Hindawi
Journal of Robotics
Volume 2020, Article ID 7167243, 10 pages
https://doi.org/10.1155/2020/7167243

Research Article

Research on Dynamic Path Planning of Wheeled Robot Based on Deep Reinforcement Learning on the Slope Ground

Peng Wang, Xiaoqiang Li, Chunxiao Song, and Shipeng Zhai

School of Mechanical and Power Engineering, Harbin University of Science and Technology, Harbin 150080, China

Correspondence should be addressed to Peng Wang; wangpcw@163.com

Received 27 May 2019; Revised 27 August 2019; Accepted 6 November 2019; Published 1 February 2020

Academic Editor: Shahram Payandeh

The existing dynamic path planning algorithm cannot properly solve the problem of the path planning of wheeled robot on the slope ground with dynamic moving obstacles. To solve the problem of slow convergence rate in the training phase of DDQN, the dynamic path planning algorithm based on Tree-Double Deep Q Network (TDDQN) is proposed. The algorithm discards detected incomplete and over-detected paths by optimizing the tree structure, and combines the DDQN method with the tree structure method. Firstly, DDQN algorithm is used to select the best action in the current state after performing fewer actions, so as to obtain the candidate path that meets the conditions. And then, according to the obtained state, the above process is repeatedly executed to form multiple paths of the tree structure. Finally, the non-maximum suppression method is used to select the best path from the plurality of eligible candidate paths. ROS simulation and experiment verify that the wheeled robot can reach the target effectively on the slope ground with moving obstacles. The results show that compared with DDQN algorithm, TDDQN has the advantages of fast convergence and low loss function.

1. Introduction

Path planning technology of mobile robot is one of the core contents of intelligent mobile robot research [1]. Path planning of mobile robot refers to that activity where the robot can perceive the environment and independently plan a route to reach the target based on the information obtained by the sensor. With the continuous development of science and technology, mobile robot is facing more and more complex and changeable environment. The traditional path planning algorithm lacks flexibility and is easy to fall into local optimal solutions, which cannot meet the needs of mobile robots [2], such as artificial potential field method [3], simulate anneal arithmetic [4] and ant colony optimization [5]. In response to this situation, deep reinforcement learning has been proposed [6]. Deep reinforcement learning is the new algorithm based on the development of deep learning in recent years. The perceptual ability of deep learning and the decision-making ability of reinforcement learning are integrated by deep reinforcement learning. It is an artificial intelligence algorithm that is closer to human thinking mode, which can better realize the mobile requirements of robots in complex environments. Mnih et al. combined the convolutional neural networks with the Q learning algorithm in traditional reinforcement learning, and proposed the Deep Q-Network (DQN), which was well verified in the Atari 2600 game [7]. Hado et al. proposed the Deep Double Q-Network (DDQN) which successfully solved the problem of over-optimistic estimation function [8]. When training DQN, Tom et al. [9] used the experience playback mechanism based on priority to replace the sampling method of equal probability, which improved the utilization rate of valuable samples, but required a large storage space. Xin et al. first proposed the path planning method for mobile robots based on DQN [10]. Tai et al. proposed DQN algorithm for path planning simulation of mobile robot [11], but the DQN algorithm has the disadvantages of overestimation of action value and slow convergence speed. JIE Zequn applied tree structure reinforcement learning to path planning to provide multiple search paths [12], but the DQN algorithm used in it adopted the same maximum Q value to select and evaluate actions, which is likely to select sub-optimal actions, so that more actions are needed to detect the target, affecting the final detection results.

Therefore, this paper proposes a dynamic path planning algorithm based on deep reinforcement learning, namely Tree Double Deep Q-Network (TDDQN). By performing the action of changing the search path, the agent can quickly find an optimal path in the local path region, the path planning results of wheeled robots will be further improved on the slope ground. The actions of the agent will be divided into two groups, in which each group adopts DDQN to select the best action in the current state. According to the state obtained after the selected action is executed, the previous step is repeated to form multiple paths resembling the tree structure. The feature of DDQN that can select the best action enables the agent to get the candidate path that meets the conditions after performing fewer actions. Combined with multiple paths of tree structure, multiple candidate paths are obtained, which is more conducive to selecting the best candidate path.

2. Deep Reinforcement Learning Based on TDDQN Algorithm

2.1. Implementation of DDQN Algorithm. DQN update mode causes overestimation of action values [13]. The algorithm overoptimistically estimates the value of a certain state, resulting in the Q value of the sub-optimal action is greater than the Q value of the optimal action, thus changing the optimal action and affecting the accuracy of the algorithm. In order to solve the problem of sub-optimal Q value when Q-network is used to select and evaluate actions, DDQN is adopted to improve the probability of selecting the optimal action in this paper [8]. That is, DDQN selects actions through the Q-network and then evaluates actions through another Q-network. Assuming that the target value of Q-network is represented by Y:

$$Y_i^{DDQN} = E[R + \gamma Q(S', \operatorname{argmax}(Q(S', A'; \theta_i)); \theta_i^-)|S, A], \quad (1)$$

where, i is the number of current iterations, S is the current state, A is the action performed under the current state, θ is the parameter of the model, R is the current reward, γ is the discount coefficient, θ_i is the parameter of Q estimation network in iteration i, and θ_i^- is the parameter of the target network in iteration i. The target network parameter θ_i^- is only updated with Q estimation network parameters θ_i in every N step and maintains a fixed value between updates, so as to ensure the delayed update of the target network and the accurate estimation of Q value.

