Path Planning for Mobile Robots Based on TPR-DDPG

Yaping Zhao[1], Xiuqing Wang[1,2,3]* , Ruiyi Wang[1], Yunpeng Yang[1], Feng Lv[1]
[1]College of Computer and Cyber Security, Hebei Normal University, Shijiazhuang 050024, China
[2]Hebei Provincial Key Laboratory of Network & Information Security, Shijiazhuang, China
[3]Hebei Provincial Engineering Research Center for Supply Chain Big Data Analytics & Data Security, Shijiazhuang, China
Email: 1500943755@qq.com, xqwang2013@163.com, rywang1024@foxmail.com, 670292112@qq.com, lvfeng@mail.tsinghua.edu.cn
*Corresponding author

***Abstract*—Path planning is one of the key research topics in robotics. Nowadays, researchers pay more attention to reinforcement learning (RL) and deep learning (DL) because of RL's good generality, self-learning ability, and DL's super leaning ability. Deep deterministic policy gradient (DDPG) algorithm, which combines the architectures of deep Q-learning (DQN), deterministic policy gradient (DPG) and Actor-Critic (AC), is different from the traditional RL methods and is suitable for continuous action space. Therefore, TPR-DDPG based path planning algorithm for mobile robots is proposed. In the algorithm, the state is preprocessed by various normalization methods, and complete reward-functions are designed to make agents reach the target point quickly by optimal paths in complex environments. The BatchNorm layer is added to the policy network, which ensures the stability of the algorithm. Finally, experimental results of agents' reaching the target points successfully through the paths generated by the improved DDPG validate the effectiveness of the proposed algorithm.**

Keywords—path planning, deep deterministic policy gradient (DDPG), policy network, value network, mobile robots

I. Introduction

Mobile robots' path planning is to find an optimal or suboptimal smooth & obstacle-avoiding path from the initial point to the target point[1]. Obstacle avoidance, how to control robots to reach the target points, and how to keep robots' trajectories smooth and feasible, are key research topics for path planning.

Most traditional path planning algorithms have poor adaptability and are just suitable for the situations with environmental models. With reinforcement learning (RL), agents can optimize strategies through interacting with unknown environments by trial-and-error methods and accumulate rewards (the value of rewards can be positive or negative) to learn the optimal behavior pattern[2]. Besides, RL has good generality and is suitable for unknown environments. So it is necessary to introduce RL into robots' path planning. Tables based on Q-Learning algorithm[3] is a typical RL algorithm for path planning, which regards the environment as a coordinate system by a grid method, and divides actions to move up, down, left and right, and refines states into coordinate points. However, concerning the storage of Q tabular method, there are two problems to be solved: 1) some Q tables are too large to be saved; 2) sparse sampling leads to slow convergence or non-convergence.

In 2013, Google DeepMind combined deep learning with Q-Learning, using neural networks (NNs) to replace tables, and proposed the deep reinforcement learning (DRL) firstly, which is deep Q-learning (DQN)[4]. To some extent, DQN makes up for the deficiency of Q-learning state dimension. Compared with Q-learning, DQN introduces experience replay mechanism and target network, which ensures the stability of the algorithm. Through optimizing DQN, Google DeepMind proposed double deep Q-learning (Double DQN)[5], dueling deep Q-learning (Dueling DQN)[6] and prioritized experience replay (PER)[7]. Double DQN solves the overestimation problem of the evaluation function by optimizing the target Q value, Dueling DQN which is based on competing architecture, speeds up the convergence rate by optimizing the structure of the neural networks, and PER improves the experience pool by prioritizing experience replay. Although DQN can complete path planning, smoothness of planned paths is not satisfied. Highly discretizing action space for smooth paths causes dimensional disaster.

In 2015, Google DeepMind proposed deep deterministic policy gradient (DDPG)[8], which combines the architectures of DQN, deterministic policy gradient (DPG)[9] and Actor-Critic (AC) [10] , and is suitable for continuous action space. Actor and criticor in AC framework correspond to the policy network and the value network in DDPG respectively. In DDPG, actions are selected by DPG algorithm from policy network, and the rules of evaluation are developed through DQN algorithm from value network. The rules of evaluation are used to evaluate actions selected by the policy network, so that the policy network can select the optimal action. In reference [11], experimental results validate that trajectories generated by DDPG algorithm are smoother than that of the improved DQN algorithm. Most of the improvements for DDPG algorithm are focused on experience pools, for example, Liu Yandong[12] uses the learning curve to improve the sampling specifications of the experience pool in DDPG algorithm, so that the accumulative reward is increased by 30%. However, the reward function is so simple that the proposed DDPG algorithm in [12] is only effective in simple environment. Xiang Hui[13] improves the convergent rate by deleting the redundant samples in the experience pool and preferentially collecting the high-priority samples. In [13], actions are discretized and continuous-action-controlling characteristics of DDPG is not taken good use of, so DDPG in [13] is only suitable for simple environment, too. Yan Tingxing[14] used DDPG to realize obstacle avoidance

2021 International Joint Conference on Neural Networks (IJCNN) | 978-1-6654-3900-8/21/$31.00 ©2021 IEEE | DOI: 10.1109/IJCNN52387.2021.9533570

through punishing robots' collision behaviors in MobileSim robot simulation environment.

