Hindawi
Journal of Robotics
Volume 2018, Article ID 5781591, 10 pages
https://doi.org/10.1155/2018/5781591

Research Article

Dynamic Path Planning of Unknown Environment Based on Deep Reinforcement Learning

Xiaoyun Lei, Zhian Zhang, and Peifang Dong

Key Laboratory of Intelligent Ammunition Technology, School of Mechanical Engineering, Nanjing University of Science and Technology, Nanjing 210094, China

Correspondence should be addressed to Zhian Zhang; zzayoyo@163.com

Received 11 July 2018; Revised 6 August 2018; Accepted 2 September 2018; Published 18 September 2018

Academic Editor: Gordon R. Pennock

Dynamic path planning of unknown environment has always been a challenge for mobile robots. In this paper, we apply double Q-network (DDQN) deep reinforcement learning proposed by DeepMind in 2016 to dynamic path planning of unknown environment. The reward and punishment function and the training method are designed for the instability of the training stage and the sparsity of the environment state space. In different training stages, we dynamically adjust the starting position and target position. With the updating of neural network and the increase of greedy rule probability, the local space searched by agent is expanded. Pygame module in PYTHON is used to establish dynamic environments. Considering lidar signal and local target position as the inputs, convolutional neural networks (CNNs) are used to generalize the environmental state. Q-learning algorithm enhances the ability of the dynamic obstacle avoidance and local planning of the agents in environment. The results show that, after training in different dynamic environments and testing in a new environment, the agent is able to reach the local target position successfully in unknown dynamic environment.

1. Introduction

Since deep reinforcement learning [1] was first proposed in 2013 formally, tremendous progress has been made in the field of artificial intelligence [2]. Deep Q-network agent was demonstrated to be able to surpass the performance of all previous algorithms and achieve a level comparable to that of a professional human games tester in Atari 2600 games [3–7]. AlphaGo zero [8, 9] has defeated all previous AlphaGo versions by self-play without using any human chess spectrum. An agent can be trained to play FPS game receiving only pixels and game score as inputs [10]. The aforementioned examples fully demonstrate the great potential in autonomous decision-making field after the reinforcement learning of neural network solving the problem of the curse of dimensionality. In [11, 12], deep reinforcement learning has been applied to autonomous navigation based on the inputs of visual information, which has achieved remarkable success. In [11], Piotr Mirowski et al. highlighted the utility of un/self-supervised auxiliary objectives, namely, depth prediction and loop closure, in providing richer training signals that bootstrap learning and enhance data efficiency. The authors analyze the agent behavior in static mazes feature complex geometry, random start position and orientation, and dynamic goal locations. Their results show that their approach enable the agent navigate within large and visually rich environments that include frequently changing start and goal locations, but the maze layout itself is static. The authors did not test the algorithm in environments with moving obstacles. If there are moving obstacles in environments, this means the images collected by cameras may be unknown for each episode. In [12], Yuke Zhu et al. try to find the minimum length sequence of actions that move an agent from its current location to a target that is specified by an RGB image. To solve the problem of a lack of generalization, i.e., the network should be retrained for new targets, they specify the task objective (i.e., navigation destination) as inputs to the model and addresses problems by introducing shared Siamese layers to the network. But as it is mentioned in the paper, in the network architecture of deep Siamese actor-critic model, the ResNet-50 layers are pretrained on ImageNet and fixed during training. This means that they have to collect a large number of different target and scene images with a constrained background to pretrain the ImageNet before training the navigation model, indicating that the generalization ability is still conditioned to the information of maps in advance.

In this paper, we present a novel path planning algorithm and solve the generalization problem by means of local path planning with deep reinforcement learning DDQN based on lidar sensor information. In the aspect of the recent deep reinforcement learning models, the original training mode results in a large number of samples which are moving states in the free zone in the pool, and the lack of trial-and-error punishment samples and target reward samples ultimately leads to algorithm disconvergence. So, we constrain the starting position and target position by randomly setting target position in the area that is not occupied by the obstacles to expand the state space distribution of the pool of sample.

To evaluate our algorithm, we use TensorFlow to build the DDQN training frameworks for simulation and demonstrate the approach in real world. In simulations, the agent is trained in a lower-level and intermediate dynamic environment. The starting point and target point are randomly generated to ensure diversity and complexity of local environment, and the test environment is a high-level dynamic map. We show details of the agent's performance in an unseen dynamic map in the real world.

2. Deep Reinforcement Learning with DDQN Algorithm

The conventional Q-learning algorithm [1] cannot effectively plan a path in random dynamic environment because of the lack of generalization ability and a large Q table. To solve the problem of the curse of dimensionality in high dimensional state space, the optimal action value function Q in Q-learning can be parameterized by an approximate value function.

$$Q(s,a;\theta) \approx Q^{*}(s,a) \tag{1}$$

where θ is the Q-network parameter. We approximate the value of Q in a definite environment state by function equation, which is mainly linear approximation; thus, there is no need to build a large Q table to determine the corresponding Q values for different state-actions. Neural network is used to approximate the linear function which can obtain nonlinear approximation with generalization ability. (s,a) is regarded as the input of the neural network and the output is the value of Q, where θ is the weight of Q-network. Q table is replaced by Q-network. The training process is to constantly adjust the network weights to reduce the bias of the output of Q-network and the target value of Q. Assuming that the target value of Q is denoted by y, thus the loss function of Q-network is yielded:

$$L_i(\theta_i) = E_{s,a}\left[\left(y_i - Q(s,a;\theta_i)\right)^2\right] \tag{2}$$

where iteration i is the current iteration times and (s,a) is a pair of state-action.

The update mode of the value of Q is similar to that of Q-learning; that is,

$$y_i = E_{s'\sim\varepsilon}\left[r + \gamma \max_{a'} Q\left(s',a';\theta_{i-1}\right) \mid s,a\right] \tag{3}$$

where r is the current reward and γ is a discount factor.

