Path Planning of Unmanned Surface Vehicle Port Docking Based on Improved Double Deep Q-Network

Xu Hou
Institute of Artificial Intelligence and Marine Robotics
College of Marine Electrical Engineering
Dalian Maritime University
Dalian, China
xuhou@dlmu.edu.cn

Meng Joo Er*
Institute of Artificial Intelligence and Marine Robotics
College of Marine Electrical Engineering
Dalian Maritime University
Dalian, China
mjer@dlmu.edu.cn

Tianhe Liu
Institute of Artificial Intelligence and Marine Robotics
College of Marine Electrical Engineering
Dalian Maritime University
Dalian, China
tianheliu@dlmu.edu.cn

2022 5th International Conference on Intelligent Autonomous Systems (ICoIAS) | 978-1-6654-9838-8/22/$31.00 ©2022 IEEE | DOI: 10.1109/ICOIAS56028.2022.9931209

***Abstract*—In this paper, a novel path planning scheme based on improved Deep Q-Network (DQN) algorithm for Unmanned Surface Vehicle (USV) docking in complex port environment is proposed. First, the port map is pre-processed for edge expansion, and the level of expansion is set according to the complexity of the port environment and the strength of wind. Next, with the view of solving the problem of slow convergence speed and instability of traditional DQN algorithm, an improved Double DQN (DDQN) algorithm with prioritized experience replay and hyperparameter adaptive control is proposed. In order to strike a balance between exploration and exploitation, a dynamic ε scheme is devised to reduce network stabilization time. Finally, the B-spline technique is used to smoothly interpolate the obtained path coordinate points to obtain a viable USV path for port docking. Simulation results demonstrate the effectiveness and efficiency of the proposed scheme.**

***Index Terms*—DDQN; Hyperparameter adaptive control; Path planning; USV**

I. Introduction

Unmanned Surface Vehicles (USVs) have been widely developed and deployed in the community of marine science and engineering [1]. Path planning for USVs has received extensive attention from many researchers around the world. Traditional path planning algorithms, including Dijkstra algorithm [2], A* algorithm [3], etc, generate desired paths with more turning points, resulting in poor real-time performance. Some algorithms based on biological evolution, such as genetic algorithms [4], particle swarm optimisation [5], and ant colony optimisation [6], suffer from the drawback that convergence is not guaranteed and applications are limited. In recent years, due to the advancement of computer power and deep learning techniques, Deep Reinforcement Learning (DRL) has been become an alternative approach towards solving the problem of USV path planning.

Google's DeepMind team combined Q-learning algorithm [7], [8] with neural network and proposed Deep Q-Network (DQN) algorithm [9] which is successfully applied to Atari and Starcraft [10]. Some researchers have applied the DQN algorithm to path planning and achieved some interesting results. In [11], the DQN network structure was improved to hasten slow convergence of DQN algorithm in USV path planning. In [12], the authors propose a novel DRL-based collision avoidance algorithm. A safe path has been successfully planned by using the Double DQN (DDQN) algorithm and Dueling DQN algorithm. In [13], an optimized Dueling DQN algorithm is proposed. This algorithm has better exploration efficiency and convergence speed than the traditional DQN algorithm in both dynamic and static environments. In [14], the authors improve the action space of the DQN algorithm using the artificial potential field method. The improved algorithm can effectively avoid obstacles. In [15], the authors use DQN algorithms in formation control to successfully accomplish obstacle avoidance and path planning tasks.

In spite of many success stories, existing DQN algorithms in path planning still have the following shortcomings: (1) Convolutional Neural Network (CNN) has slow convergence speed and long training time [16] and hyperparameters need to be set manually. (2) DQN only works in discrete spaces.

In this context, this paper mainly addresses with the problem of slow convergence speed and long training time of DQN algorithm for USV path planning. To be more specific, a DDQN algorithm with prioritized experience replay and hyperparameter adaptive control is proposed. The main contributions of this paper are as follows:

(1). The port map is pre-processed for edge expansion. The expansion level, which corresponds to a penalty value, is assigned according to the strength of wind and the complexity of the port environment.

(2). The DDQN algorithm is employed to solve the overestimation problem of DQN. When the model is updated iteratively, a dynamic ε scheme is devised to balance the conflicting issues between exploration and exploitation in the training process.