In this paper, TPR- DDPG is proposed which improved the reward function according to the three factors: the distance between robots and the target points, the situation of the obstacles in the environment and the relationship of robots' moving direction and the target point. Mobile robots can complete the end-to-end path planning in complex environments by TPR-DDPG successfully. The structure of this paper is as follow: Sec. 2 details the TPR-DDPG algorithm for mobile robots' path planning. Sec. 3 presents the experimental results and the analysis for the experiments. The paper is concluded in Sec. 4.

II. PATH PLANNING OF MOBILE ROBOT BASED ON TPR- DDPG ALGORITHM

A. Principle of TPR-DDPG algorithm

DDPG extends the experience replay mechanism and the target network of DQN. Experience replay refers to randomly batch sampling from the experience pool to make samples de-correlation. The target network can remove models' oscillation and divergence caused by the same parameters of models in the evaluation network and the target network. Compared with DQN, DDPG has improved the updating mode of target network in DQN, which improves the stability of the models of DDPG. DQN uses hard update, which directly assigns the evaluated parameters to the target network after certain numbers of training rounds. DDPG adopts soft update, which modifies the parameters of the target network by a small updating rate to improve the stability in the target network. In order to prevent the agent from only performing the output action of the current network, in the policy network, the DPG algorithm is used to select the action added noise to generate executed policy randomly, which expands the exploration range for agents.

The schematic of TPR-DDPG, which is composed of the evaluation network and the target network, is shown in Fig. 1. The architectures of the evaluation network and the target network are the same: both of them include the policy network and the value network. The states s_i and actions a_i from the experience pool are input into the evaluation value network to obtain $Q(a_i)$ .The next state s_{i+1} from the experience pool is input into the target policy network to obtain the action a' , the state s_{i+1} and action a' are put into the target value network to obtain $Q(a')$ .The mean squared error (MSE) function is used to update the parameters of the evaluation value network.

The state s_i from experience pool is input into the evaluation policy network to obtain the action a, and then the state s_i and action a are input into the evaluation value network to obtain $Q(a)$ .The policy gradient algorithm is used to update the parameters of the evaluation policy network.

θ represents parameters in the following networks including the target policy network, the target value network, the evaluation policy network and the evaluation value network. The main steps of TPR-DDPG are listed as follows:

Step 1: Initialize the evaluation policy network $\mu(s,\theta^{\mu})$ and the evaluation value network $Q(s,a|\theta^{Q})$, initialize corresponding target networks $\mu' \leftarrow \mu, Q' \leftarrow Q$, initialize experience pool *R*, and initialize the noise distribution *N*.

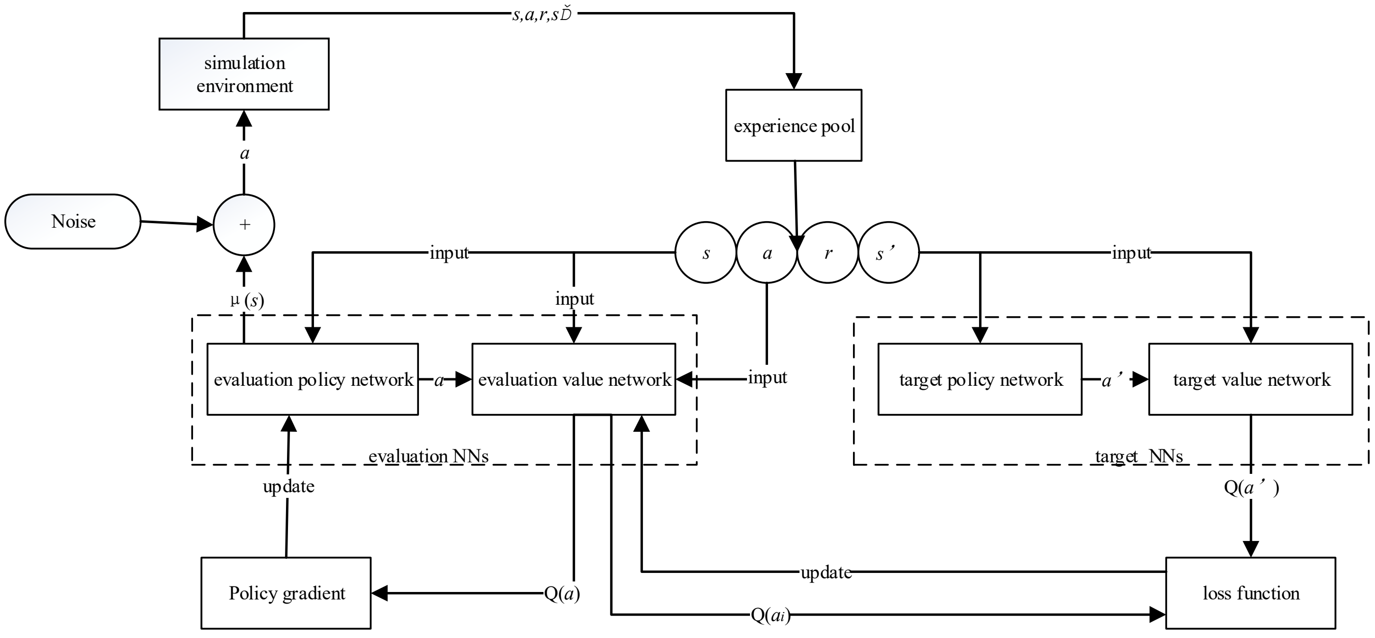


Fig. 1 Schematic of TPR-DDPG

Step 2: The determined action $\mu(s_t \mid \theta^u)$ is obtained by inputting the state s_t into the evaluation policy network, and the final action a_t is obtained through (1),. Then the reward r_t and state s_{t+1} are obtained by executing the action a_t.