The pool of data sample is expanded by executing the ε-greedy strategy to search the environment. During the learning period, the random data in the sample pool is used to train the network, so as to eliminate the correlation in the observation sequence and improve the stability of the algorithm. The loss function of DDQN algorithm is:

$$L_i^{DDQN}(\theta_i) = E\left[\left(R + \gamma Q(S', \operatorname{argmax}(Q(S', A'; \theta_i)); \theta_i^-) - Q(S, A; \theta_i)\right)^2\right]. \quad (2)$$

2.2. Description of TDDQN Dynamic Path Planning Method. In this section, multiple paths will be generated by the tree structure, where each path adopts DDQN algorithm [14]. Figure 1 is the search process of tree structure. In the figure, the solid line represents the selected global path planning action, and the dotted line represents the local path planning action. The previous two levels as an example, the root node in level 1 is referred to as the current state, or candidate path. For the current candidate path, the two actions with the highest predicted value in the global path planning action group and the local path planning action group are selected respectively, and the two actions are respectively performed on the local navigation area to obtain two current states: left subtree and right subtree, namely candidate regions represented by two nodes in level 2. By analogy, the third level, the fourth level of the tree structure and so on can be obtained respectively.

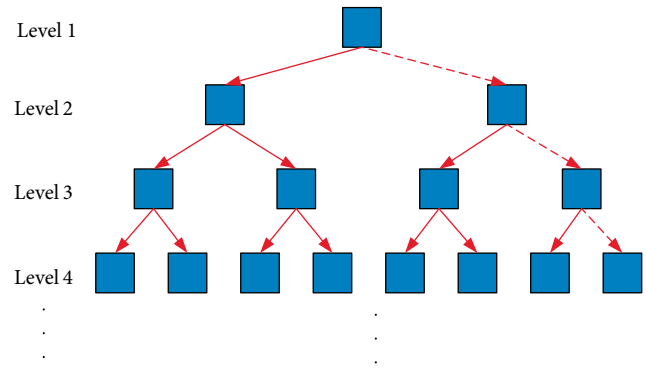
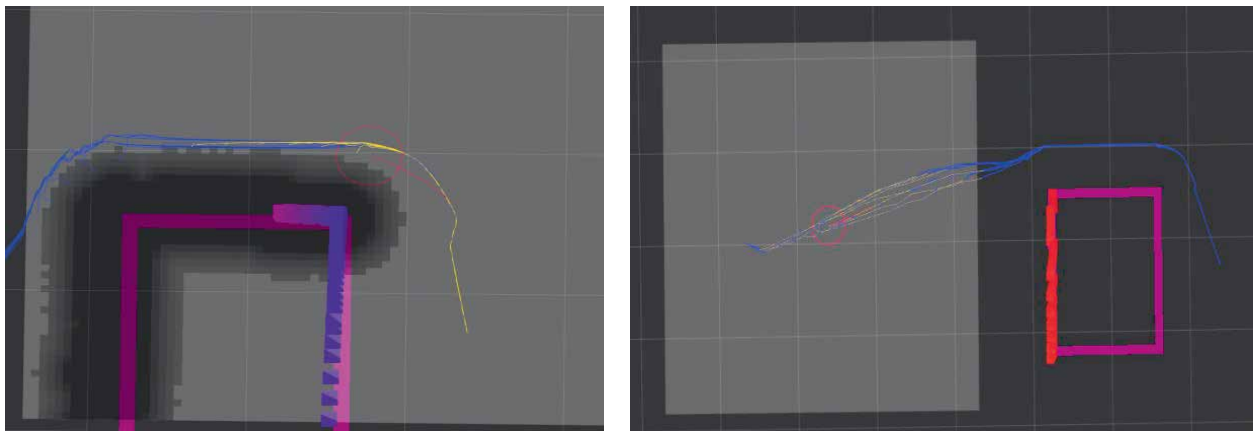


Figure 1: Schematic diagram of the tree structure. The solid line represents the selected global path planning action, and the dotted line represents the local path planning action.

In the process of executing an action on the local navigation region, the search will be stopped if the reward value obtained is greater than the threshold value R_n. The possible results are shown in Figure 2, the red line represents the actual path result, the blue line represents the global path planning, and the yellow line represents the local path planning. Figure 2(a) shows that the result meets the requirements but is not the best result, if the local navigation region moves slightly to the lower left corner, the better path result can be generated. In this case, this path can be terminated, while other paths in the tree can continue to execute until the reward value of each node is greater than R_n. Therefore, the choice of multiple paths can increase the probability of reaching the target. As shown in Figure 2(b), it is difficult to reach the target point in fewer paths. In general, the path with too many execution actions is unlikely to be the ideal path for final selection, so an upper limit value M_n can be set for the number of levels in the tree, so that the search can be stopped as long as the number of levels M exceeds M_n, regardless of whether the node greater than the reward value R_n.

In Figure 2, the red circle represents the robot, the red line represents the actual path result, the blue line represents the global path plan, the yellow line represents the local path plan, and the purple rectangle represents the obstacle. According to the above analysis, the tree search structure is designed in this paper, as shown in Figure 3. Assuming that the reward values of two branches on the right side of level 3 are greater than R_n, the search is stopped at level 4, so only the left branch of level 4 can continue. By analogy, several local paths will eventually be obtained, the best candidate local path will be found by the non-maximum suppression method.



(a) (b)

Figure 2: Possible detection results. The red circle represents the robot, the red line represents the actual path result, the blue line represents the global path plan, the yellow line represents the local path plan, and the purple rectangle represents the obstacle. (a) Detected incomplete path. (b) Over-detected paths.

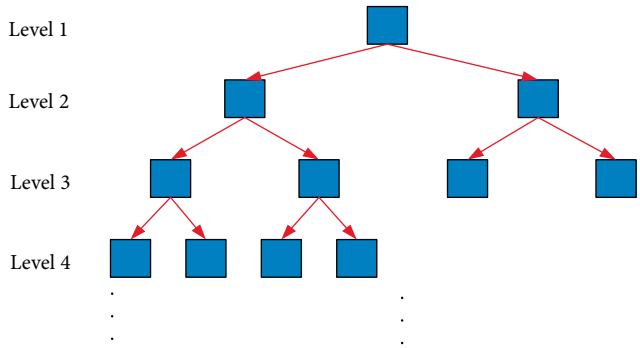


Figure 3: Schematic diagram of tree structure generation.