Stochastic gradient descent algorithm is adopted to train the neural network. According to the back propagation of the derivative value of loss function, the network weights are constantly adjusted, resulting in the network output approaching the target value of Q. The network gradient can be derived according to (2) and (3).

$$\begin{aligned}\nabla_{\theta_i} L_i(\theta_i) &= E\left[\left(r + \gamma \max_{a'} Q\left(s',a';\theta_{i-1}\right) - Q(s,a;\theta_i)\right)\right. \\ &\quad \left.\cdot \nabla_{\theta_i} Q(s,a;\theta_i)\right]\end{aligned} \tag{4}$$

Equation (4) indicates that the updates of neural network and Q-learning are simultaneous. The Q-network of the current iteration epoch is updated with the target value of Q updated from the previous iteration epoch.

Reinforcement learning is known to be unstable or even to diverge when a nonlinear function approximator such as a neural network is used to represent the action value (also known as Q) function [13]. This instability has several causes: the correlations present in the sequence of observations and the correlations between the action values (Q) and the target values. To address these instabilities, DeepMind presents a biologically inspired mechanism termed experience replay [14–16] that randomizes over the data. To perform experience replay, DeepMind stores the agent's experiences, environment state, action, and reward at each time-step t in a dataset. The pool of data samples is extended with running e-greedy policy to search the environment. During the learning, train the network with random data from the pool of stored samples, thereby removing correlations in the observation sequence and improving the stability of algorithm.

In 2015, DeepMind improved the original algorithm in the paper "Human-Level Control through Deep Reinforcement Learning" published in *Nature*. They added a target Q-network, which is delayed updating in comparison with the predicted Q-network, to update the target values of Q, thereby restraining a big bias of Q resulting from a fast dynamic updating of the target Q-network. The improved loss function is

$$L_i(\theta_i) = E\left[\left(r + \gamma \max_{a'} Q\left(s',a';\theta_i^{-}\right) - Q(s,a;\theta_i)\right)^2\right] \tag{5}$$

where θ_i are the parameters of the Q-network at iteration i and θ_i^{-} are the network parameters used to compute the target at iteration i. The target network parameters θ_i^{-} are only updated with the Q-network parameters θ_i every C steps and are held fixed between individual updates, thus ensuring that the update of target network is delayed and the estimate of Q is more accurate.

The update mode of Q-learning algorithm results in a problem of overestimate action values. The algorithm estimates the value of a certain state too optimistically, consequently causing that the Q value of subprime action is greater than that of the optimal action, thereby changing the optimal action and affecting the accuracy of the algorithm. To address the overestimate problem, in 2016, DeepMind presented an improved algorithm in the paper "Deep Reinforcement Learning with Double Q-Learning", namely, Double Q-network algorithm [17], which selected an action by maximizing the value of the predicted Q-network and updating the predicted Q-network with the selected action value in Q-network instead of directly updating the predicted Q-network with the maximum value of target Q-network.

$$y_i = E\left[r + \gamma Q\left(s', \arg\max\left(Q\left(s', a'; \theta_i\right)\right); \theta_i^-\right) \mid s, a\right] \quad (6)$$

The loss function of the improved algorithm is

$$L_i\left(\theta_i\right) = E\left[\left(r + \gamma Q\left(s', \arg\max\left(Q\left(s', a'; \theta_i\right)\right); \theta_i^-\right) - Q\left(s, a; \theta_i\right)\right)^2\right] \quad (7)$$

The framework illustration of DDQN is shown in Figure 1.

3. Local Path Planning with DDQN Algorithm

In this paper, we achieve local path planning with deep reinforcement learning DDQN. Lidar is used to detect the environment information over 360 degrees. The sensor range is regarded as observation window. The accessible point nearest to the global path at the edge of the observation window is considered as the local target point of the local path planning. The network receives lidar dot matrix information and local target point coordinates as the inputs and outputs the direction of movement.

Considering the computation burden and actual navigation effect, we set angle resolution to be 1 degree and range limit to be 2 meters so each observation consists of 360 points indicating the distance to obstacles within a two-meter circle around the robot. The local target point is the intersection point of the observation window circular arc and the global path. If there were several intersection points, the optimal point would be chosen by heuristic evaluation.

As to the lidar dot matrix information, the angle and distance can be denoted by $[angle, distance]$. We set a rule that the angles of the lidar points increases clockwise relative to the front of the mobile robot. 360 points result in 720 inputs of data. The relative coordinate of local target point and the current center coordinate of the mobile robot body ($[\Delta x, \Delta y]$) are also regarded as the input. To achieve the network output weights of the relative target points in training and conveniently design the convolutional network, we make 40 copies of the relative target points as the inputs; namely, the total inputs are 800 datasets.

The outputs are fixed-length omnidirectional movements of eight directions, where the movement length is 10 cm, and the directions are forward, backward, left, right, left front, left rear, right front, and right rear, which are denoted by 1 to 8 in order as follows.

$$\boldsymbol{a} = \left[a_1, a_2, a_3, a_4, a_5, a_6, a_7, a_8\right] \quad (8)$$

The design of positive activation function mainly accounts for obstacle avoidance and approaching the target point. The shortest movement path which satisfied the two conditions is the most effective. The rewards and punishments function is designed as

$$r = \begin{cases} 1\left(p\left(x_1, y_1\right) = g\left(x, y\right)\right) \\ -0.01\begin{pmatrix} p\left(x_1, y_1\right) \neq g\left(x, y\right) \\ p\left(x_1, y_1\right) \neq o\left(x, y\right) \end{pmatrix} \\ -1\left(p\left(x_1, y_1\right) = o\left(x, y\right)\right) \end{cases} \quad (9)$$

where p is the current position, g is a local target point, and o represents an obstacle with expansion processing. Equation (9) indicates that the reword would be -1 if the agent encountered an obstacle; if the agent arrived at the target point, it would get a reward +1; in the other states, each step of movement would cost -0.01. During the training, in order to continuously improve the cumulative reward, the agent gradually avoids the obstacle and reaches the target point in a short path by the method of the trial-and-error learning.