$$a_t = (1\text{-}\alpha)\ \mu(s_t \mid \theta^u) + \alpha N_t \qquad (1)$$

where, α is the environment exploration rate and N_t is the standard normal distributed noise.

Step 3: Put s_t, a_t, r_t, s_{t+1} into the experience pool. When the number of the samples in the experience pool reaches a certain amount, a batch of samples are extracted from the experience pool and put into the network for training. If the number of the samples in the experience pool is saturated, the sample which was first put into the experience pool would be removed.

Step 4: The evaluation value network is updated according to (2), where L is MSE loss function and γ is the cumulative empirical discount rate.

$$L = \frac{1}{N}\sum_i (r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{u'})|\theta^{Q'}) - Q(s_i, a_i|\theta^Q))^2 \qquad (2)$$

Step 5: The evaluation policy network is updated by (3), in which ∇J is the policy gradient.

$$\nabla_{\theta^u} J = \frac{1}{N}\sum_i \nabla_a Q(s, a \mid \theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu}\mu(s \mid \theta^\mu)|_{s_i} \qquad (3)$$

Step 6: According to (4), the parameters are modified by the update rate τ in the target network.

$$\begin{aligned}\theta^{Q'} &\leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'} \\ \theta^{\mu'} &\leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'}\end{aligned} \qquad (4)$$

Step 7: Go to step 2, the action is obtained by the evaluation policy network, and the experience pool is updated. After that, a batch of samples are input into the network for training. Repeat the process from step 2 to step 6 until the number of training rounds equals to the *Max_episode*, and the process ends.

B. Structure of the Policy network

The function of policy network shown in Fig. 2 is to select actions. The 10 states including the distance from the agent to the target point, the azimuth angle of the agent and the sampling of the eight ultrasonic sensors in the front of the agents are input into the DDPG.

5 fully connected layers are used in the hidden layers, and the number of neurons in each hidden layers are 600, 400, 200, 20 and 1. In the first two layers of the neural networks, the activating function ReLU6 is used to make the output value of the layers is $\in$ [0,6] and improve the stability of the model. The next two layers use the activating function ReLU. In the first four hidden layers, a BatchNorm layer is added after the activating function.

The batch data x is normalized by (5), where μ is the mean value of batch data x and σ is the variance of batch data x. In (6), ρ is the scaling parameter, β is the translation parameter. ρ and β are used to approximate σ and μ, then y is reduced to x before normalization. That is, the scaling and translation reused before normalization, which means that BatchNorm does not work. The advantage of the above operation is that the data can be normalized to extract the features learned through fully-connection layer and the activating function.

$$x\_norm = \frac{x-\mu}{\sigma} \qquad (5)$$

$$y \leftarrow \rho x\_norm + \beta \qquad (6)$$

The output layer uses Tanh as the activating function and the output is $\in$ [-1, 1], If the output value is $\in [-1, 0)$, the agent will turn left, and if the output value is $\in$ [0, 1], the agent will turn right.

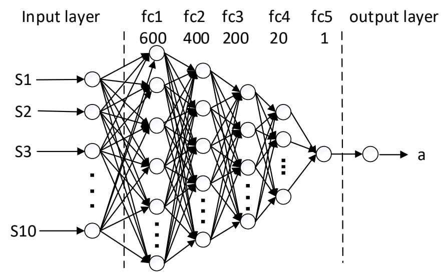


Fig. 2 Structure of the Policy network

C. Structure of the Value network

The function of value network is to evaluate the actions selected by policy network through Q value, as shown in Fig. 3, The distance from the agent to the target point, the azimuth angle of the agent, the readings of the eight ultrasonic sensors at the front of the agent, shown in Fig. 5, and the actions selected by the policy network are taken as the inputs of value network.

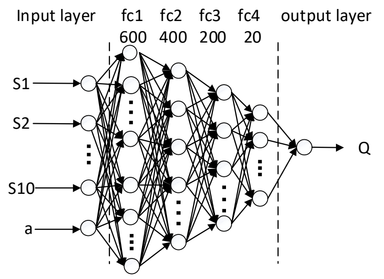


Fig. 3 Structure of the Value network

There are four fully connected layers in the hidden layers, and the number of neurons in the hidden layers are 600, 400, 200 and 20 respectively. ReLU6 and ReLU are used as the activating function in the first two hidden layers and the last two hidden layers respectively.

The output layer is a fully connected layer with only one output neuron to output Q value.