2.3. Implementation of TDDQN Dynamic Path Planning Method. In this paper, local path planning is realized through TDDQN. Lidar is used for 360-degree environmental information detection, and the local navigation region is the sensor range. The target point of local path planning is the point closest to the global path at the edge of local navigation region. Lidar dot matrix information and local target point coordinates are taken as the network input, the output is the direction of movement.

Considering the calculation burden and actual navigation effect, the angle resolution is set to 1 degree and the distance limit is set to 2 meters. Therefore, each observation consists of 360 points, showing the distance of obstacles within 2 meters around the robot. The local target point is the intersection of local navigation region and global path. If there are multiple intersection points, the optimal point is selected by heuristic evaluation. For the lidar lattice information, the angle and distance can be expressed by [*angle, distance*]. The rule is made that the angle of the lidar point increased clockwise relative to the front of the mobile robot, so that 360 degrees produce 720 input data. Meanwhile, the relative coordinates of the local target point and the current central coordinates of the mobile robot body $[\Delta x, \Delta y]$ are taken as input. In order to facilitate the design of convolutional network and achieve the output of relative target point network weights in training, the data of 40 relative target points are copied as input, that is, the total input is 800 data sets.

The output is a fixed-length omnidirectional motion in eight directions, in which the motion length is 10 cm. The directions are forward, backward, left, right, left front, left back, right front and right back, expressed by A_1–A_8, the order is as follows:

$$A = [A_1, A_2, A_3, A_4, A_5, A_6, A_7, A_8]. \tag{3}$$

The design of the reward function mainly considers obstacle avoidance and approach to the target point. The shortest motion path that satisfies these two conditions is the most efficient, and the reward function is designed as follows [15]:

$$R\begin{cases} 1 & (p(x, y) = q(x', y')), \\ -0.01 & \begin{pmatrix} p(x, y) \neq q(x', y') \\ p(x, y) \neq q(x', y') \end{pmatrix}, \\ -1 & (p(x, y) = o(x', y')), \end{cases} \tag{4}$$

where, p is the current position, q is the local target point, and o represents an obstacle with expansion processing. The above equation indicates that if the agent encounters obstacles, the reward will be −1. If the agent reaches the target point, it will get a +1 reward. In other states, the cost of each movement is −0.01. In order to improve the cumulative rewards in training, the agent adopts the trial-and-error learning method, and tries to reach the target point in a short distance and avoid obstacles.

The continuous measurement value received by each lidar detection point ranges from 0 cm to 2 m, which means that the state space tends to be infinite, and it is impossible to express all the states with Q value. Using the generalization ability of DDQN neural network, the trained network can approach all the states. Therefore, the agent can plan appropriate path and reach the target location according to the network weight when the environment changes.

In order to ensure the normal convergence of deep reinforcement learning, the training pool should be large enough to store the state actions of each time-step, maintain the

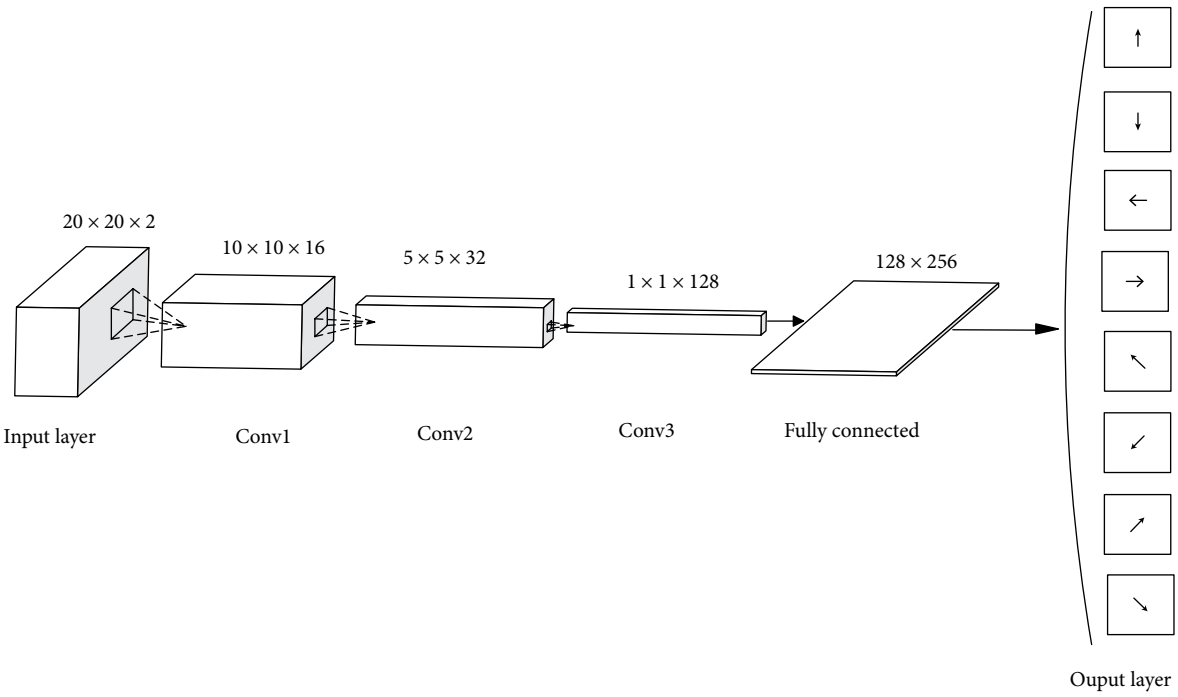


Figure 4: CNN framework.

independent and consistent distribution of the neural network training samples, while the environmental punishment and reward should reach a certain proportion. It is difficult to achieve stable training effect if the sample space is too sparse, that is, the main state is the random motion in free space. For the instability of DDQN training and the sparsity of the reward in the state space, the starting point is randomly set in the circle with the radius of L centered on the target point. A small initial value of L increases the probability from the starting point to the target point in the random exploration and guarantees the positive excitation of sample space. With the updating of neural network and the increase of greedy probability, L value increases gradually. The local space explored by the agent is as follows:

$$L = \begin{cases} L_0 & (n \le T_1), \\ L_0 + \sqrt{\frac{(n-N_1)}{V}} & (T_1 \le n \le T_2), \\ L_m & (n \ge T_2), \end{cases} \tag{5}$$

where, n is the time-step of the current iteration, when learning the model, the Episode is not the smallest learning unit, but the time-step is the unit to learn one state transition at a time. L_0 is the initial value of L, L_m is the maximum value of L, V is the spatial search speed. T_1 and T_2 are the iteration time threshold, which need to be adjusted according to the training parameters.