Each lidar detected point receives continuous measurement values changing from 0 cm to 200 cm, which means that the state space tends towards infinitude and it is impossible to represent all of the states with Q table. With DDQN, the generalization ability of neural network makes it possible that the network is able to approach all states after training; therefore, when the environment changed, the agent could plan a proper path and arrive at the target position according to the weights of the network.

To ensure the deep reinforcement learning training converging normally, the pool should be large enough to store state-action of each time-step and keep the training samples of neural network be independent identical distribution; besides, the environment punishment and reward should reach to a certain proportion. If the sample space was too sparse, namely, the main states were random movements in free space, it was difficult to achieve a stable training effect. As to the instability of DDQN in training and the reward sparsity of the state space, the starting point is randomly set in the circle with the target point as the center and the radius L. The initial value of L is small, thereby increasing the probability that the agent reaches the target point from the starting point in the random exploration and ensuring a positive incentive in the sample space. The value of L gradually increases with the update of neural network and the increase of greedy law probability. The local space searched by agent is expanded as follows.

$$L = \begin{cases} i_{\min} & \left(n \leq \mathrm{N}_1\right) \\ i_{\min} + \sqrt{\dfrac{\left(n - \mathrm{N}_1\right)}{m}} & \left(\mathrm{N}_1 < n < \mathrm{N}_2\right) \\ i_{\max} & \left(n \geq \mathrm{N}_2\right) \end{cases} \quad (10)$$

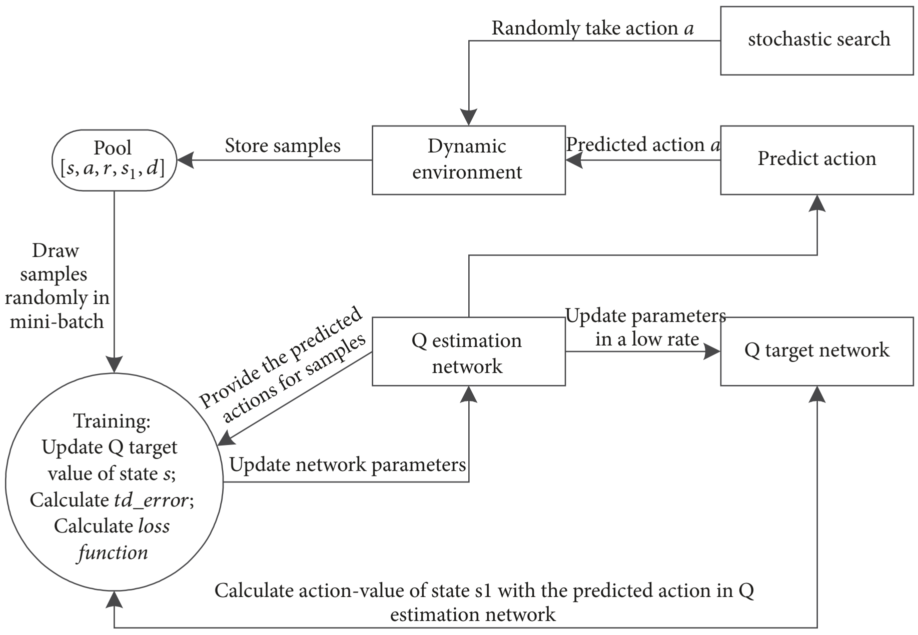

Randomly take action a
stochastic search
Pool $[s, a, r, s_1, d]$
Store samples
Dynamic environment
Predicted action a
Predict action
Draw samples randomly in mini-batch
Provide the predicted actions for samples
Q estimation network
Update parameters in a low rate
Q target network
Training: Update Q target value of state s; Calculate *td_error*; Calculate *loss function*
Update network parameters
Calculate action-value of state s1 with the predicted action in Q estimation network

Figure 1: The framework illustration of DDQN.

where n is the current iteration time-step; $i_{\min}$ is the initial value of L; $i_{\max}$ is the maximum value of L; m is the search speed in space; N_1 and N_2 are thresholds of iteration times which need to be adjusted according to training parameters.

The termination of each episode is to achieve a fixed number of moving steps instead of directly terminating the current training episode when encountering obstacles or reaching the target point. The original training mode results in a large number of samples which are moving states in the free zone in the pool, and the lack of trial-and-error punishment samples and target reward samples ultimately leads to algorithm disconvergence.

4. The Neural Network in DDQN

Neural network is a primary method for reinforcement learning to enhance the ability of generalization. The optimization design of the neural network architecture is able to reduce overfitting and improve the prediction accuracy with high training speed and low computational cost.

Considering 800 inputs, if we directly adopted fully connected layers, the number of parameters needed to be trained would increase exponentially with the increase of the layers, resulting in a heavy computation burden and overfitting. Convolutional neural networks (CNNs) have made breakthrough progress in the field of image recognition in recent years. CNNs are featured with a unique network processing mode which is characteristic of local connection, pooling, sharing the weights, etc. The unique mode effectively reduces the computational cost, computational complexity, and the number of the trained parameters. With the method of CNNs, the image model in the translation, scaling, distortion, and other transformations is able to be invariant to a certain degree, thereby improving the robustness of system to failures. Based on these superior features, CNNs surpass fully connected neural network in information processing task where the data are relevant to each other [18].

The framework of CNNs is shown in Figure 2. Generally, the CNNs consist of input layer, convolutional layer, pooling layer, fully connected layer, and output layer. convolutional layer adopts local connection; namely, each node is connected with some nodes of the upper layer. Four adjacent pixels of input layer form a 2-by-2 local sensory field which is connected with a neuron in convolutional layer. Each local sensory field represents a local feature, and the size of the field can be adjusted artificially.