D. Preprocess of State

TABLE I. STATES OF TPR DDPG ALGORITHM

name	*range*	*description*
DisToGoal	$[0, x]$ mm	the distance from the agent to the target point, x is the distance from the initial point to the target point
Angle	$[-180°, 180°]$	rotation range of the agent's azimuth
Sonar	[0,5000] mm	8 readings from the front sonar sensors of the agent

The states are composed of the distance from the agent's current position to the target point, the azimuth angle and the 8 front sonars' sensors readings of the agent, as shown in Table 1. The values of different states are scaled by various preprocessing methods, and are normalized to range [0, 1]. The details are as follows:

States are preprocessed by (7), in which, s_p is the value after normalization, s_b is the value before normalization, and s_{max} is the largest state value.

$$s_p = \frac{\text{abs}(s_b)}{s_{\max}} \tag{7}$$

E. Reward Function

The reward function in (8) is a baseline to evaluate the action. If the agent reaches the target point, a maximum positive reward value, 50 is given, and if the agent collides with an obstacle, a negative reward value, -50 is given. If the agent neither collides nor reaches the target point, the *reward* consists of three parts: *reward1*, *reward2* and *reward3*, which are shown in equations from (9) to (11) respectively.

$$reward = \begin{cases} reward1 + reward2 + reward3, \text{not reach the target point} \cap \text{not collide} \\ 50, \text{ reach the target point} \\ -50, \text{ collide} \end{cases} \tag{8}$$

$$reward1 = (predis - curdis) / 50, \text{ not reach the target point } \cap \text{ not collide} \tag{9}$$

By the reward given in (9), an agent can approach the target point. *curdis* is the straight line distance from the position of the agent to the target point in the current step, and *predis* is the straight line distance from the position of the agent to the target point in the previous step. Since the maximum distance of the agent is 500 mm, *reward1* is ∈ [-10, 10], When *predis - curdis* ≥ 0 is satisfied, *reward1* is∈ [0, 10], which means that the agent is moving towards the target point, and when the agent is closer to the target point, the greater reward is given. Conversely, when *predis - curdis* < 0 is satisfied, *reward1* is$\in[-10,0)$, indicating that the agent is moving towards the opposite direction of the target point, and if the distance between the agent and the target point is larger, the reward will be smaller.

$$reward2 = \begin{cases} \text{-}curcount, precount \neq 0 \cap precount \leq curcount \\ 0, curcount = precount = 0 \end{cases} \tag{10}$$

By equation (10), an agent can avoid obstacles in the process of approaching the target point. *curcount* is the number of the four ultrasonic sensors' reading being less than 500 mm at the current sampling time, and *precount* is the number of the four ultrasonic sensors' reading being less than 500 mm at the previous sampling time. When precount ≤ curcount is satisfied, the risk of collision with obstacles at the current sampling time is greater than that at the previous sampling time, and the agent will be punished; otherwise, it means that the risk of collision with obstacles at the current sampling time is less than that at the previous sampling time, and the agent will be rewarded.

By the use of the reward given in (11), the time for the agent to reach the target point will be shorten. In (11), abs(*dir*) is the absolute value of azimuth. When *count* =0 is satisfied, it means that there is no obstacle within 500 mm in front of the agent. If 30 - abs(*dir*) ≥ 0 is satisfied, *reward3* is∈ [0, 9], and the agent moves towards the direction of the target point, the positive reward will be given; if 30 - abs(*dir*) < 0 is satisfied, *reward3* is $\in[-4.5,0)$, and the moving direction is opposite to the target point, the punishment will be given. The reward coefficient is set to 0.3 and the penalty coefficient is set to 0.03 to avoid missing the optimal path when there are obstacles in a large range.

$$reward3 = \begin{cases} 0.3 * (30 - \text{abs}(dir)), -30 \leq dir \leq 30 \\ 0.03 * (30 - \text{abs}(dir)), dir < -30 \cup dir > 30 \end{cases} \tag{11}$$

III. EXPERIMENTS AND ANALYSIS

A. Parameters of the robot and the environments

The Pioneer3-DX robot is used in the simulation platform-MobileSim, maps are designed by Mapper3, the development environment is Microsoft Visual Studio 2013 with the programming language C++, and the deep learning framework is Caffe.

Pioneer3-DX adopts two-wheel differential drive. The centroid c is in the center of the axis of the two driving wheels, the agent centroid coordinate is (x_c, y_c), the agent pose vector is (x_c, y_c, θ), and Pioneer3-DX kinematics model is shown in Fig. 4.

Equation (12) and (13) are the kinematic equations for the two-wheel-differential-drive mobile robots. v is the linear velocity, and ω is the angular velocity. v_l and v_r are the linear velocities of the left wheel and right wheel of the mobile robot respectively, θ is the azimuth angle, and l is the distance between the two driving wheels. The pose of the robot is q represented by $(x_c, y_c, \theta)^{\mathrm{T}}$.