978-1-6654-9838-8/22/$31.00 ©2022 IEEE

(3). A prioritized experience replay mechanism is developed to extract data from the experience pool and improve the convergence speed of the training process. The Population Based Training (PBT) hyperparameter search algorithm is exploited to update the learning rate online and improve stability and final performance of the model.

The rest of this paper is organized as follows. Section II breifly presents some prelimary knowledge including DQN and DDQN. Section III presents the USV port docking method with prioritized experience replay and hyperparameter adaptive control. Simulation results are presented in Section IV to show the effectiveness and efficiency of proposed method. Conclusions are drawn in Section V.

II. PRELIMINARIES

A. Q-learning and Deep Q-Network

Reinforcement Learning (RL) is typically employed to solve sequential decision problems, which emphasize on what actions the agent may take to achieve maximum return.

The training process of RL is modeled using a Markovian decision process, which is described by a five-tuple $< S, A, P, R, \gamma >$, where S is a finite set of states, A is a finite set of actions, P is the state transfer probability, R is the reward function, and γ is the discount factor, which is used to calculate the cumulative reward. The objective function to be learned is the control policy $\pi : S \rightarrow A$, which outputs a suitable action a from the set A given s in the set S of current states.

In the Markov decision process, at each discrete time step t , the agent selects an action a_t based on current state s_t. The environment generates a reward $r_t = r(s_t, a_t)$, and the agent is in a subsequent state $s_{t+1} = \delta(s_t, a_t)$.

Q-learning first evaluates the value function and then uses the value function to improve the current policy. The evaluation of the value function is the key factor for successful execution of Q-learning. For the state value function $Q(s, a)$, it is represented as a Q-table where each value is a state-action pair. The Q-learning algorithm is governed by

$$Q\left(s_t, a_t\right) \leftarrow Q\left(s_t, a_t\right) + \alpha\left[r_t + \gamma \max_a Q\left(s_{t+1}, a\right) - Q\left(s_t, a_t\right)\right] \tag{1}$$

Based on Q-learning, the DQN algorithm is modified in three main ways: (1) The value function is approximated via a CNN; (2) Experience replay mechanism is used for training; (3) The target network is set to handle bias in Temporal-Difference (TD) algorithm separately.

The value function of Q-learning is a linear function of parameters, while the CNN used in DQN is a non-linear approximation function. Both of them are parametric approximation methods although the approximation algorithms are different. In DQN algorithm, the value function $Q(s, a; \theta)$ corresponds to a set of parameters, and in the CNN, the parameters are the weights of each layer of the network, denoted by θ. Similar to updating and iterating over Q-tables, updating the value function is actually updating parameter θ. When the network structure is fixed, the value function is fixed.

When training a neural network, the data needs to be independently and identically distributed. However, the DRL states are related to the antecedents and successors, and training with data that are correlated can lead to non-convergence. The experience replay mechanism may break the correlation between data and smooth out changes in the distribution of observations. During training, the agent saves the data to the experience pool, and then extracts the data from the experience pool using uniform random sampling for the next training step. After making the two aforementioned improvements, the update equation is governed by

$$\theta_{t+1} = \theta_t + \alpha\left[r + \gamma \max_{a'} Q\left(s', a'; \theta\right) - Q(s, a; \theta)\right] \nabla Q(s, a; \theta) \tag{2}$$

The TD target is $r + \gamma \max_{a'} Q\left(s', a'; \theta\right)$ and the network parameter used to calculate the action value function for the TD target is θ. The parameter θ is the same as the network parameter used for the value function to be approximated in the gradient calculation, which leads to correlation between observations and could result in training instability. The network used for the value function approximation is updated at each step. The target network called TD network is set to handle target values in the TD algorithm separately with parameters of θ^-. The parameters of the target values are updated periodically. The value function of parameter θ is updated as follows:

$$\theta_{t+1} = \theta_t + \alpha\left[r + \gamma \max_{a'} Q\left(s', a'; \theta^-\right) - Q(s, a; \theta)\right] \nabla Q(s, a; \theta) \tag{3}$$

In summary, the structure of the DQN algorithm is shown in Fig.1.