The next layer of convolutional layer is pooling layer, which can also reduce the matrix size. The spatial invariant feature is obtained by reducing the resolution of the feature surface [19]. The number of feature maps in the pooling layer is equal to the number of convolutional layers, and each neuron performs a pooling operation on the local sensory field. Pooling is suitable for separating the sparse features.

A fully connected layer is added after one or more convolutional layers and pooling layers. The fully connected layer integrates the partial information involved in category distinguishing in convolutional layers or pooling layers [20]. Besides, rectified linear unit (ReLU) is a recent used activation functions in CNNs, which can improve the performance of a CNN. The output value of the fully connected layer is directly transferred to the output layer. For deep reinforcement learning, the output layer is the action space.

The lidar data between adjacent points reflects the distribution of obstacles and the width of the free zone. The convolutional network can effectively extract the information

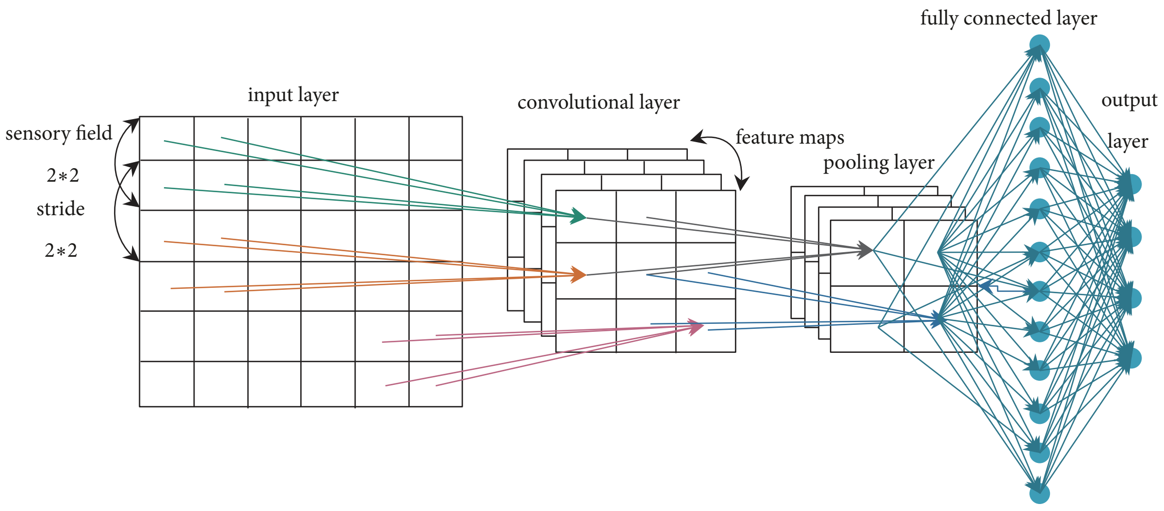


Figure 2: The architecture of CNNs.

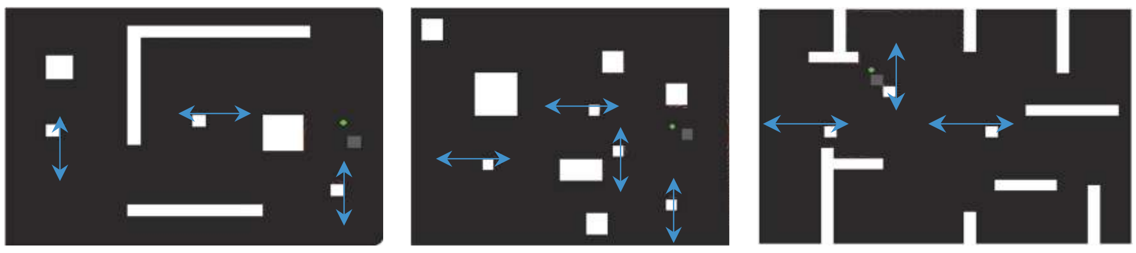

(a) Low level environment
(b) Intermediate environment
(c) High-level environment

Figure 3: Dynamic environments.

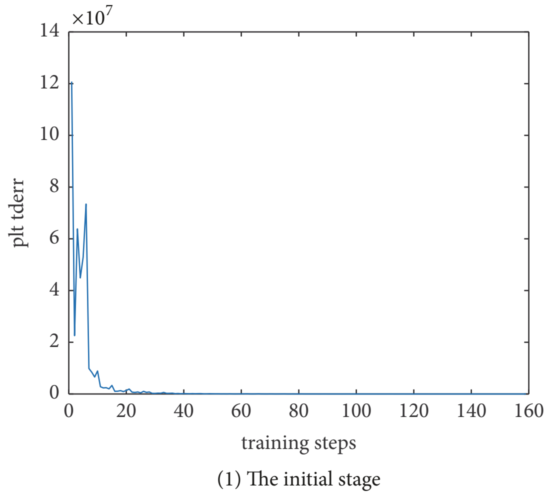
of these characteristics and reduce the network parameters greatly. Therefore, the CNNs are used to train the path planning of local dynamic environment.

In this paper, the architecture of Q-network consists of three convolutional layers and a fully connected layer. The input layer is a three-dimensional matrix with the size of 20×20×2 formed by a vector with 800 elements, where the data in the third dimension represents the angle and distance of a lidar point. The size of the input layer is 20×20×2. According to the size of the input layer, we design the first convolutional layer, of which the size of the receptive field is 2-by-2, the stride is 2-by-2, and the number of feature maps is set to be 16. Thus, the size of the output layer is 10×10×16. The kernel size of the second convolutional layer is 2-by-2; the stride is 2-by-2; and the number of the characteristic plane is 32. The size of the output of this convolutional layer is 5×5×32. The kernel size of the third convolutional layer is 5-by-5; the stride is 1-by-1; and the number of the characteristic plane is 128. The size of the output of the third convolutional layer is 1×1×128. Then a three-dimensional structure is transformed to a one-dimensional vector with 128 elements, which are connected to the fully connected layer, of which the size is 128-by-256. The size of the output layer is the number of the actions, namely, 8. We use ReLU activation function and Adam optimizer.