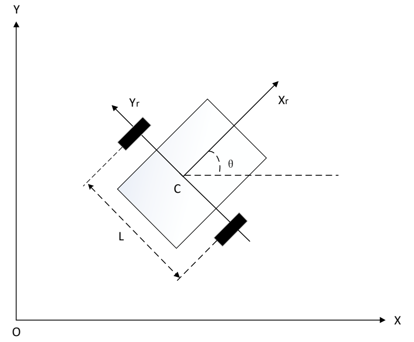


Fig. 4 Kinematics model for a differential-drive mobile robot

$$\begin{pmatrix} v \\ \omega \end{pmatrix} = \begin{pmatrix} \frac{1}{2} & \frac{1}{2} \\ \frac{1}{l} & \frac{-1}{l} \end{pmatrix} \begin{pmatrix} v_l \\ v_r \end{pmatrix} \tag{12}$$

$$\begin{cases} x(k+1) = x(k) + v\cos(\theta(k))\Delta t, \\ y(k+1) = y(k) + v\sin(\theta(k))\Delta t, \\ \theta(k+1) = \theta(k) + \omega(k)\Delta t, \end{cases} \tag{13}$$

Pioneer3-DX could have up to four sonar rings, and there are up to eight sonar transducers in each sonar ring for distance measuring and position sensing. Sonar sensors' range information is from 10 mm to 5000 mm. The location of each sonar sensor in the ring of Pioneer3-DX is fixed: there is one on both sides of the robot, and the other six are evenly distributed at 20-degree intervals. In the simulation, only eight sonars in the front of the agent are used, and the layout of the sonar sensors are shown in Fig. 5.

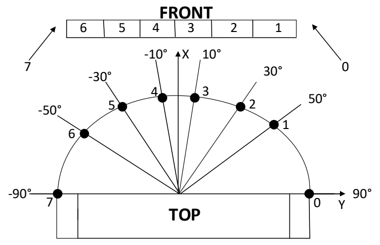


Fig. 5 Layout of the sonar sensors of Pioneer3-DX

The simulation environment is a square area of $12000 \times 8000\ mm^2$ with walls, as shown in Figures from 6 to 8. The complexity is increased from environment 1 to environment 4: In environment 1, there is no obstacle besides the walls; In environment 2, rectangular, polygonal and triangular obstacles are added compared with environment 1; In environment 3, some rectangle and square obstacles are added on the basis of environment 2; In environment 4, a wedge obstacle is added further.

B. Experimental parameters

The specific parameters of the experiment are shown in Table 2. Due to various complexity of the environments, the number of training episodes are different: the *Max_episode* are set to 300 rounds in environment 1 and environment 2; *Max_episode* are set to 600 rounds in environment 3 and environment 4. In the process of approaching the target point, it is unnecessary for the agent to explore the whole environment, and the selected action in the previous step has a direct impact on the action choose in the current step, so *Max_Step* is set to 50.

In the process of robots exploring the unknown environment, due to the uncertainty of the environments' complexity, the environment-exploration-rate α should be a small fixed value, and α is set to 0.2, aiming to allow an agent to explore in a small range. The experience pool is a container for storing samples, and each sample is composed of state s_t , action a_t ,reward r_t ,and the next state s_{t+1} . The experience pool size is set to 2000 to ensure the experience pool is large enough for the diversity of sampling.

To ensure the stability of the model, the update rate of the target network is usually set to 0.001. In order to prevent the gradient from oscillating around the local minimum value, which eventually leads to the failure of convergence, the learning rate of policy network and value network are set to 0.0001. The accumulated experience discount rate γ should not be too small. If γ is too small, previous experiences will be ignored and the target value network will not work, so γ is set to 0.95.

TABLE II. PARAMETERS OF TPR-DDPG ALGORITHM

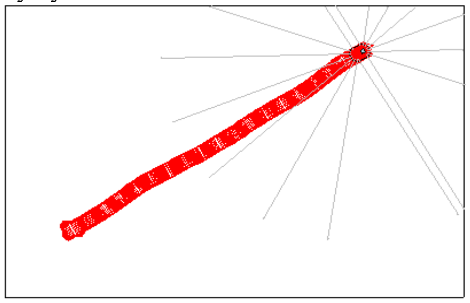
parameter	***Value***	***Description***
Max_episode	300、600	Number of training rounds
Max_Step	50	Maximum steps per round
Batch	24	Number of samples taken from the experience pool per round
γ	0.95	Cumulative experience discount rate
τ	0.001	Target network update rate
Memory_size	2000	Experience pool size
α	0.2	Environmental exploration rate
Lr	0.0001	learning rate of the policy network and the value network

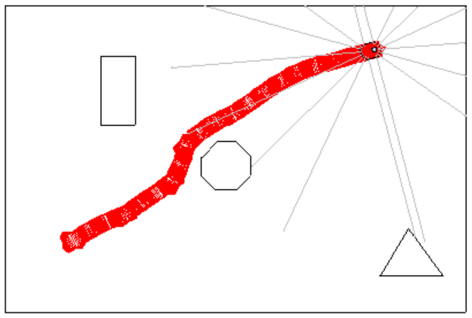
C. Experimental results in various environments

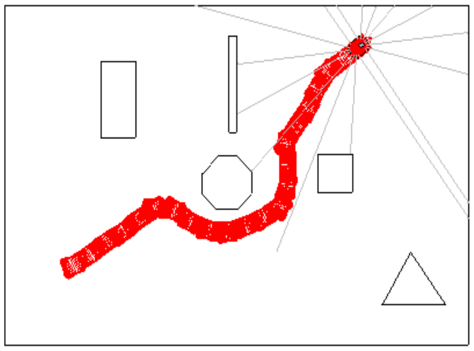
In order to evaluate the reward function, the average reward of each step is calculated by (14), *R* is the accumulated reward, *count* is the number of executed steps, and *r_last* is the maximum reward when the agent reaching the target point.