In this section, the Q network structure consisting of three convolutional layers and one fully connected layer is proposed [16]. The CNN framework is shown in Figure 4. The input layer is a three-dimensional matrix with a size of $20 \times 20 \times 2$, which consisted of 800 vector elements. The three-dimensional data represents the angle and distance of the lidar point. According to the size of input layer, the first convolutional layer is designed, in which the size of the receptive field is 2×2, the convolution step is 2×2, and the number of feature maps is 16, so the size of the output layer is $10 \times 10 \times 16$. The convolution kernel size of the second convolution layer is 2×2, the convolution step is 2×2, the number of feature maps is 32, and the output size of the convolutional layer is $5 \times 5 \times 32$. The convolution kernel size of the third layer is 5×5, the convolution step is 1×1, the number of feature maps is 128; and the size of the output layer is $1 \times 1 \times 128$. The three-dimensional structure is then transformed into a one-dimensional vector with 128 elements connected to a complete connection layer of size 128×256. The size of the output layer is 8. The activation function is ReLU and the optimizer is Adam.

Firstly, the lattice information of lidar and local target coordinates are obtained. Then the current state is entered into the DDQN network. The network selects the best action from two groups of actions according to the current state, and the selected action is evaluated by Q network. The agent generates a new state (candidate path) after performing the action and acts as the next node of the tree structure. The above steps will be repeated until all branches of the tree structure have reached the detection criteria or the upper limit of the number of levels in the tree. At last, the optimal candidate path is selected by the nonmaximum suppression method. The implementation flow of selection method of candidate path based on the above model is shown in Table 1.

The TDDQN algorithm discards detected incomplete and over-detected paths by optimizing the tree structure, combines DDQN method with tree structure method, and selects the best actions in the current state after performing fewer actions through DDQN algorithm, so as to obtain candidate paths that meet the conditions. Then, according to the obtained state, the above process is repeatedly executed to form multiple paths of

Table 1: Selection method of candidate path based on TDDQN.

Input	Current state (candidate path, root node of tree)
Output	Next state (new candidate path, child node of tree)
Step 1	Initialize the threshold value R_n and the maximum level number M_n of the tree, and set the initial level number M of the tree as 1.
Step 2	According to the current state, the two actions with the highest predicted value obtained by DDQN method were selected from the global path planning action group and the local path planning action group respectively.
Step 3	The state obtained after performing the global path planning action as the left node, and the state obtained after performing the local path planning action as the right node.
Step 4	Add 1 to the number of levels in the tree.
Step 5	If the current number of levels in the tree is less than M_n and there are still branches that are not cut off, step 6 is performed, otherwise step 7 is performed.
Step 6	If the left node's reward value is greater than R_n, it will be cut off, otherwise, the left node will be taken as the current state of its path and step 2 will be executed. Accordingly, if the reward value of the right node is greater than R_n, it will be cut off, otherwise, the right node will be taken as the current state of its path and step 2 will be executed.
Step 7	The nonmaximum suppression method is used to select the optimal candidate path for all tree nodes, thus TDDQN path planning strategy is formed.

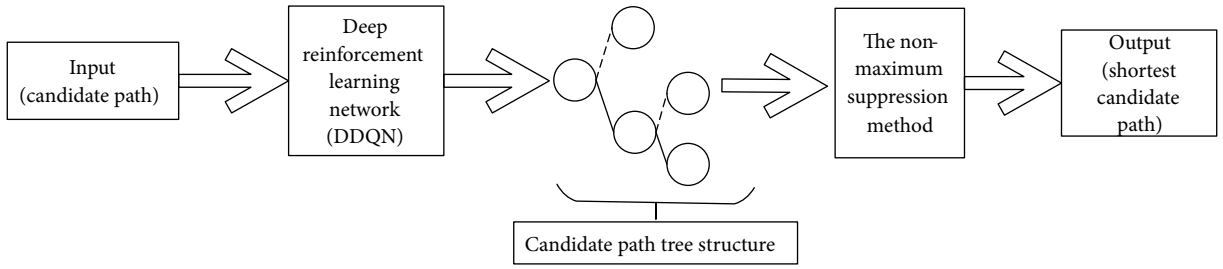


Figure 5: The overall structure of TDDQN algorithm.

the tree structure. Finally, the shortest path is selected from the plurality of eligible candidate paths by the nonmaximum suppression method. The overall structure of TDDQN algorithm is shown in Figure 5.

3. Simulation of Local Dynamic Path Planning

We use the open source machine learning framework TensorFlow to build the TDDQN training framework [17]. The Gazebo physical simulation engine is used to build the dynamic environment. The mobile environment of the wheeled robot studied in this paper is a slope of $15^\circ \le \theta \le 30^\circ$. As shown in Figure 6, the blue cube represents the dynamic obstacle with a size of $20 \times 20 \times 20$ cm, it moves at a constant speed. The range and direction of movement are indicated by red arrows. The red cube represents the local target locations. Blue cylindrical objects represent mobile robots (agent). During training, the current and target points are randomly generated to ensure the diversity and complexity of the local environment. Two CNN have the same Q estimation network and Q target network frame. The network parameters are randomly initialized to a normal distribution, where the mean value is 0.