5. Local Path Planning Simulation with Pygame Module

We use the open source machine learning framework TensorFlow to build the DDQN training framework. Pygame module is to build dynamic environment. As shown in Figure 3, the white area denotes obstacles, with the size of 20-by-20 cm, are moving in a constant speed, and the corresponding blue arrow represents its moving range and directions. The green point denotes a local target position. The gray square object represents the mobile robot (agent). During the training, the local path planning is trained in the lower-level and intermediate environment. The starting point and target point are randomly generated to ensure diversity and complexity of local environment, and the test environment is a high-level dynamic map. Two CNNs of the same architecture are established, which are Q estimation network and Q target network. The network parameters are randomly initialized with normal distribution, of which the mean value is 0.

The training policy is a variable random greedy rule. At the beginning of the training, the pool of experience is updated by the method of random exploitation because of the lack of environmental information. The sample consists of five components, namely, $[s, a, r, s_1, d]$, where s denotes



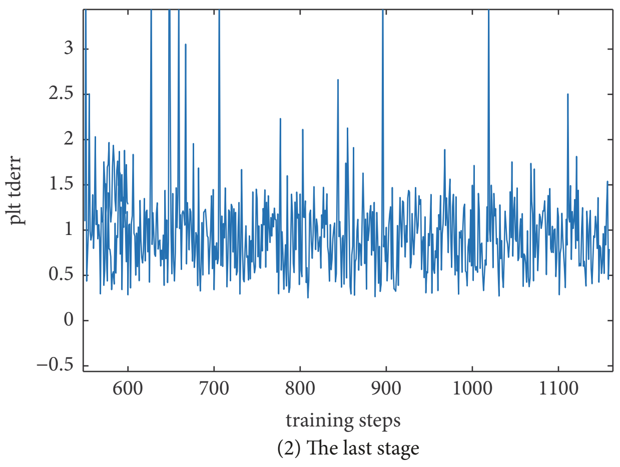


FIGURE 4: **Training curves of the loss function of Q target network.** Each point is the average loss function value achieved per ten epochs. The y-axis denotes the value of loss function and x-axis denotes iteration epoch.

the current state; a denotes the adopted action; s_1 is the state after taking action; R is the reward value obtained for the current state transition; d is a flag representing that whether the current iteration epoch ends. We set the size of the pool to 40000 samples. If the samples stored in the pool reached a certain number, the network was to be trained with the randomly selected samples in the pool. In the first 5000 time-steps of the random exploration, the network parameters are not updated, but the samples of the pool are increased. After the sample size reaches to 5000, the network is trained in every four time-step movements. We randomly draw 32 samples in sequence by means of minibatch to update the Q estimation network and update the Q target network with a low learning rate to make the parameters of Q target network approach the parameters of Q estimation network, thereby ensuring a stable learning of Q target network. If the sample size of the pool was up to the upper limit, the earliest sample would be excluded from the pool in the policy of first-in first-out after a new sample is added, thereby ensuring a continuous updating of the pool.

According to the actual experimental effect, if the position of agent and the target point were completely randomly set at first, there would be a large probability that the distance between the agent and target point was too large, resulting in that the agent was unable to reach the target point in fixed steps with random exploration. Consequently, to ensure the target point being detected by the agent, we set the initial distance between the agent and target point randomly in a range of L and gradually increase L with exploration and training. This process is also a process of constantly changing the environmental states which leads to a more extensive distribution of samples.

During the training process, the values of loss function of Q estimation network and Q target network decrease continuously. Figure 4 shows the training curves of the loss function of Q target network. Each point is the average loss function value achieved per ten epochs. Figure 4 (1) indicates that, in initial stage of the training, the magnitude of loss function value is 10^7. while, as we got from our data, after training for 1000 epochs, actually, the value dropped to about 10,000 (seen from the figure, the curve tends to be close to x-axis); after 7000 epochs, as we can see from Figure 4 (2) that it dropped to less than 1. With the whole process completed, the loss function value is less than 0.25, and the Q-network converges successfully. The cumulative reward of each iteration is gradually increased; that is, in the process of exploration, the agent gradually learns to take an action which can increase the reward. Figure 5 shows the average cumulative reward curves. Each point is the average cumulative reward achieved per hundred epochs. The figure indicates that the average reward gradually increases with the increase of training epochs. After training for 30000 epochs, the reward value tends to be stable and average cumulative reward is greater than 80. We set the fixed step times to be 100, which means the agent can avoid obstacles in a dynamic