$$r = \frac{R - r\_last}{count - 1} \tag{14}$$

The experimental results of the same initial pose and target point in various environments with different complexity are shown in Fig. 6 (the initial pose q is $(-3200, -3100, 0^{\circ})$ and the target point is (4600, 2000)). In Fig. 6 (d), the agent takes 22 steps to reach the target point, the accumulated reward is 287.2, and the average reward is 11.3. Compared with environment 3, the newly added wedge-shaped obstacle in environment 4 increased the complexity of the environment at the initial point, and the agent approaches the target point along the lower right side of the wedge-shaped obstacle, so more episodes are used. From the above experimental results, it can be seen that a robot with the same initial pose can approach the same target point successfully in the environments of various complexity by TPR-DDPG.


(a) In Environment 1


(b) In Environment 2


(c) In Environment 3

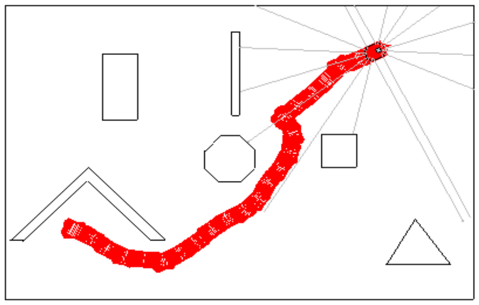
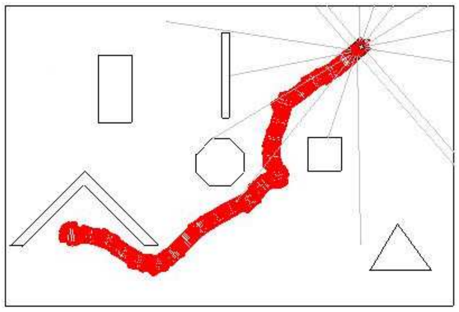
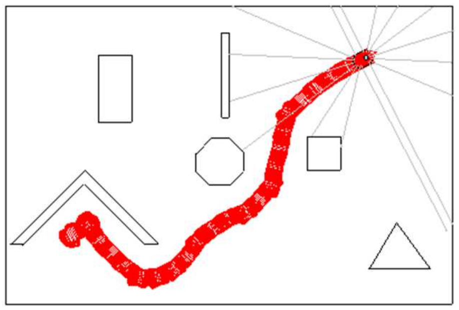

(d) In Environment 4

Fig. 6 Experimental results for robots with same initial pose and the same target point in various environments

D. *Experimental results for robots with various initial pose and the same target point*

The experimental results of the same starting point and target point with various initial azimuths in environment 4 are shown in Fig. 7 - (the initial point is (-3200, -3100), the target point is (4600, 2000) and the initial azimuths are -90°, 90° and 180° respectively). In Fig. 7 (a), the initial azimuth


(a) Initial azimuth is -90°


(b) Initial azimuth is 90°

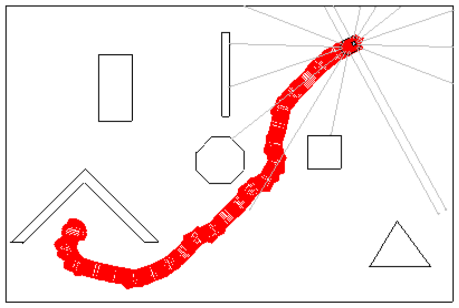
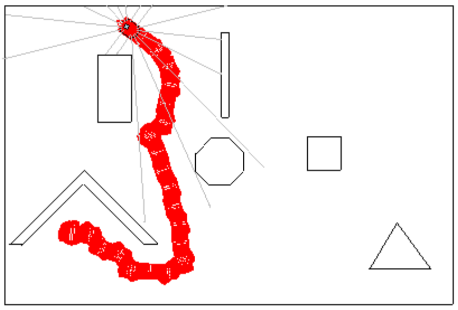

(c) Initial azimuth is 180°

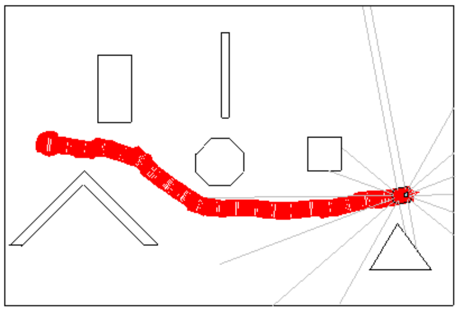
Fig. 7 Experimental results for various initial azimuths of the robot from the same starting point to the same target in environment 4