Training policy is a random variable greed rule. At the beginning of training, the experience pool was updated by random exploration due to the lack of environmental information. The sample includes 5 parts, namely $[S, A, R, S', D]$,

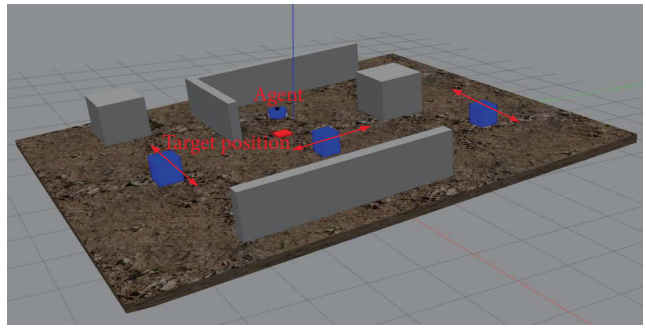


Figure 6: Gazebo physical simulation environment. The range and direction of movement are indicated by red arrows.

where S represents the current state, A represents action taken, R is the reward value obtained by the current state transition, S' refers to the state after taking action, and D is a sign that indicates whether the current iteration era is over. The pool size is set to 40,000 samples. If a certain number of samples are stored in the pool, the network will randomly select samples from the pool for training. In the first 5000 time-steps of random exploration, the network parameters were not updated, but the number of samples in the pool increased. When the sample size reaches to 5000, the network is trained to move in every four time-step. The Q estimation network was updated by random sampling of 32 samples in small batch, and the Q target network is updated at a lower learning rate

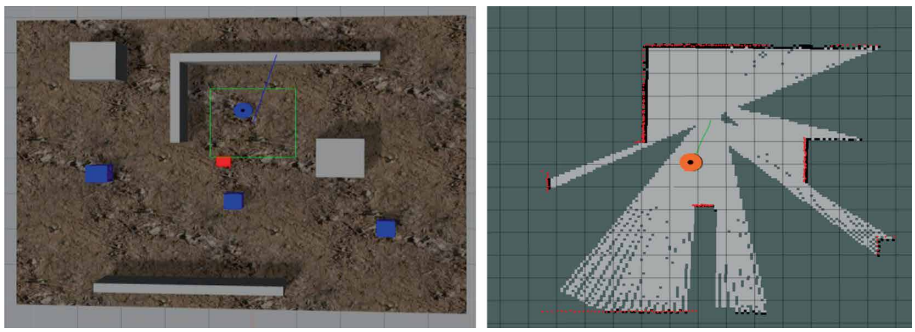


Figure 7: Local exploration process.

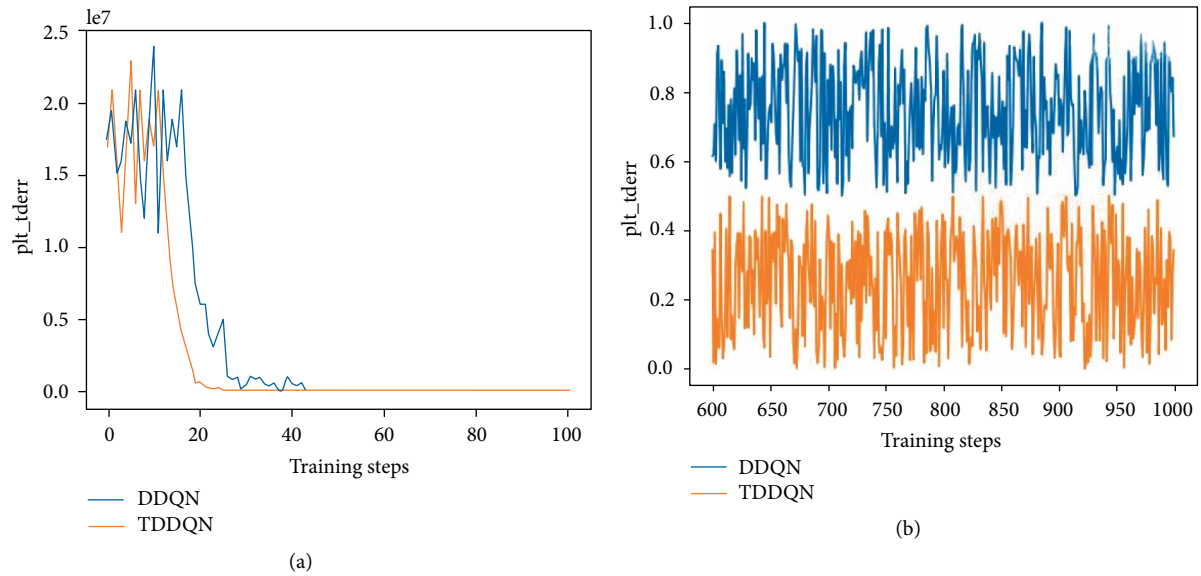


Figure 8: Training curve of Q target network loss function. Each point is the average loss function value achieved per ten epochs. The y-axis denotes the value of loss function and x-axis denotes iteration epoch. (a) Initial stage of training. (b) Convergence stage of training.

of 4 Hz, so that the parameters of the Q target network were close to the parameters of the Q estimation network, so as to ensure the stable learning of the Q target network. If the sample size of the pool was up to the upper limit, the earliest sample will 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 samples in the pool.

According to the experimental results of simulation, the distance between the agent and the target point is likely to be too large if the position of the agent and the target point is set completely randomly at the beginning, resulting in that the agent cannot reach the target point with a fixed number of steps through random exploration. Therefore, in order to ensure the target point is detected by the agent, the initial distance between the agent and the target point is randomly set within the range of L, and gradually increase L by exploration and training. This process is also a process in which the state of the environment is constantly changed, resulting in a wider distribution of samples, as shown in Figure 7.

In the process of training, the loss function values of Q estimation network and Q target network of TDDQN algorithm are continuously reduced. Compared with DDQN algorithm,

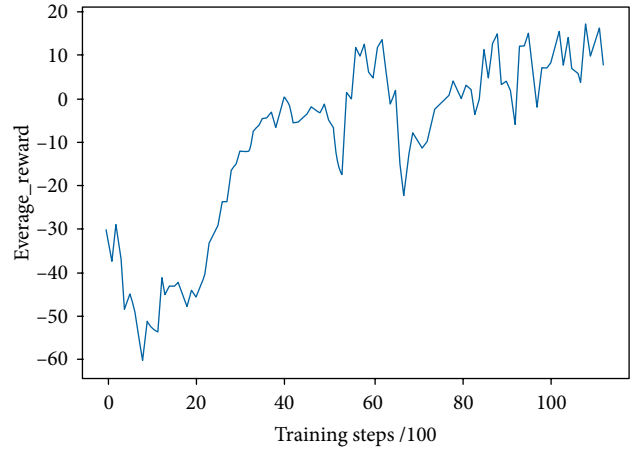


Figure 9: The average cumulative reward curve. Each point is the average cumulative reward achieved per hundred episodes. The y-axis denotes the average cumulative reward and x-axis denotes iteration epoch.