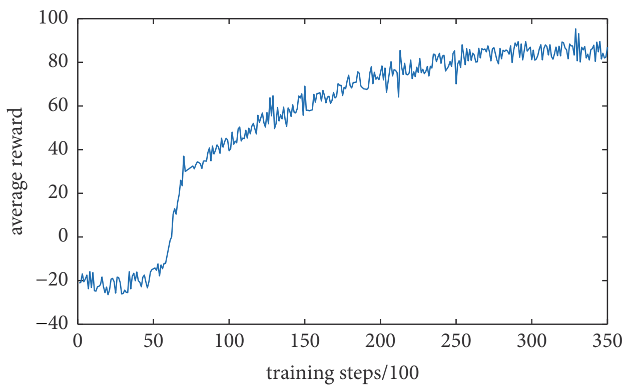
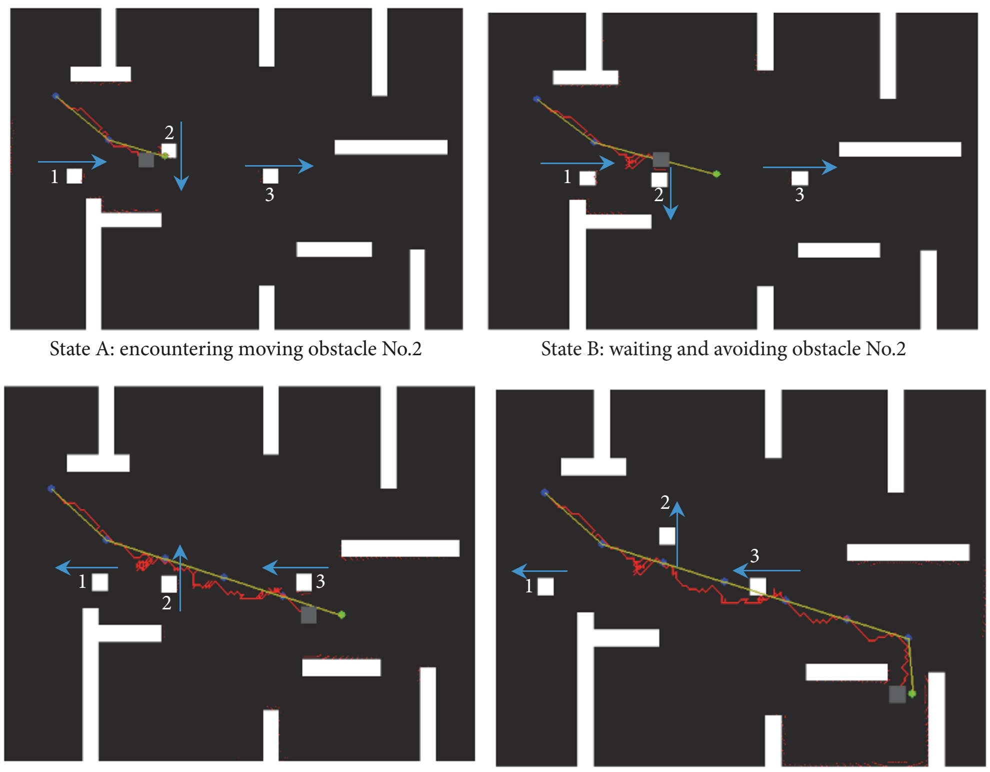


FIGURE 5: **The average cumulative reward curves.** Each point is the average cumulative reward achieved per hundred episodes. The y-axis denotes the average cumulative reward and x-axis denotes iteration epoch.



State A: encountering moving obstacle No.2

State B: waiting and avoiding obstacle No.2

State C: avoiding obstacle No. 3 towards down right

State D: arriving at target point

Figure 6: Local path planning in a test map.

environment and quickly reach the target point to obtain a continuous reward until the current training epoch ends.

After training for 40000 epochs, Q estimation network and Q target network converge. We store the network parameters and test in an experimental environment map. The test is designed as the following. Assuming a global path, we set several local target points in the lidar searching area regardless of the positions of the dynamic and static obstacles to test the agent's local path planning ability. Figure 6 is a new environmental map which has never tested in training. This map is used to evaluate the generalization ability and the path planning ability of DDQN in a new unknown dynamic environment. There are three free moving obstacles with constant speed and the moving directions are denoted by the blue arrows. In the figure, state A demonstrates that the local path is blocked by dynamic obstacle No. 2; then, the agent waits for the obstacle to move downwards. When the obstacle is out of the path, the agent moves towards upper right. As shown in state B, the agent successfully reaches the third local target. In state C, obstacle No. 3 moves towards the agent. The agent perceives obstacle No. 3 before collision and gets to the sixth local target point by moving toward the bottom right. In state D, the agent reaches the end point and completes the path planning without any accident collision with the obstacles. The whole process demonstrates that the agent after being trained by DDQN is able to perceive the moving obstacles in advance with the knowledge of lidar data in unknown dynamic environment. An intelligent planning method is presented by Q target network which makes each step move towards a higher cumulative reward.

6. Test in Actual Environment with ROS Framework

In order to verify the effect of the algorithm in actual environment, we use an omnidirectional mobile robot based on Mecanum Wheels and achieve autonomous navigation with ROS framework [21–23]. The *move_base package* in ROS provides local path planning algorithms for user, that is, dynamic windows method and track-reckoning method. In this paper, DDQN local path planning algorithm is transplanted into ROS in the form of plug-ins, which is encapsulated in the pure virtual function *base_local_planner* of *nov_core* as subclass, namely, *DDQN_local_planner*. The outputs of the deep reinforcement learning algorithm are the translational movements of the agent in eight directions. Then the movements are mapped to actions of moving forward and turning. The agent firstly changes its direction in situ to the corresponding direction before moving and then moves forward 10 cm with a preset speed.

The path planning of the unknown environment is to navigate without using the prior environment SLAM map. Figure 7 is a local map. On this map, only local information is detected by lidar in real-time, where the gray area is unknown. The agent avoids obstacles with local path planning. The starting position is set to be the current position

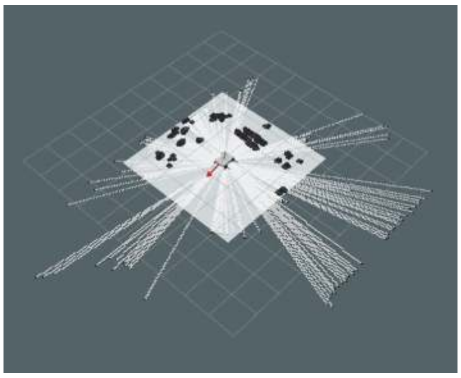


Figure 7: A local environment map.

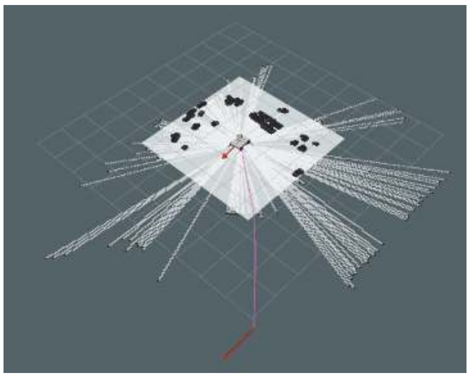


Figure 8: The schematic of global path planning in unknown environment.

of agent and the target point is set to be in unknown area. The unknown area between the starting point and the target point is considered as an available area. The schematic of path planning is shown in Figure 8.