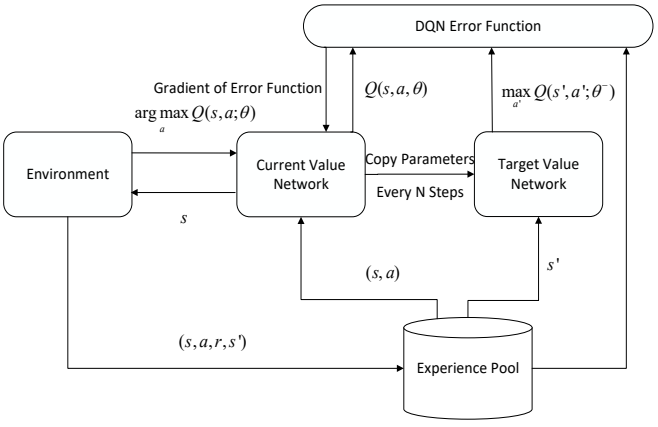


Fig. 1. Structure of DQN algorithm

B. Double Deep Q-Network

Although DQN makes many improvements, its framework is still Q-learning. As foreshadowed, DQN adopts a CNN

to approximate value function and uses experience replay to improve sample utilization as well as setting the target network to handle target values separately. However, the use of these techniques does not address the overestimation problem inherent in Q-learning.

The overestimation problem arises from the maximization operation in Q-learning governed by equation (1). Each point of the value function is overestimated by different magnitudes in different states, resulting in uneven amount of overestimation. As such, overestimation of the value function affects the final policy, resulting in a sub-optimal policy.

The approach to resolve this problem is to implement the selection of actions and evaluation of actions with separate value functions. This approach is termed DDQN.

The main idea of the DDQN approach is the following: in the calculation of the TD target, an action a^* is first selected which maximizes $Q(s_{t+1}, a; \theta)$ at state s_{t+1}. After selecting a^*, the TD target is constructed using the network parameters of the value function at a^*. Therefore, the TD target for the DDQN is given by

$$\begin{aligned} Y^{Double} =& r_{t+1} + \gamma Q(s_{t+1}, \\ & \arg\max_a Q(s_{t+1}, a; \theta_t^-); \theta_t) \end{aligned} \tag{4}$$

III. Methodology

A. Prioritized Experience Replay

The uniform distribution used in DDQN is not efficient for data extraction. The primary reason is that the previously stored data does not play an equally important role in the learning process of the agent. If data of high importance is not sufficiently useful, the convergence speed can be slowed. The prioritized replay mechanism is to increase the sampling weight of the states with high learning efficiency and break away from uniform sampling. This makes the experience pool more efficient and allows useful experiences to be deployed in the training with increased probability.

Different samples should have distinct usage weights. The agent needs to increase the amount of updates and can choose a state with large TD bias δ. The larger the TD bias, the larger the gap between the value function and the TD target. A larger update of the agent can improve the learning efficiency of that state.

Let the TD bias at data point i be δ_i, the sampling probability at that sample is given by

$$P(i) = \frac{p_i^\alpha}{\sum_k p_k^\alpha} \tag{5}$$

where $\alpha \in [0, 1]$ represents the degree of priority use, with $\alpha = 0$ for uniform sampling, and $\alpha = 1$ for greedy sampling. p_i is determined by the TD bias δ_i and is obtained as follows:

$$p_i = |\delta_i| + \epsilon \tag{6}$$

The estimate of the value function is a biased estimate when using probability sampling with prioritized experience replay, because the sampling distribution and the distribution of the value function are two completely different. To correct such bias, it is necessary to multiply by a sampling factor $\omega_i = (\frac{1}{N} \cdot \frac{1}{P(i)})^\beta$, where β represents the degree of correction.

The DDQN pseudo-codes with prioritized experience replay are shown in Algorithm 1.

Algorithm 1 Double DQN with Prioritized Replay

Input: minibatch k, step-size η, replay period K and size N, exponents α and β, budget T.