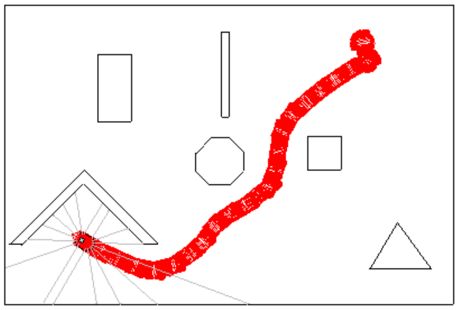
is -90°, the agent takes 22 steps to reach the target point, the accumulated reward is 287.2, and the average reward is 11.3. In Fig. 7 (b), the initial azimuth is 90°, the agent takes 23 steps to reach the target point, the accumulated reward is 270.3, and the average reward is 10. In Fig. 7 (c), the initial azimuth is 180°, the agent takes 23 steps to reach the target point, the accumulated reward is 267.2, and the average reward for each step is 9.9. From Fig. 7 (a) to Fig. 7 (c) the average reward for the agent and the planned path are similar, except for the paths near the initial points. The reason why the paths at the initial point are slightly different is that the optimal action chosen by the agent varies with different initial azimuths. From the experimental results, it can be concluded that the TPR-DDPG algorithm is effective for robots with various initial azimuths to find optimal paths in complex environments.

E. Experimental results for robots with various initial poses and various target points

The experimental results for different initial pose and target point in environment 4 are shown in Fig. 8. In Fig. 8 (a), the agent takes 21 steps to reach the target point, the accumulated reward is 164, and the average reward is 5.7. Compared with the average reward in other cases, the average reward is small, because *reward1* and *reward3* (referred to (9) and (11)) are negative in the area under the wedge-shaped obstacle. In Fig. 8 (a), the reason why the partial path under the wedge-shaped obstacle is not smooth is that the moving direction for the agent to avoid wedge-shaped obstacle is opposite to the direction for approaching the target point. After completing obstacle avoiding task, the agent tries to approach the target point again. However, in this study only eight front sonar sensors are used, the agent cannot detect the obstacles behind it, so it moves into the obstacle area again. The agent takes many steps to avoid obstacles and move to the target point repeatedly under wedge-shaped obstacle; In Fig. 8 (b), the agent takes 20 steps to reach the target point, the accumulated reward is 318.5, and the average reward is 14.1; In Fig. 8 (c), the agent takes 21 steps to reach the target point, and the accumulated reward and the average reward are 265.5 and 10.8. In Fig. 8 (c), near the initial point, the selected action is not optimal, that is because the maximum rotation angle of the agent is set as 90 degree and the initial azimuth of the agent is 0 degree, and the first selected action is turning right by 87 degree; In Fig. 8 (d) , the agent takes 18 steps to reach the target point, the accumulated reward is 214, and the average reward is 9.6; From the above experimental results, the following conclusions can be drawn: TPR-DDPG algorithm can complete the path planning for robots with various initial poses to approach different target points successfully in complex environments, which verifies the effectiveness of the algorithm.


(a) Initial q is (-3200, -3100, -90°) and the target point is (2000，2300)


(b) Initial q is (-3800，-700，-90°) and the target point is (6000，-1800)


(c) Initial q is (4600，2000，0°) and the target point is （-3200，-3100）

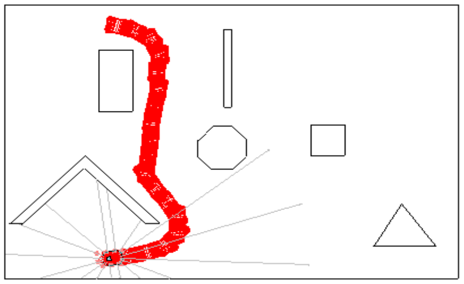

(d) Initial q is (-2000，2400，0°) and the target point is (-2500，-4500)

Fig. 8 Experimental results for various initial poses and target points of the robot in environment 4

F. Comparisions for the Experimental results for Q-learning and TPR-DDPG

The experimental results of Q-learning [15] and TPR-DDPG by MobileSim simulation platform are shown in Fig. 9 and Fig. 10 respectively. In Figure 10, after 21 training steps to reach the target, the accumulated reward is 285. The rewarding curve of each episode in the whole training process is shown in Figure 11. The fluctuation of the curve is due to the addition of 20% noise in the selection of actions. Although the simulation platform and environment are the same, it is clearly that the trajectory of TPR-DDPG is much smoother than that in Fig. 9. The reasons why the trajectories generated by TPR-DDPG are better than that of Q-learning in [15] are as follows: firstly, the action space of Q-learning is discretized, while the action space in TPR-DDPG is continuous. The continuous action space leads to smoother trajectory. Secondly, the reward function of TPR-DDPG, which includes three parts, is more complete and feasible than that in [15]. The three-part-reward is: *reward1*, *reward2* and *reward3*. *reward1* is decided by the distance between the robots and the target point; *reward2* is decided by the situation related with the obstacles within the diameter of 500mm, and *reward3* represents whether the robot is moving towards the target or not (the details of the three-part-reward can be referred to (9-11)). From the above analysis, conclusions can be drawn that TPR-DDPG has better effect in mobile robots' path planning than traditional Q-learning.