The actual layout of the unknown environment is shown in Figure 9. In the figure, the red arrows denote preplanned global paths, which is blocked by obstacle No. 2; thus the agent needs to readjust the path to reach the target position without hitting the obstacles.

The local path planning is completed by the trained DDQN algorithm. The inputs are angles, distance detected by lidar, and local planning points. The output is moving direction. And, meanwhile, the trajectory is smoothed. During the movement, the agent builds map and navigates in real-time. The local path planning effect of the algorithm and generalization ability of the convolutional network are tested in unknown environment. Figure 10 shows the final path. The agent followed the global path but modified its path and successfully avoided the obstacles when it encountered obstacles. A new environmental map was gradually built in the process of moving.

The movement effect of the agent in actual environment is shown in Figure 11. When the agent was in front of the obstacle with a distance of about 30 cm, it changed direction and avoided the obstacle on the left. The agent approached

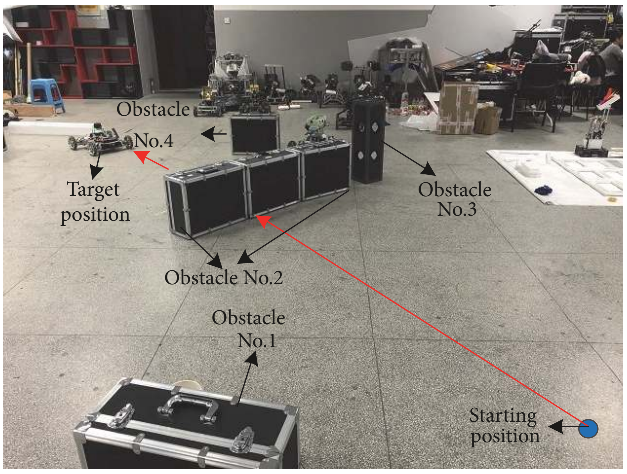


Figure 9: The actual layout of the unknown environment.

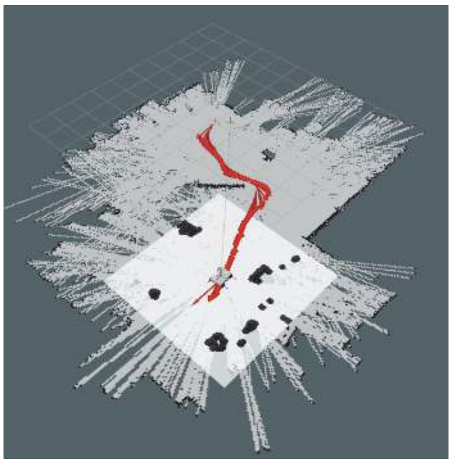


Figure 10: Path planning in unknown environment.

the global path again, finally arriving at the target point. The test verifies the feasibility and efficiency of the algorithm in unknown environment.

7. Conclusions

Deep reinforcement learning solves the problem of curse of dimensionality well and is able to process multidimensional inputs. In this paper, we design a specific training method and reward function, which effectively addresses the problem of disconvergence of the algorithm due to the reward sparsity of state space. The experimental results show that DDQN algorithm is capable and flexible for local path planning in unknown dynamic environment with the knowledge of lidar data. Compared with visual navigation tasks, the local lidar information, which can be transplanted into more complex environments without retraining the network parameters, has wider applications and better generalization performance.

