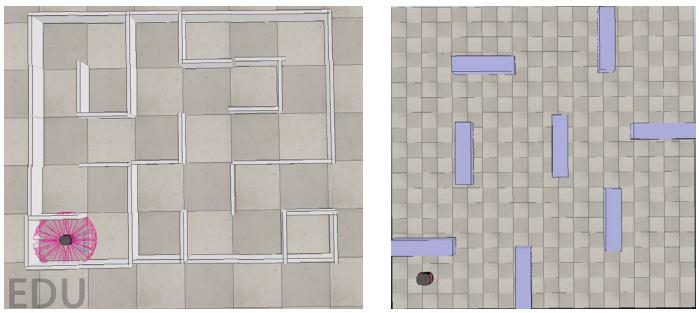
Table 1

Experimental configuration parameter

Name	Configuration
system	Windows 10
CPU	Interl(R) Core i7-11800H
GPU	NVIDIA GeForce RTX 3090
mobile robot	Pioneer3-DX
simulation platform	V-REP
algorithm environment	Python 3.9

4.2 Analysis of simulation result

In this paper, we build a simulation environment based on V-REP and simulate it interactively with Python through API. Two different V-REP obstacle avoidance scenarios are shown in Figure 6. As shown in Figure 6(a), V-REP scenario 1 is a maze map we built. As shown in Figure 6(b), V-REP scenario 2 is a fixed obstacle scene built with eight cuboids. Our proposed algorithm is applied to these two V-REP simulation environments to train the Pioneer robot for dynamic obstacle avoidance.



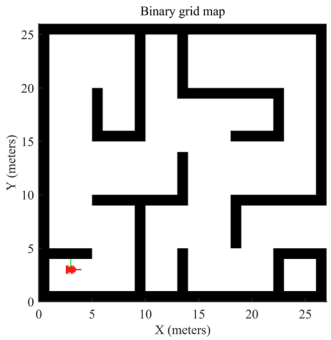
(a) V-REP scenario 1 (b) V-REP scenario 2

Fig. 6. Two different V-REP scenarios

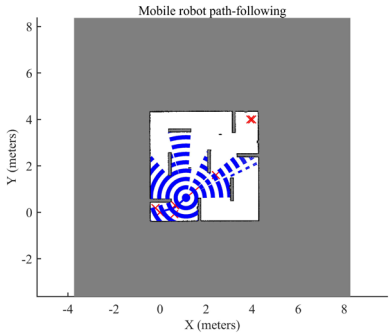
We build the three-dimensional model of the Pioneer mobile robot in SolidWorks. The unified robot description format (URDF) file is exported and the URDF model is imported into V-REP software. In Figure 6(a), the pink part around the robot is the set sensor module, and the white part is the maze map built with a 200cm×80cm wall. The joints of the robot are set for torque transmission. The blue part in Figure 6(b) is an obstacle built with eight cuboids. We combined V-REP with Python to train the mobile robot in two scenarios with our proposed algorithm. Set the learning gain of scene 1 to 0.5 and the robot moving speed to 0.7mm/s. Set the learning gain of scene 2 to 1 and the robot moving speed to 0.7 mm/s. When the robot's left wheel speed is greater than the right wheel speed, the robot turns right. Instead, the robot turns left.

The simulation results are shown in Figure 7. Figure 7(a) shows the binary grid map of V-REP scenario 1. We build the binary map of V-REP scenario 1 in MATLAB and set the sensor scanning parameters. We set the binary code of the solid black line in Figure 7(a) to 1 and the rest set code to 0. Figure 7(b) shows the visualization of path tracking for V-REP scenario 2. The mobile robot is equipped with a LIDAR, which enables it to scan for obstacles in front of it while it is working. The blue part in Figure 7(b) shows the LIDAR scanning state, where we can visualize the mobile robot path tracking.