Initialize reply memory $M = \emptyset$, $\Delta = 0$, $p1 = 1$
Observe S_0 and choose $A_0 \sim \pi_\theta(S_0)$
for $t = 1$ to T **do**
  Observe S_t, R_t, γ_t
  Store transition $(S_{t-1}, A_{t-1}, R_t, \gamma_t, S_t)$ in M with maximal priority $p_t = \max_{i<t} p_i$
  if $t \equiv 0$ mod K **then**
    for $j = 1$ to k **do**
      Sample transition $j \sim P(j) = p_j^\alpha / \sum_i p_i^\alpha$
      Compute importance-sampling weight $\omega_j = (N \cdot P(j))^{-\beta} / \max_i \omega_i$
      $\delta_j = R_j + \gamma_j Q_{target}(S_j, \arg\max_a Q(S_j, a)) - Q(S_{j-1}, A_{j-1})$
      Update transition priority $p_j \leftarrow |\delta_j|$
      Accumulate weight-change $\Delta \leftarrow \Delta + \omega_j \cdot \delta_j \cdot \nabla_\theta Q(S_{j-1}, A_{j-1})$
    end for
    Update weights $\theta \leftarrow \theta + \eta \cdot \Delta$, reset $\Delta = 0$
    Copy weights into target network $\theta_{target} = 0$
  end if
  Choose action $A_t \sim \pi_\theta(S_t)$
end for

B. Dynamic $\varepsilon - greedy$ Policy

The goal of RL is to obtain maximum cumulative reward value. In the training process, there are two approaches, namely exploration and exploitation. Exploration revolves around distributing opportunities evenly into the action space while exploitation involves selecting the action with the greatest reward for execution. Exploration and exploitation are two conflicting actions, and enhancing one action may weaken the other one. To obtain maximum cumulative reward, we need to strike a balance between the two actions.

The $\varepsilon - greedy$ algorithm is a compromise between exploration and exploitation. When performing action selection in the current state, exploration with a probability of ε selects an action with a uniform distribution. On the other hand, exploitation with a probability of $1 - \varepsilon$ chooses the action with the highest current reward. By choosing an appropriate parameter ε, exploration and exploitation can be balanced.

In this paper, the $\varepsilon - greedy$ algorithm is optimized. In early stage of the training process, more exploration is needed to fully develop the environment. The possibility of finding

more paths facilitates the convergence of training results, which requires larger ε values. In the later stage of training, more exploitation of the previous data is needed to find an optimal policy, which requires a smaller ε value. At this time, the output of the neural network employs a larger probability to select the action in the current training results and combines with the exploration method to continuously adjust and improve the policy to find an optimal result. In this paper, the dynamic ε parameter approach is used. The calculation of ε is given by

$$\varepsilon = \frac{(t+\delta)^3}{v} e^{-\sqrt{t+\delta}} \tag{7}$$

where t is the number of training sessions, δ is the design offset, and v is the variable that varies with the environment. The ε value decreases with training time. In early stage of the training process, the probability of exploration is high, and USV attempts all directions. In later stage of the training process, when the coordinate point data become more and the value function approaches the final result, it is exploited with high probability. In our approach, USV chooses larger reward values at each step and keeps optimizing on the base path.

C. Edge Expansion

Map edge expansion, which can improve the safety of USV navigation, is depicted in Fig. 2.

In this paper, the map around the port is pre-processed with edge expansion. When driving into port, the USV needs to avoid collision with other obstacles. According to the complexity of the environment and the strength of the wind, the map edge expansion can be adjusted to handle the port environment flexibly.

The penalty value is set distinctly for different levels of edge expansion. The penalty value increases from the outer layer to the inner layer and becomes maximum when it reaches the real obstacle. When the USV deviates from its course and collides with an obstacle because of wind, the environment gives a large penalty value to weaken the effect of the wind on the USV path planning. In this context, adjustments are made according to the wind direction when the action selection is made afterwards, and reasonable decisions are obtained.

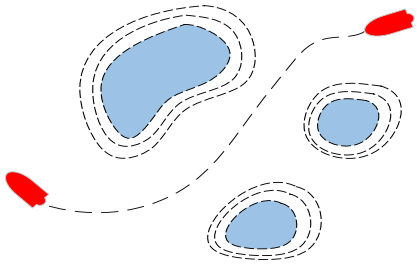


Fig. 2. Pre-processed map

D. Hyperparameter Adaptive Control

Evolutionary strategies (ES) are a class of methods that imitate natural evolution for solving parametric optimization problems. ES is efficient and easy to parallelize with excellent global search capabilities. One of the disadvantages of the DRL algorithm is the tendency to fall into local optimality and the difficulty to search for a global optimal solution. The two algorithms are complementary in this respect. In this paper, ES and DRL are combined to achieve better performance.