TDDQN algorithm has faster convergence speed and lower loss function value. Figure 8 shows the training curve of Q target

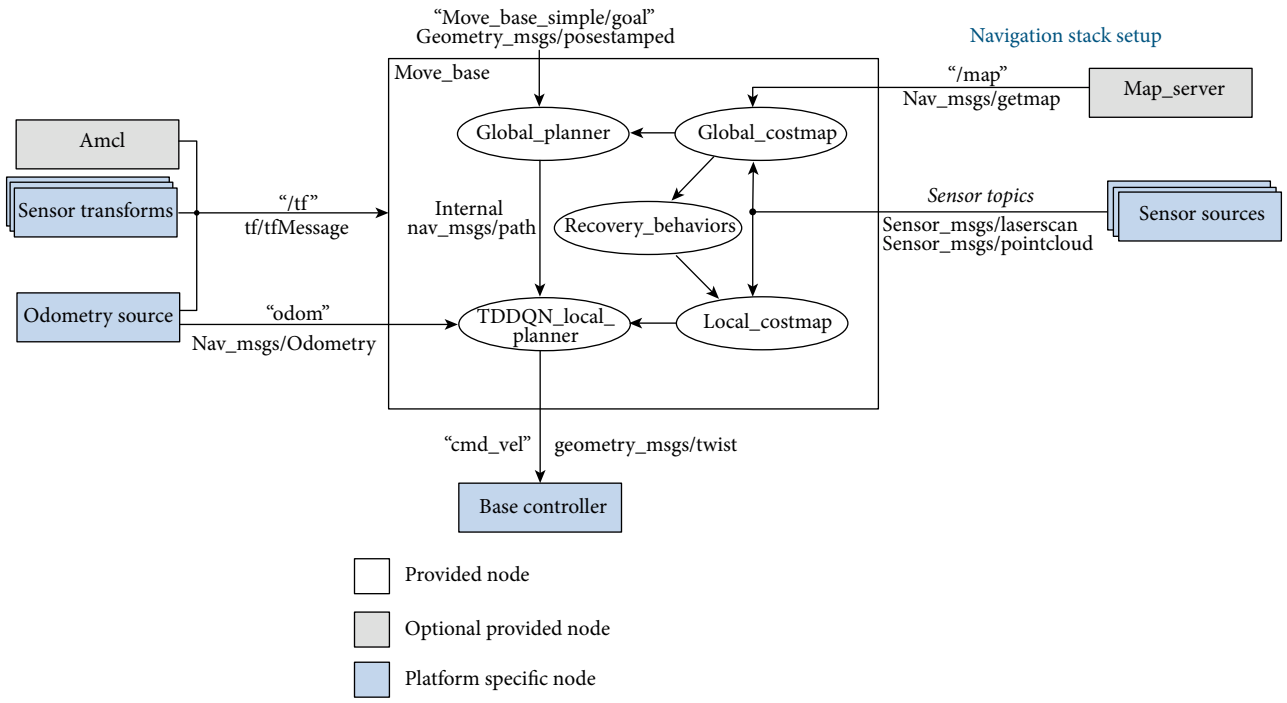


Figure 10: ROS navigation framework.

network loss function of DDQN algorithm and TDDQN algorithm. Each unit represents the average loss function value for ten periods. Figure 8(a) shows that the loss function value is 10^7 in the initial stage of training, but after 200 periods of training, the loss function value of TDDQN rapidly approaches the x-axis, while DDQN requires more than 400 periods of training. Figure 8(b) shows that Q network of TDDQN tends to converge after training for 10,000 periods, the loss function value is about 0.25, while the loss function value of DDQN is about 0.8. Figure 9 shows the average cumulative rewards curve. Each unit represents the average cumulative reward per 100 periods. This curve indicates that the average reward increases gradually with the increase of training time. After 10,000 periods of training, the reward value tends to be stable, and the average cumulative reward is greater than 5.

4. Test on a Dynamic Slope Ground Using the ROS Framework

In order to verify the effectiveness of the algorithm in slope environment, a wheeled mobile robot with lidar is adopted to realize autonomous navigation using ROS framework [18], as shown in Figure 10. The *move_base package* in ROS provides users with local path planning algorithm, namely dynamic window method and track-reckoning method. In this paper, TDDQN local path planning algorithm is transplanted into ROS in the form of plug-ins, which is encapsulated in the pure virtual function library *base_local_planner* of nov_core as subclass, namely, *TDDQN_local_planner*. The output of the deep reinforcement learning algorithm is the motion in eight directions on the slope, and then maps the motion to the forward and corner motions.

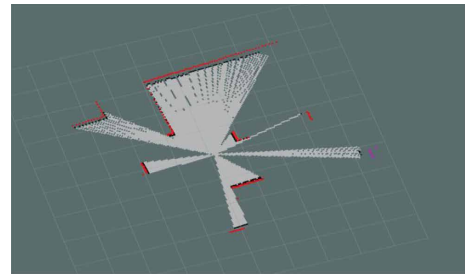


Figure 11: Local environment map.

The path planning of the slope dynamic environment is to navigate without using the SLAM map built before. Figure 11 is the local environment map, on which the lidar can only detect local information in real time, while dark areas are unknown. The agent avoids obstacle by the local path planning *TDDQN_local_planner*, the starting position is set as the current position of the agent, the target point is set as the unknown area, and the unknown region between the starting point and the target point is regarded as the movable region. The schematic diagram of path planning is shown in Figure 12.