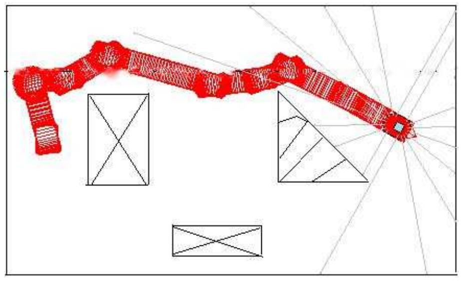


Fig. 9 Experimental result of Q-learning [15]

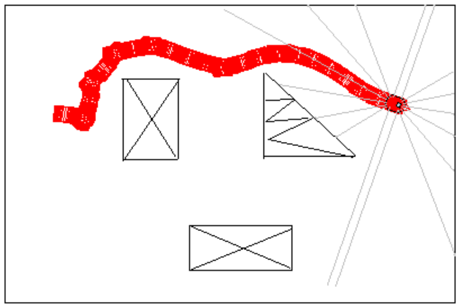


Fig. 10 Experimental result of TPR-DDPG

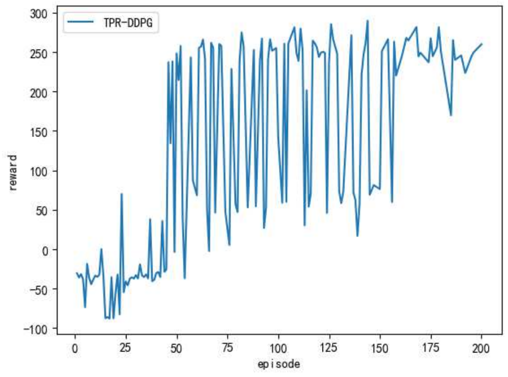


Fig.11 Accumulated reward of TPR-DDPG

IV. CONCLUSION

In the TPR-DDPG algorithm, effective policy networks and value networks are designed, and various preprocessing methods are used to normalize states. The distance between the last step and the current step, the azimuth of the robot and the obstacles in the environment are fully considered, and a complete three-part-reward function is designed for the target point approaching task of the mobile robot. Experimental results verify the effectiveness of the proposed algorithm.

ACKNOWLEDGMENTS

This work was supported in part by Natural Science Foundation (NSF) of Hebei Province under contract (Grants No. F2018205102), Natural Science Foundation of China under contract (Grants No. 61673160), Foundation of Science and Technology Project of Hebei Education Department (Grants No. ZD2021063), Key Foundation of Hebei Normal University (Grants No. L2019Z11).

REFERENCES

[1] G. Xu, X. Zong, G. Yu, and H. Su, " Research on intelligent obstacle avoidance method of unmanned vehicle based on DDPG,"[J]. *Automotive Engineering,* vol. 41, no. 02, pp. 90-96, 2019.

[2] H. Li, and Y. Qi, "A robot path planning method based on deep reinforcement learning in complex environment ,"[J]. *Application Research of Computer,* vol. 37, no. S1, pp. 129-131, 2020.

[3] Watkins, Christopher J. C. H. , and P. Dayan, "Q-learning,"[J]. Machine Learning, vol. 8, no. 3-4, pp. 279-292, 1992.

[4] V. Mnih, K. Kavukcuoglu, D. Silver, and A. Graves, "Playing Atari with Deep Reinforcement Learning,"[J].*arXiv preprint arXiv:1312.5602*, *2013,*

[5] H. Van Hasselt, A. Guez, and D. J. C. S. Silver, "Deep Reinforcement Learning with Double Q-learning," [J].*arXiv preprint arXiv:1509.0646,* 2015.

[6] Z. Wang, N. D. Freitas, and M. Lanctot, "Dueling Network Architectures for Deep Reinforcement Learning, " [J].arXiv preprint arXiv:1511.06581, 2016.

[7] T. Schaul , J. Quan , I. Antonoglou , D. Silver, "Prioritized Experience Replay,"[J].*arXiv preprint arXiv:1511.05952*, 2015.

[8] T.P. Lillicrap, J.J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa , D. Silver , D. Wierstra, "Continuous control with deep reinforcement learning,"[J]. *Computer science,* vol. 8, no. 6, pp.187, 2015.

[9] D. Silver , G. Lever , N. Heess , T. Degris , M. Riedmiller , "Deterministic Policy Gradient Algorithms," [C]. *ICML,* pp.387-395, 2014.

[10] T. Degris, M. White, R.S. Sutton, "Off-Policy Actor-Critic," arXiv preprint arXiv:1205.4839,2013.

[11] J. Yu, Y. Su, and Y. Liao, "The Path Planning of Mobile Robot by Neural Networks and Hierarchical Reinforcement Learning,"[J]. *Frontiers in Neurorobotics,* vol. 14, pp.14-63,2020.

[12] Y.D. Liu, "Path planning of mobile robot based on DDPG reinforcement learning,"[D].Inner Mongolia: Inner Mongolia Polytehnic University, 2019.

[13] H. Xiang, "Research on indoor target path planning based on deep reinforcement learning,"[D]. GuiLin:GuiLin University of Electonic Technology, 2019.

[14] Y.Yan, "Design of path planning method for mobile robot based on Reinforcement Learning," [D]. JiNan: University of JiNan,2019.

[15] S Li, X Wang, L Hu, Y Liu. "Mobile robot path planning based on Q-learning algorithm,"[C]. WRC SARA. 2019.