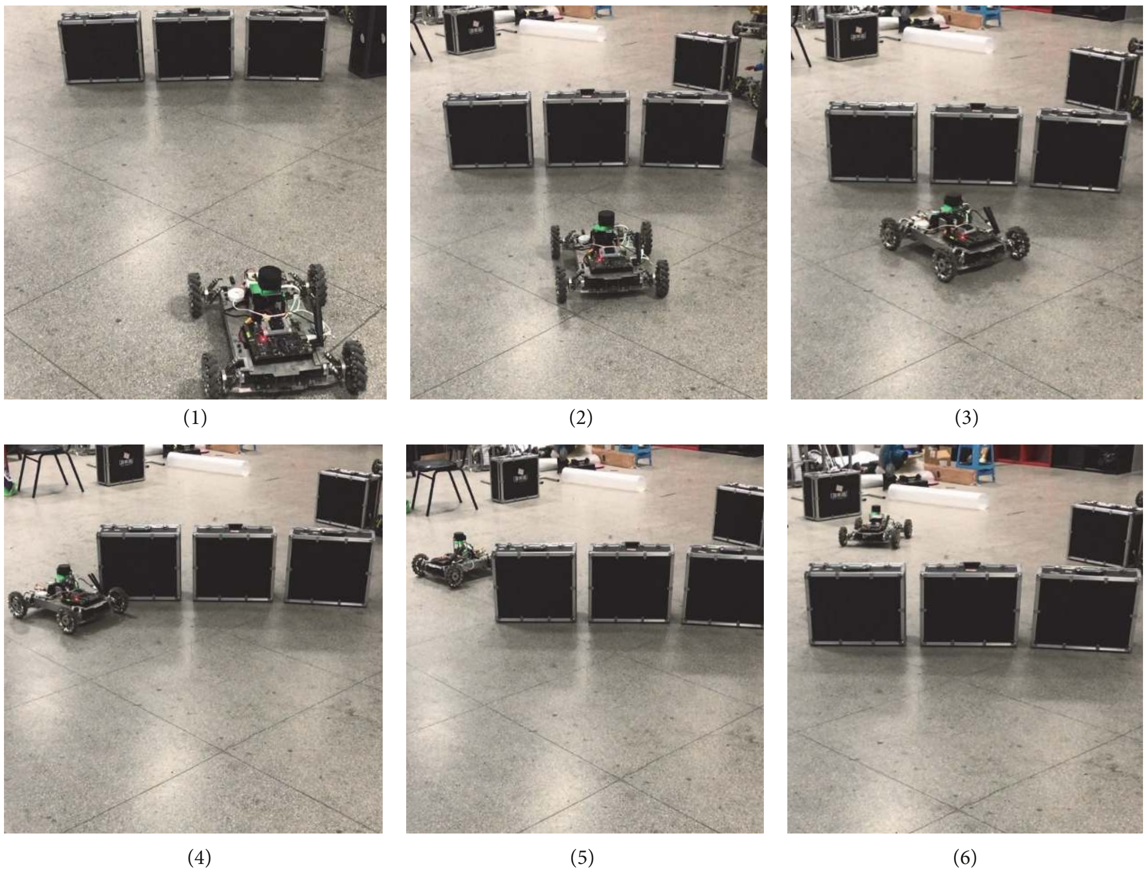


Figure 11: The movements of the agent in actual environment.

Data Availability

The datasets and codes generated and analyzed during the current study are available in the Github repository [https://github.com/raykking].

Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this paper.

Acknowledgments

This work is supported by National Natural Science Foundation (NNSF) of China under Grant 11472008.

References

[1] R. Sutton and A. Barto, *Reinforcement Learning: An Introduction*, MIT Press, 1998.

[2] S. Legg and M. Hutter, "Universal intelligence: A definition of machine intelligence," *Minds and Machines*, vol. 17, no. 4, pp. 391–444, 2007.

[3] V. Mnih, K. Kavukcuoglu, D. Silver et al., "Playing atari with deep reinforcement learning," https://arxiv.org/abs/1312.5602, 2013.

[4] V. Mnih, K. Kavukcuoglu, D. Silver et al., "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[5] Y. Bengio, "Learning deep architectures for AI," *Foundations and Trends in Machine Learning*, vol. 2, no. 1, pp. 1–27, 2009.

[6] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Proceedings of the 26th Annual Conference on Neural Information Processing Systems (NIPS '12)*, pp. 1097–1105, Lake Tahoe, Nev, USA, December 2012.

[7] G. E. Hinton and R. R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," *The American Association for the Advancement of Science: Science*, vol. 313, no. 5786, pp. 504–507, 2006.

[8] D. Silver, J. Schrittwieser, K. Simonyan et al., "Mastering the game of Go without human knowledge," *Nature*, vol. 550, no. 7676, pp. 354–359, 2017.

[9] Z. Tang, K. Shao, D. Zhao, and Y. Zhu, "Recent progress of deep reinforcement learning: from AlphaGo to AlphaGo Zero," *Control Theory & Applications*, vol. 34, no. 12, pp. 1529–1546, 2017.

[10] G. Lample and D. S. Chaplot, "Playing FPS games with deep reinforcement learning," in *Proceedings of the 31st AAAI Conference on Artificial Intelligence, AAAI 2017*, pp. 2140–2146, USA, February 2017.

[11] P. Mirowski, *Learning to navigate in complex environments*, 2016.

[12] Y. Zhu, R. Mottaghi, E. Kolve et al., "Target-driven visual navigation in indoor scenes using deep reinforcement learning," in *Proceedings of the 2017 IEEE International Conference on Robotics and Automation, ICRA 2017*, pp. 3357–3364, Singapore, June 2017.

[13] J. N. Tsitsiklis and B. Van Roy, "An analysis of temporal-difference learning with function approximation Technical," Tech. Rep. LIDS-P-2322, Laboratory for Information and Decision Systems, Massachusetts Institute of Technology, 1996.

[14] J. L. McClelland, B. L. McNaughton, and R. C. O'Reilly, "Why there are complementary learning systems in the hippocampus and neocortex: insights from the successes and failures of connectionist models of learning and memory," *Psychological Review*, vol. 102, no. 3, pp. 419–457, 1995.

[15] J. O'Neill, B. Pleydell-Bouverie, D. Dupret, and J. Csicsvari, "Play it again: reactivation of waking experience and memory," *Trends in Neurosciences*, vol. 33, no. 5, pp. 220–229, 2010.

[16] L. J. Lin, "Reinforcement learning for robots using neural networks," Tech. Rep. CMU-CS-93-103, Carnegie-Mellon University Pittsburgh PA School of Computer Science, 1993.

[17] H. Van Hasselt, A. Guez, and D. Silver, "Deep Reinforcement Learning with Double Q-Learning," *Association for the Advancement of Artificial Intelligence (AAAI)*, vol. 2, 2016.

[18] Q. Zhang, "Convolutional Neural Networks," in *Proceedings of the 3rd International Conference on Electromechanical Control Technology and Transportation*, pp. 434–439, Chongqing, China, January 2018.

[19] J. Gu, Z. Wang, J. Kuen et al., "Recent advances in convolutional neural networks," *Pattern Recognition*, vol. 77, pp. 354–377, 2018.

[20] T. N. Sainath, A.-R. Mohamed, B. Kingsbury, and B. Ramabhadran, "Deep convolutional neural networks for LVCSR," in *Proceedings of the 38th IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP '13)*, pp. 8614–8618, IEEE, Vancouver, Canada, May 2013.

[21] Willow Garage, *Robot Operating System*, Willow Garage, 2015, http://www.ros.org/.

[22] M. Quigley, K. Conley, B. Gerkey et al., "ROS: an open-source Robot Operating System," vol. 3, no. 3.2, ICRA workshop on open source software, May 2009.

[23] M. Cashmore, M. Fox, D. Long et al., "Rosplan: Planning in the robot operating system," in *Proceedings of the 25th International Conference on Automated Planning and Scheduling, (ICAPS '15)*, pp. 333–341, June 2015.