There are many hyperparameters involved in each step of the neural network training, and the adjustment of hyperparameters can affect accuracy of the final model. In DRL, the learning rate is usually set manually according to experience, but this approach is very time-consuming and is difficult to obtain an optimal solution. Two popular automatic adjustment methods are parallel search and sequential optimization. Parallel search sets several sets of parameters for training at the same time, which has the disadvantage of not utilizing mutual parameter optimization information and consuming large resources. Sequential optimization attempts optimization in a single model training, with the disadvantage of taking long time. The Googles DeepMind team proposes the PBT algorithm, which combines parallel search and sequential optimization. The PBT algorithm searches for the best performing hyperparameters in a population evolutionary manner.

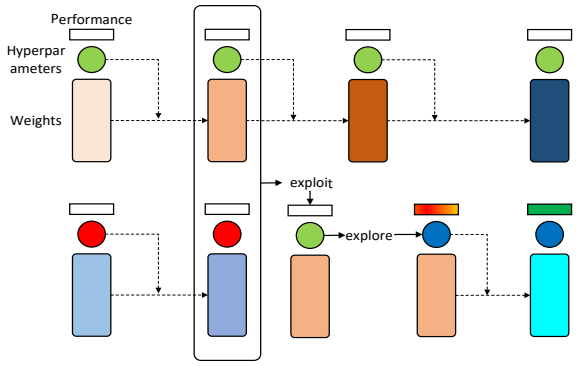


Fig. 3. Block diagram of PBT training

In this paper, the improved DDQN algorithm is trained using the PBT hyperparametric search algorithm. In Fig. 3, a random initialization sets up multiple models for parallel exploration. The model with good performance can eliminate the sub-optimal model in each sequence. The learning rate is updated online during training, and the entire network is trained in an iterative manner. The key idea of the PBT algorithm is based on exploration, comparison and evolution for model improvement. The training population is assigned, using 10-80 models. For each model of asynchronous parallelism, there is a whole set of neural networks corresponding to network weights and hyperparameters as well as performance parameters as depicted in Fig. 3. The PBT algorithm sets a checkpoint every training period and adjusts its own model

according to good or bad performance of other models. If its own model has better performance metrics, training continues. If the performance metrics of own model are not good, then replace it with better model parameters. This process is the exploit operation depicted in Fig. 3. The explore operation in Fig. 3 corresponds to adding random perturbations to continue training. Note that the model update is only performed after several iterations of the model, hence it is different from model iteration. Corresponding to Fig. 3, details of the key steps in the algorithm are defined as follows.

step1 Select hyperparameter learning rate.

step2 During the training process, each model performs one iteration which is also known as stochastic gradient descent, all of which update the network parameters.

step3 Evaluate the updated model with the last 10 episode reward values. Evaluation results are defined as Eval values.

step4 After a model in the population completes the designated number of iterations, the other members of the population also approximate the completion of the model update.

step5 Truncation selection, sort all models in the population by their Eval values. If the current Eval value is in the bottom 20% of the ranking, replace its own model parameters and hyperparameters by sampling another node uniformly from the top 20%.

step6 Perturb settings, each learning rate is randomly multiplied by a value in $[0.8, 1.2]$.

The PBT algorithm can adjust the learning rate of the improved algorithm online to enhance stability of the model and reduce convergence time. Its effectiveness will be demonstrated via a simulation experiment.

IV. Simulation Results and Discussions

In order to evaluate the proposed method, simulation experiments were conducted under the satellite map of Dalian Xinghai port, which is shown in Fig. 4(a). The map is binarized and then trained, as depicted in Fig. 4(b). In the training process, the map is divided into grid maps, which are shown in Fig. 4(c). The action output of the USV is the four directions, namely up, down, left, and right. The configuration used in the simulation example is shown in Table I. Experiment related parameters are given in Table II.

