A modular robotic arm configuration design algorithm based on prioritized experience playback and Double DQN algorithm

NAME,,,,

School of Mechanical Engineering, University of shanghai science and technology, Shanghai, China

*Corresponding author: XXXXX@st.usst.edu.cn

Abstract. The reconfigurable modular collaborative robotic arm, due to its component reconfigurability, allows the arm to adapt to tasks deployed in different scenarios. Determining the optimal robotic arm configuration faces exponential complexity in real-world applications due to a large number of arrangements of module connectivity. In this paper, a Double DQN modular robotic arm reconfiguration algorithm with prioritized empirical playback is proposed, the idea of which intends to search for the optimal configuration of the robotic arm under different tasks. The Double DQN algorithm performs the evaluation and selection of actions through different networks, which can be used to solve the overestimation problem existing in reinforcement learning with Q-Learning.In addition, the experience playback mechanism is improved based on SumTree to achieve preferential experience playback, which enables the algorithm to effectively use historical experience, to avoid the algorithm from falling into local optimal solutions. In this study, by establishing a small robotic arm module library and conducting comparative experiments with the traditional DQN-based algorithm on the PyBullet simulation platform, the results show that this method has higher search efficiency and accuracy. This research result provides a new solution idea and tool for the robot configuration search problem and promotes the development of intelligence and automation in the field of robot applications.

Keywords: Modular reconfigurable robotic arm; Double DQN; Preferential experience playback; Deep reinforcement learning

1. INTRODUCTION

With the development of the economy and technology, China has received more and more financial, technical, and policy support in the field of aviation, and the space industry has been highly valued. In the face of a complex space environment and the diversity of tasks, multi-function robotic arm in space increasingly has significance. Nowadays, large instruments used in space are inevitably faulty in complex environments, and because of their high manufacturing costs, local damage can also cause large financial damage. Robotic arms in the space station often have to consider the cost and economical and are usually restricted by quality control and space limitations [1]. At the same time, in the special space environment, facing the problem of limited resources and the complexity and diversity of exploration tasks, the modular reconfiguration robotic arm makes it an ideal solution under its excellent modular replaceability and versatility. Tu Yuanyuan et al. proposed a reconfigurability determination and quantification method in line with the actual operating environment of spacecraft, taking into account the influence of interference [2].

A modular robotic arm relative to the traditional robotic arm, for its structure, is characterized by its reconfigurability, which can be flexibly adapted to different complex environments. Each module has a specific function and quantification, through flexible combination and replacement, it can achieve the needs of different fields and application situations. Its function can be in the face of all kinds of different scales and types of work tasks, the robotic arm can be assembled and reconfigured twice or even more times, without the need to re-acquisition new functional machines and equipment, to minimize the cost of consumption. Therefore, it can effectively assist operations in the industrial field, aerospace field, and logistics field, and this modular design makes the robot system more flexible, expandable, and able to be customized and transformed according to the actual environmental requirements. However, when building a modular robotic arm, the number of possible modular reconfiguration models grows exponentially as the number of module types and connection methods is increased. With a huge design space, reconfiguration models need to be accurately evaluated against the question of whether they can achieve the tasks for which they were intended. This assessment involves planning and comparing the relative motion costs between candidate design models, which can result in the computational scale becoming too massive, so efficient algorithms and techniques are needed to accelerate the design assessment system and find the optimal modular configuration to achieve effective functionality of the modular robot for a given task.

Under the continuous development of information technology, intelligent algorithms represented by reinforcement learning are more often used in the field of robot control under their self-adaptive characteristics, as well as having the ability to enable robots to continuously trial and error, exploration, as well as the establishment of corresponding reward and punishment mechanisms instead of other algorithms, to enhance the flexible control strategies based on machine learning algorithms that continuously update and adjust themselves according to their tasks and environmental changes. Luo et al. proposed a self-reconfiguration algorithm based on a spherical modular robot for helping to cross various types of obstacles [3]. Whitman et al. proposed a DQN-best-first search algorithm based on the Deep Q Network (DQN) algorithm in deep reinforcement learning, by which the intelligence trained by the algorithm can solve the robot configuration synthesis problem and, in terms of execution efficiency, are superior to existing search algorithms [4]. However, the traditional DQN algorithm requires a large number of samples in the training process, which leads to the problem of low sampling efficiency, and the instability of its optimization process also causes the volatility of the training results and is prone to over-estimation and other problems.