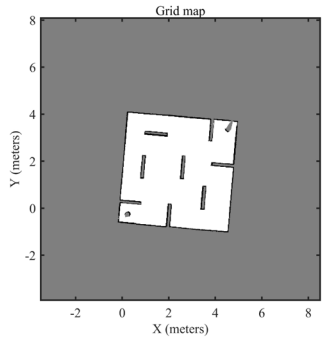
Figure 7(c) shows the two-dimensional grid map of V-REP scenario 2. Figure 7(d) shows the two-dimensional simultaneous localization and mapping (SLAM) map of V-REP scenario 2. We use the SLAM Map Builder application in MATLAB to create the SLAM map of V-REP scenario 2. We import the LIDAR scanned data and set the downsample to 100%. The data is sampled uniformly to reduce the computation time of the SLAM algorithm. To decrease the likelihood of accepting and using the detected loop closure, we set the loop closure threshold to 300. The optimization interval is set to 10 to optimize the pose graph. After filtering the data and setting SLAM parameters, the application begins to process the scanned data and finally builds the SLAM map as shown in Figure 7(d).



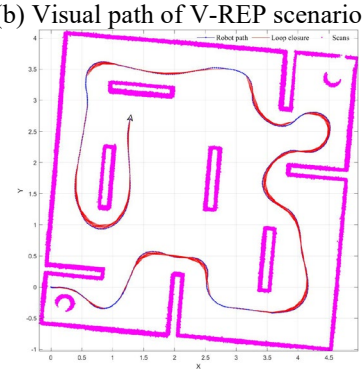
(a) Binary grid map of V-REP scenario 1



(b) Visual path of V-REP scenario 2



(c) 2D grid map of V-REP scenario 2



(d) 2D SLAM map of V-REP scenario 2

Fig. 7. Simulation results

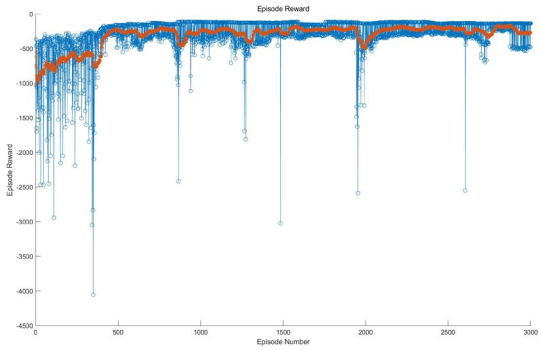
4.3 Our proposed algorithm training results

To make the model training more effective and converge faster, we pre-set the tuning parameters before starting the learning process. After several simulations, the hyperparameters of our proposed algorithm are set as shown in Table 2. These hyperparameters are chosen to ensure stable training of the neural network model and faster learning of data features. The maximum training set of our proposed algorithm is 3000, the maximum length of each set is 500, and the average length is 50.

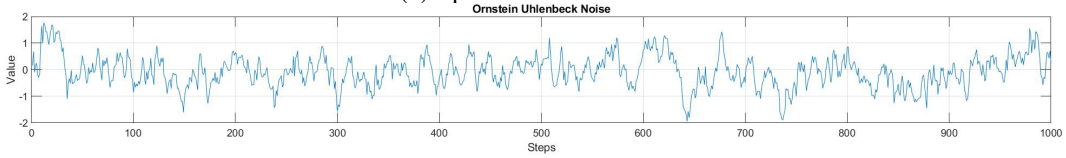
Table 2

The hyperparameter of our proposed algorithm

Hyperparameter	Value
learn rate for actor	0.0001
learn rate for critic	0.001
maximum episode	3000
maximum step	500
average window length	50
experience buffer	1×10^6
discount factor	0.995
batch size	128
OU noise	0.15