The dynamic slope environment layout is shown in Figure 13. In the figure, the red arrow represents the global path planned in advance, which is blocked by obstacles no. 1, no. 2 and no. 3. Therefore, the agent needs to re-adjust the path to reach the target position without encountering obstacles.

The time series diagram of the path planning of the agent on the dynamic slope ground is shown in Figure 14. When the agent is 20 cm away from the obstacle no. 1, it waits for the obstacle to move downward. When the obstacle no. 1 leaves

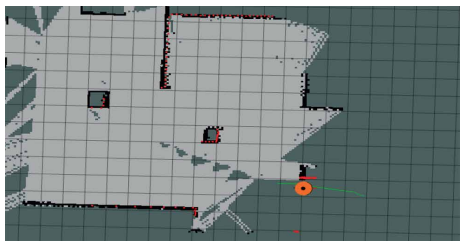


Figure 12: Schematic diagram of global path planning.

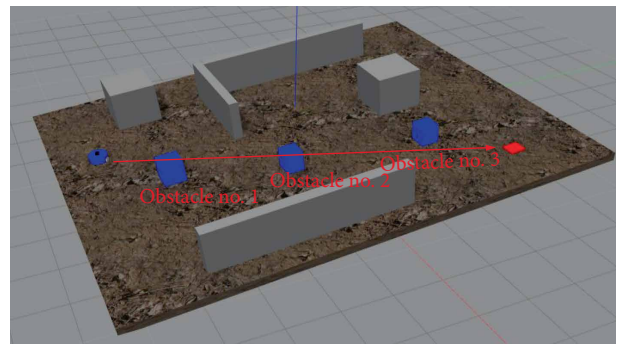


Figure 13: Actual dynamic slope environment.

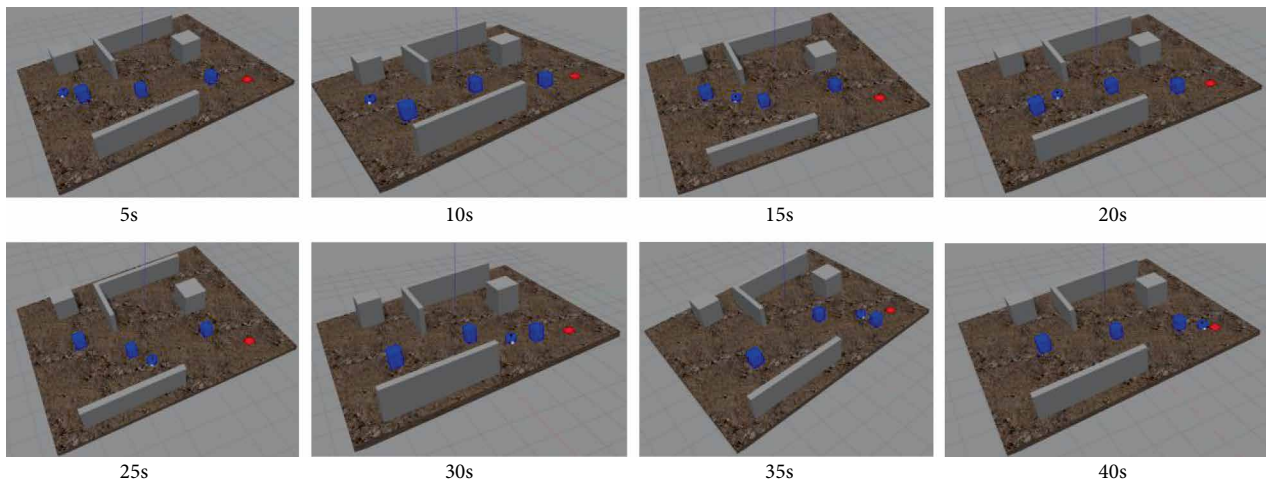


Figure 14: The agent moving on a dynamic slope ground.

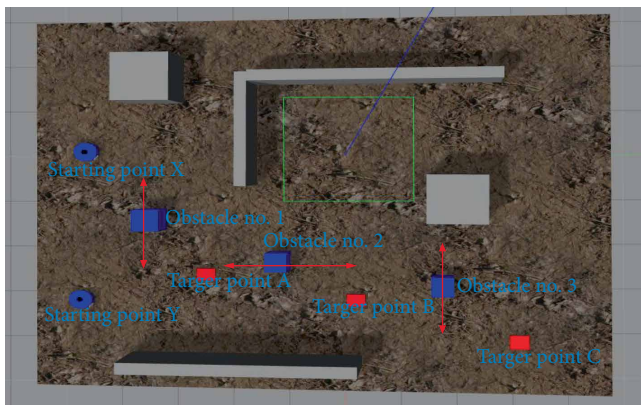


Figure 15: Dynamic scene test diagram of TDDQN and D* algorithm.

the global path of the agent, the agent moves downward to the right. At this time, the agent is blocked by the obstacle no. 2, and continues to wait for the obstacle no. 2 to move to the right. When the obstacle no. 2 leaves the global path of the agent, the agent continues to move to the lower right. Before the collision, the agent senses the obstacle no. 3 and waits for the obstacle no. 3 to move down. When the obstacle no. 3 leaves the global path of the agent, the agent reaches the destination and completes the path planning. The whole process shows that the agent trained by TDDQN can use the data of lidar to perceive obstacles in advance to avoid obstacles and complete the task of path planning on the dynamic slope ground.

Table 2: Comparison of TDDQN and D* experimental results. (Time (s)).

Algorithm	Start point	Target point A	Target point B	Target point C
D*	X	55.2	51.6	52.1
	Y	32.8	49.7	48.8
TDDQN	X	45.3	39.8	41.4
	Y	27.4	38.6	39.2

D* is Dynamic A* (D-Star, Dynamic A Star), a dynamic heuristic path search algorithm proposed by Dr. Stentz of Carnegie Mellon Robot Center, which is mainly used for robot dynamic path planning. This paper uses the dynamic scene shown in Figure 15 to conduct comparison test with D* algorithm. There are five points on the map: starting point X, starting point Y, target point A, target point B, target point C, and three blue dynamic moving obstacles. Let the agent arrive at the target point A, the target point B and the target point C in turn from the starting point X, and arrive at the target point A, the target point B and the target point C in turn from the starting point Y. Record the time spent and take it as a cycle.