TABLE I
Configuration of the Experiment

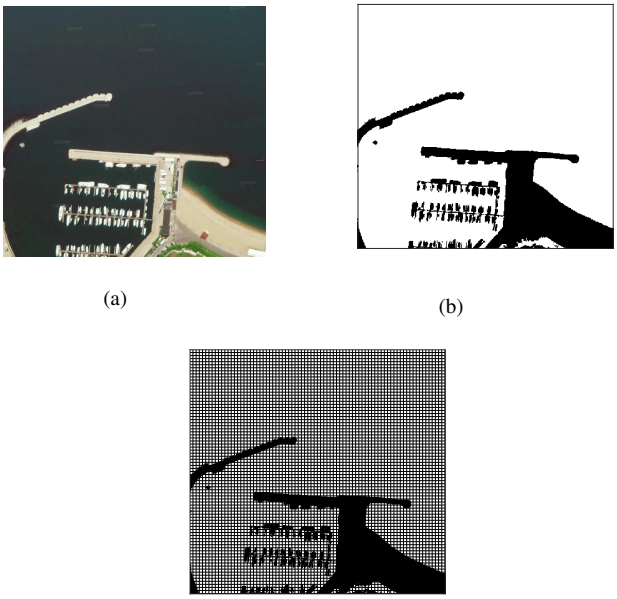
Configuration	Parameters
System	Ubuntu18.0
Graphics	NVIDIA 2080
Pytorch Framework Version	1.10.2
Python Version	3.9
CUDA Version	10.2

USV may deviate from the course when docking under the influence of wind. In general, the wind strength and direction are variable, and is hard to predict. External wind disturbance is simulated and the course is corrected accordingly. The wind which is randomly distributed in the port is set to three levels: low, medium and high, with four directions: east, south, west and north. The windless path is shown in Fig. 5, and the windy path in Fig. 5 is the deviated route after adding wind disturbance. In this paper, the map is pre-processed with edge expansion in windy conditions to avoid collision with obstacles, so the corrected route is not exactly the same as the original one. At the same time, if a collision occurs because of wind, a great penalty value is returned to balance the impact of wind in this area and ensure the safety of USV. The route after balancing interferences is shown as the corrected path in Fig. 5.

In this paper, the starting point is set outside the port, and the target location is the ship position inside the port. In order to reflect comprehensiveness of the algorithm, the starting points in two different directions are selected, and the key turning

TABLE II
Parameters of the Experiment

Parameters	Value
Minibatch	32
Replay Memory Size	10000
Discount Factor	0.9
Initial Learning Rate	0.0001
Initial Exploration	1
Number of Iterations In Model Update	$1 * 10^4$
Number of Parallel Nodes	10
α	0.5
β	0.5



(a) (b) (c)

Fig. 4. Port map

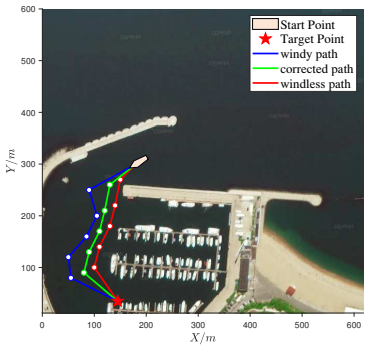
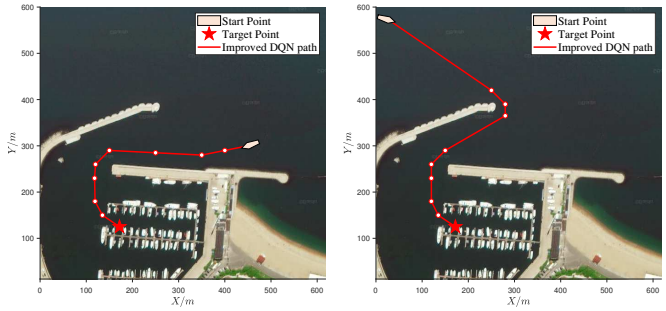


Fig. 5. Corrected path

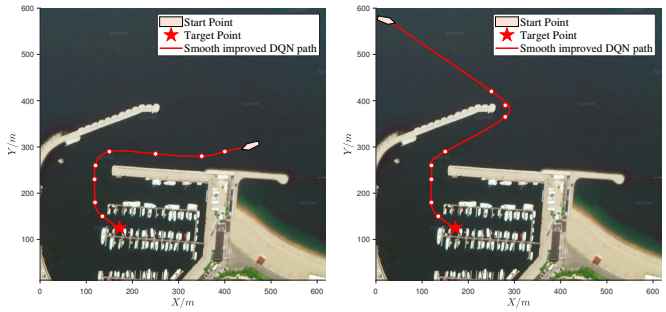
points are marked. In the end, there are no collisions in both Fig. 6(a) and Fig. 6(b), and safe paths have been successfully planned.