To solve the above problems, some new improvement schemes based on the DQN algorithm have been successively proposed. Firstly, a Double DQN algorithm was proposed by Chen Zesheng et al. to solve the over-estimation problem of the traditional DQN, and according to the comparison of the simulation results based on the interference pattern selection method, the DDQN algorithm has a better performance than the DQN algorithm [9]. Subsequently, Schaul et al. proposed a preferred experience playback idea, which can be applied to the algorithm to enable it to make effective use of historical experience [5]. For a further improvement, Zhai et al. proposed a dual DQN algorithm combined with the preferred experience playback idea. This algorithm, compared to the traditional DQN algorithm, makes use of the preferred experience playback idea, which improves the convergence speed and ensures that the algorithm achieves better obstacle avoidance results in performing the obstacle avoidance task [6]. Through the fusion of the two thoughts, the algorithm can utilize prior experience to update the model in a more specific and effective scope, improving the overall performance of the algorithm.

To reconstruct the optimal robotic arm model for different complex operation scenarios, it is easier, faster, and more probable to judge and realize the optimal reconstruction model. In the same conditions, the reconfiguration model can be calculated more quickly, the cost of the reconfiguration model can be minimized, the number of modules to be selected in the design configuration is less, and the quality of the reconfigured mechanical weight is lighter. Now a Double DQN modular robotic arm reconstruction algorithm with an improved preferred experience playback idea is proposed, based on which, the experience playback mechanism is improved by using SumTree to achieve preferred experience playback, so that the algorithm can make effective use of the historical experience, thus avoiding the algorithm from falling into the local optimal solution [7,8]. The main contributions of this paper include:

(1) Establish a library of small modules for collaborative robotic arms, which can constitute a variety of tandem robotic arms. The model is converted to a URDF robot description file in XML format through the SW2URDF plug-in of SolidWorks so that it can be imported into the software for simulation.

(2) In this paper, a Double DQN algorithm with prioritized experience playback is proposed for modular robotic arm configuration design. The Double DQN algorithm solves the over-estimation problem in Q-Learning by utilizing different networks for action evaluation and selection, which improves the effectiveness of learning. To make better use of experience from history, this paper improves the experience playback mechanism and introduces the SumTree-based prioritized experience playback technique. This playback mechanism can effectively select important experiences for training, preventing the algorithm from falling into local optimal solutions at the late stage of training, thus enhancing the convergence speed and training efficiency of the algorithm.

(3) Simulation experiments are conducted on the PyBullet simulation platform using the method proposed in this paper, which is used to search for the optimal configuration of the robotic arm under different tasks. The experimental results show that the method successfully improves the success rate of searching for feasible configurations and finds lighter and fewer feasible configuration modules, which means that better configurations are obtained.

2. PROBLEM FORMULATION

2.1 Modelling of MRP

This paper aims to investigate the task of searching for the optimal configuration of a robotic arm for different environmental conditions and tasks in a real industrial environment. In the study of modular reconfigurable robots by Jinguo et al, reconfigurable robots can be classified into two categories, which are large reconfigurable robotic arms and small reconfigurable robotic arms [9]. The large-scale reconfigurable robotic arm has the advantages of flexibility, easy maintenance, and interchangeability, which can be more widely used. Therefore, this paper focuses on the tandem structure of a large-scale reconfigurable manipulator arm (Fig.1 branch shown by redarrow).

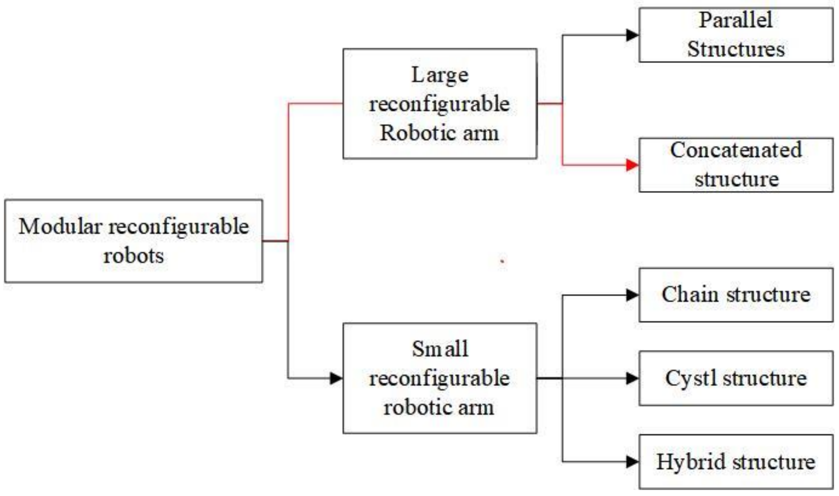
Modular reconfigurable robots
Large reconfigurable Robotic arm
Small reconfigurable robotic arm
Parallel Structures
Concatenated structure
Chain structure
Cystl structure
Hybrid structure