Test 100 times with the same step to take the average, and get the time results as shown in Table 2.

It can be analysed from Table 2 that the TDDQN algorithm spent less time from the two starting points to the target points A, B and C than D* algorithm. Since TDDQN algorithm discards detected incomplete and over-detected paths by optimizing tree structure, the time efficiency was significantly better than D* algorithm.

5. Conclusions

The path planning method of wheeled robot on dynamic slope ground is studied in this paper, and the Tree Double Deep Q-Network dynamic path planning algorithm based on Deep Reinforcement Learning is proposed. Through learning actions, the agent selects a series of actions according to the current local navigation area, constantly changing its position, and finally reaching the target position. In the initial stage of training, DDQN spends twice as much time as TDDQN to stabilize the loss function value. The Q network of TDDQN tends to converge after training for 10,000 periods. The loss function value is about 0.25, and the average cumulative reward is greater than 5, while the loss function value of DDQN is about 0.8. Experiments show that the TDDQN algorithm can flexibly use lidar data for local path planning on dynamic slope ground, and has faster convergence rate and lower loss function value than DDQN algorithm. Compared with the traditional dynamic path planning D* algorithm, TDDQN can quickly select the optimal path through the optimized tree structure, making the algorithm more efficient in time.

In order to better adapt to the complex slope ground, the next step of this paper is to increase the number of moving obstacles and multiple dynamic scenes for simulation training, so that the algorithm has more application scenarios and better generalization ability, and the algorithm will be transplanted to the actual wheeled robot for experimental testing.

In order to better adapt to the complex slope ground, the next step of this paper is to increase multiple moving obstacles and multiple dynamic scenes for simulation training, and optimize the ε-greedy exploration efficiency of the algorithm. An improved strategy will be proposed to force exploration by adding Noisy Net to the network, which can accelerate the convergence rate of TDDQN. Making the algorithm has more application scenarios and better generalization ability. The algorithm will be transplanted to the actual wheeled robot for experimental testing.

Data Availability

The data used to support the findings of this study are available from the corresponding author upon request.

Conflicts of Interest

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

Acknowledgments

This work was supported by the Natural Science Foundation of Heilongjiang Province of China (Grant No. E2017049).

References

[1] M. A. H. Ali and M. Mailah, "Path planning and control of mobile robot in road environments using sensor fusion and active force control," *IEEE Transactions on Vehicular Technology*, vol. 68, no. 3, pp. 2176–2195, 2019.

[2] Y. Sun, J. Cheng, G. Zhang, and H. Xu, "Mapless motion planning system for an autonomous underwater vehicle using policy gradient-based deep reinforcement learning," *Journal of Intelligent and Robotic Systems*, vol. 6, pp. 1–11, 2019.

[3] O. Khatib, "Real-time obstacle avoidance for manipulators and mobile robots," *International Journal of Robotics Research*, vol. 5, no. 1, pp. 90–98, 1986.

[4] H. E. Romeijn and R. L. Smith, "Simulated annealing for constrained global optimization," *Journal of Global Optimization*, vol. 5, no. 2, pp. 101–126, 1994.

[5] A. Reshamwala and D. P. Vinchurkar, "Robot path planning using an ant colony optimization approach: a survey," *International Journal of Advanced Research in Artificial Intelligence*, vol. 2, no. 3, pp. 65–71, 2013.

[6] R. S. Sutton and A. G. Barto, "Reinforcement learning: an introduction," *IEEE Transactions on Neural Networks*, vol. 9, no. 5, p. 1054, 1998.

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

[8] V. H. Hado, G. Arthur, and S. David, "Deep reinforcement learning with double Q-learning," in *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence (AAAI-16)*, Google DeepMind Computer Science, 2015.

[9] S. Tom, Q. John, and A. Ioannis, *Prioritized Experience Replay*, Google DeepMind Computer Science, 2015.

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

[11] L. Tai and M. Liu, *Towards Cognitive Exploration through Deep Reinforcement Learning for Mobile Robots*, arXiv preprint, 2016.

[12] Z. Jie, X. Liang, J. Feng, X. Jin, W. Lu, and S. Yan, "Tree-structured reinforcement learning for sequential object localization," in *International Conference on Neural Information Processing Systems*, pp. 127–135, Neural Information Processing Systems (NIPS), Barcelona, Spain, 2016.

[13] Y. Wu and B. Zeng, "Trajectory tracking and dynamic obstacle avoidance of mobile robot based on deep reinforcement learning," *Journal of Guangdong University of Technology*, vol. 36, no. 1, pp. 42–50, 2019.

[14] G. Zuo, T. Du, and L. Ma, "Region proposal generation for object detection using tree-DDQN by action attention," *Journal of Electronics & Information Technology*, vol. 41, no. 3, pp. 666–673, 2019.

[15] X. Lei, Z. Zhang, and P. Dong, "Dynamic path planning of unknown environment based on deep reinforcement learning," *Journal of Robotics*, vol. 2018, pp. 1–10, 2018.

[16] J. Pan, X. Wang, Y. Cheng, and Q. Yu, "Multisource transfer double DQN based on actor learning," *IEEE Transactions*

on Neural Networks and Learning Systems, vol. 29, no. 6, pp. 2227–2238, 2018.

[17] A. Martín, "TensorFlow: learning functions at scale," *ACM Sigplan Notices*, vol. 51, no. 9, p. 1, 2016.

[18] D. Shen, Y. Huang, Y. Wang, and C. Zhao, "Research and implementation of SLAM based on LIDAR for four-wheeled mobile robot," in *2018 IEEE International Conference of Intelligent Robotic and Control Engineering (IRCE)*, pp. 19–23, IEEE, Lanzhou, China, 2018.