(a) (b)

Fig. 6. Final path

Obviously, the planned path has turning points that do not match the actual navigation needs of the USV since USVs generally travel in a smooth path. Therefore, this paper employs the B-spline technique for smooth interpolation. In comparing Fig. 7 with Fig. 6, it can be seen that the paths in both different directions become smooth, making the final route practically feasible.



(a) (b)

Fig. 7. Final feasible path

V. Conclusions

In this paper, an improved DDQN algorithm based on prioritized experience replay and hyperparameter adaptive control has been proposed for path planning in USV port docking in complex environment. With the objective of increasing the convergence speed of DDQN, prioritized experience replay technique is adopted. Dynamic $\varepsilon - greedy$ along with PBT hyperparameter searching algorithm is used to ensure better stability and system performance. In addition, the map is pre-processed via edge expansion technique to improve USV navigation safety. B-spline technique makes the final route practically feasible. Simulation results prove that the proposed method can effectively accomplish path planning for USV port docking task. In future work, USV dynamics models and continuous action processing will be researched.

References

[1] Z. Liu, Y. Zhang, X. Yu, and C. Yuan, "Unmanned surface vehicles: An overview of developments and challenges," *Annual Reviews in Control*, vol. 41, pp. 71–93, 2016.

[2] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische Mathematik*, vol. 1, no. 1, pp. 269–271, 1959.

[3] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE Transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 28–29, 1972.

[4] D. Whitley, "A genetic algorithm tutorial," *Statistics and Computing*, vol. 4, no. 2, pp. 65–85, 1994.

[5] R. Poli, J. Kennedy, and T. Blackwell, "Particle swarm optimization," *Swarm Intelligence*, vol. 1, no. 1, pp. 33–57, 2007.

[6] M. Dorigo, V. Maniezzo, and A. Colorni, "Ant system: optimization by a colony of cooperating agents," *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 26, no. 1, pp. 29–41, 1996.

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

[8] J. Kober, J. A. Bagnell, and J. Peters, "Reinforcement learning in robotics: A survey," *International Journal of Robotics Research*, vol. 32, no. 11, pp. 1238–1274, 2013.

[9] M. Volodymyr, K. Koray, S. David, A. A. Rusu, V. Joel, M. G. Bellemare, G. Alex, R. Martin, A. K. Fidjeland, and O. a. Georg, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[10] O. Vinyals, I. Babuschkin, W. M. Czarnecki, M. Mathieu, A. Dudzik, J. Chung, D. H. Choi, R. Powell, T. Ewalds, P. Georgiev *et al.*, "Grandmaster level in starcraft ii using multi-agent reinforcement learning," *Nature*, vol. 575, no. 7782, pp. 350–354, 2019.

[11] Z. Zhu, C. Hu, C. Zhu, Y. Zhu, and Y. Sheng, "An improved dueling deep double-q network based on prioritized experience replay for path planning of unmanned surface vehicles," *Journal of Marine Science and Engineering*, vol. 9, no. 11, p. 1267, 2021.

[12] Y. Fan, Z. Sun, and G. Wang, "A novel reinforcement learning collision avoidance algorithm for usvs based on maneuvering characteristics and colregs," *Sensors*, vol. 22, no. 6, p. 2099, 2022.

[13] X. Wu, H. Chen, C. Chen, M. Zhong, S. Xie, Y. Guo, and H. Fujita, "The autonomous navigation and obstacle avoidance for usvs with anoa deep reinforcement learning method," *Knowledge-Based Systems*, vol. 196, p. 105201, 2020.

[14] L. Li, D. Wu, Y. Huang, and Z.-M. Yuan, "A path planning strategy unified with a colregs collision avoidance function based on deep reinforcement learning and artificial potential field," *Applied Ocean Research*, vol. 113, p. 102759, 2021.

[15] X. Zhou, P. Wu, H. Zhang, W. Guo, and Y. Liu, "Learn to navigate: cooperative path planning for unmanned surface vehicles using deep reinforcement learning," *IEEE Access*, vol. 7, pp. 165 262–165 278, 2019.

[16] J. Gu, Z. Wang, J. Kuen, L. Ma, A. Shahroudy, B. Shuai, T. Liu, X. Wang, G. Wang, J. Cai *et al.*, "Recent advances in convolutional neural networks," *Pattern recognition*, vol. 77, pp. 354–377, 2018.