(a) Episode reward curve



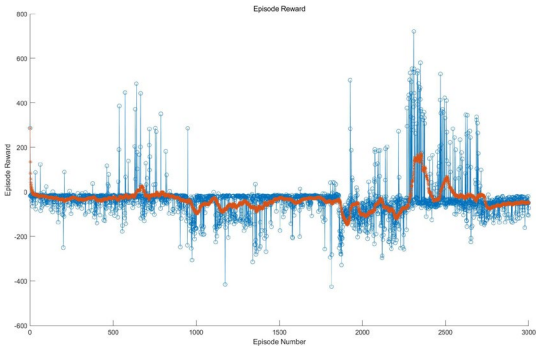
(b) OU noise

Fig. 8. Group 1 results of our proposed algorithm

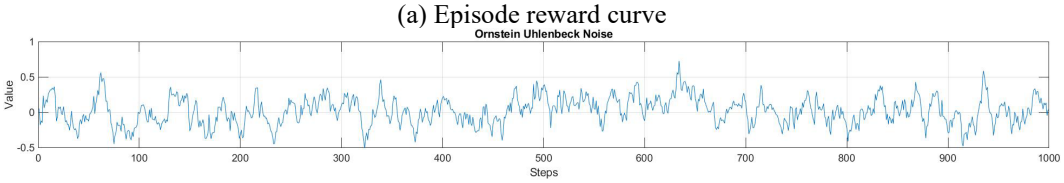
Our proposed algorithm has the shortest training time in group 1, and the training result is shown in Figure 8. Figure 8(a) shows the episode reward curve of our proposed algorithm. The solid blue line in Figure 8(a) is the episode reward, and the solid red line is the average reward. It can be seen from Figure

8(a) that the episode reward gradually goes to negative reinforcement and reaches the maximum set reward when the set is around 400. The average reward shows a stable state in general, and episode nodes are evenly distributed. The average reward is -272 and the average step is 130.2. Figure 8(b) shows the OU noise of our proposed algorithm. As can be seen from Figure 8(b), the difference between two adjacent steps of OU noise of our proposed algorithm is small and generally fluctuates around the mean value of 0.15.

To further verify the reliability of our proposed algorithm, our proposed algorithm is compared with the DDPG algorithm as shown in Table 3. The DDPG algorithm has the shortest training time in group 3, and the training result is shown in Figure 9. Figure 9(a) shows the episode reward curve of the DDPG algorithm. As can be seen from Figure 9(a), the episode reward fluctuates a lot and is distributed around -72.4. The average reward is -49.2 and the average step is 82.3. Figure 9(b) shows the OU noise of the DDPG algorithm. As can be seen from Figure 9(b), the mean value of OU noise of the DDPG algorithm is small, and the overall fluctuation is between -0.5 and 0.5. We further analyze the superiority of our proposed algorithm.



(a) Episode reward curve



(b) OU noise

Fig. 9. Group 3 results of the DDPG algorithm

We use the Reinforcement Learning Designer application in MATLAB to train the hyperparameters of our proposed algorithm. We import the created smart agent and the defined environment. As can be seen from Table 3, there are three groups for each algorithm. We select three optimal training results in each of the multiple simulations. The differences between our proposed algorithm and the DDPG algorithm are compared and analyzed. Our proposed algorithm has the

shortest training time in group 1, which takes 8630s. The DDPG algorithm has the shortest training time in group 3, which takes 11717s. Our proposed algorithm reduces the training time by 26.3% compared with the DDPG algorithm. The different training time in Table 3 is due to the randomness of the neural network algorithm, in which weights and thresholds are initialized randomly. Therefore, training the same network under the same conditions would get different results.

Table 3

Comparison between our proposed algorithm and the DDPG algorithm

Algorithm	Group	Episode reward	Average reward	Average step	Time
Our proposed	1	-136.5	-272	130.2	8630s
	2	-149.5	-164.6	73.8	10349s
	3	-222.5	-169.2	80	9596s
DDPG	1	-23.7	-63.2	74.1	12138s
	2	-56.8	-60.5	82.6	13449s
	3	-72.4	-49.2	82.3	11717s

5. Conclusions

In this paper, a method is proposed to create DDPG smart agents with recurrent neural networks to optimize the path strategy, which is based on a backpropagation algorithm to train RNN to train the agent. In dynamic and complex environments, the smart agent is used to learn how to avoid obstacles and reach the target point efficiently. Our proposed algorithm reduces the training time by 26.3% compared with the DDPG algorithm. Our proposed scheme can solve the problem of not finding paths in dynamic environments, thus improving the path efficiency of the Pioneer robot.

In the future, we plan to deploy our proposed scheme in actual experiments and optimize the smart agent neural network model. There are still many improvements that need to be done. First, the random initialization leads to different training results. This paper uses gradient descent to reduce the difference, but there are still subtle differences. We further use the random seed to increase certainty. Second, the use of one noise in this paper is not enough. Our next work studies a multi-step exploration method to improve the exploration ability of the DDPG algorithm. Third, we use a neural network with better performance instead of RNN for optimization.

Acknowledgement

This work was supported by the National Natural Science Foundation of China (62073239).

REFERENCES

[1]. *Y. Huang, J. Su.* "Visual servoing of nonholonomic mobile robots: A review and a novel perspective". IEEE Access, **vol. 7**, Sept. 2019, pp. 134968-134977.

[2]. *H. Zhang, Y. Zhu, X. Liu, et al.* "Analysis of obstacle avoidance strategy for dual-arm robot based on speed field with improved artificial potential field algorithm". Electronics, **vol. 10**, no. 15, Jul. 2021, pp. 1-17.

[3]. *Y. Tian, W. Feng, M. Ouyang, et al.* "A positioning error compensation method for multiple degrees of freedom robot arm based on the measured and target position error". Advances in Mechanical Engineering, **vol. 14**, no. 5, May. 2022, pp. 1-13.

[4]. *Z. Liu, H. Liu, Z. Lu, et al.* "A dynamic fusion pathfinding algorithm using Delaunay triangulation and improved A-star for mobile robots". IEEE Access, **vol. 9**, Jan. 2021, pp. 20602-20621.

[5]. *Q. Song, S. Li, J. Yang, et al.* "Intelligent Optimization Algorithm-Based Path Planning for a Mobile Robot". Computational Intelligence and Neuroscience, **vol. 2021**, Sept. 2021, pp. 1-17.

[6]. *S.M.H. Rostami, A.K. Sangaiah, J. Wang, et al.* "Obstacle avoidance of mobile robots using modified artificial potential field algorithm". EURASIP Journal on Wireless Communications and Networking, **vol. 2019**, no. 1, Mar. 2019, pp. 1-19.

[7]. *Q. Wu, Z. Chen, L. Wang*, *et al*. "Real-time dynamic path planning of mobile robots: a novel hybrid heuristic optimization algorithm". Sensors, **vol. 20**, no. 1, Dec. 2019, pp. 1-18.

[8]. *J. Choi, G. Lee, C. Lee.* "Reinforcement learning-based dynamic obstacle avoidance and integration of path planning". Intelligent Service Robotics, **vol. 14**, Oct. 2021, pp. 663-677.

[9]. *X. Peng, R. Chen, J. Zhang, et al.* "Enhanced Autonomous Navigation of Robots by Deep Reinforcement Learning Algorithm with Multistep Method". Sensors and Materials, **vol. 33,** no. 2, Nov. 2020, pp. 825-842.

[10]. *X. Cheng, S. Zhang, S. Cheng, et al.* "Path-Following and Obstacle Avoidance Control of Nonholonomic Wheeled Mobile Robot Based on Deep Reinforcement Learning". Applied Sciences, **vol. 12**, no. 14, Jul. 2022, pp. 1-14.

[11]. *M. Pan, J. Li, X. Yang, et al.* "Collision risk assessment and automatic obstacle avoidance strategy for teleoperation robots". Computers & Industrial Engineering, **vol. 169**, Jul. 2022, pp. 1-19.

[12]. *M.Á. Muñoz-Bañón, E. Velasco-Sánchez, F.A. Candelas, et al.* "OpenStreetMap-Based Autonomous Navigation With LiDAR Naive-Valley-Path Obstacle Avoidance". IEEE Transactions on Intelligent Transportation Systems, **vol. 23**, no. 12, Sept. 2022, pp. 24428-24438.

[13]. *R. Coban.* "A context layered locally recurrent neural network for dynamic system identification". Engineering Applications of Artificial Intelligence, **vol. 26**, no. 1, Jan. 2013, pp. 241-250.

[14]. *T. Schaul, J. Quan, I. Antonoglou, et al.* "Prioritized experience replay". arXiv preprint arXiv:1511.05952, Feb. 2016, pp. 1-21.

[15]. *T.P. Lillicrap, J.J. Hunt, A. Pritzel, et al.* "Continuous control with deep reinforcement learning". arXiv preprint arXiv:1509.02971, Jul. 2015, pp. 1-14.