Fig.1 Categories of Modular Reconfigurable Robots

Establishing a robotic arm module library can improve design efficiency, reduce cost, increase system reconfigurability, and simplify the maintenance and update process, which brings greater convenience and flexibility to the development and application of robotic arms. According to relevant design experiences at home and abroad, three specifications of joint modules are usually used: shoulder joint, elbow joint, and wrist joint. At the same time, the mass of each joint is pre-assigned, in which the wrist joint does not exceed 2kg, the elbow joint does not exceed 3kg, and the shoulder joint does not exceed 4.5kg. The predefined parameters of the tandem robotic arm are shown in Fig (Fig.2). The three joint specification size parameters set in this paper are shown in the following table (Table 1).

Table 1.Predefined parameters for tandem robotic arms

Joint type	*Diameter*	*Height*	*Motion Range*
Shoulder Joint	*40mm*	*50-70mm*	*±160°*
Elbow Joint	*36mm*	*48-56mm*	*±145°*
Wrist Joint	*32mm*	*38-46mm*	*Approx. ±120°, Grip: Approx. 0-90°*

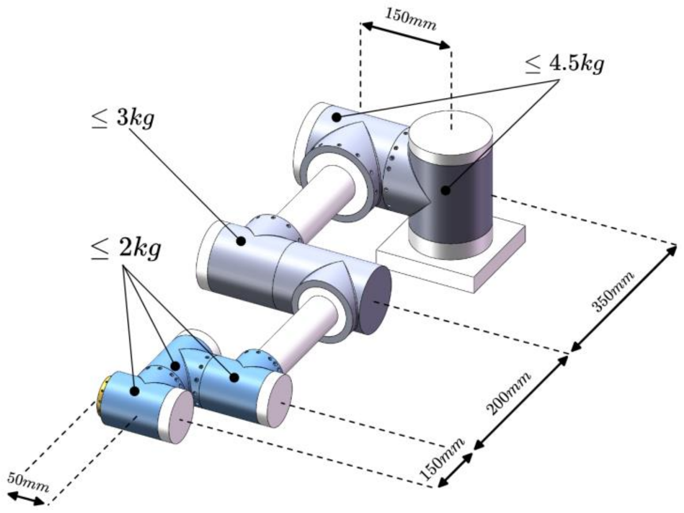

Fig.2 Predefined parameters for tandem robotic arms

A small module library was created using SolidWorks software, containing a total of 13 modules, including two models of base modules, three models of active modules and eight passive modules with different size and shape (Fig. 3). At the same time, the SW2URDF plug-in in SolidWorks was used to convert these models into URDF (Unified Robot Description Format) robot description files as XML format, so that the relevant training environment could be built in Pybullet simulation software to conduct simulation experiments.

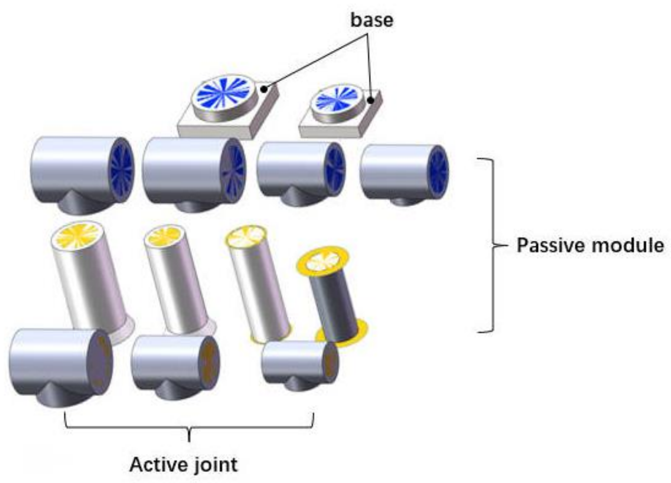

Fig.3 Robotic Arm Small Module Library

URDF is an XML file format for describing robot models. It can organize the physical, kinematic, and visual attributes of a robot model, including information about joints, sensors, colliders, inertia, links, etc. URDF files can be imported into robot simulation software such as Gazebo, PyBullet, etc., to load and simulate robot models. In this paper, each joint module has a standard input or output interface, using the characteristics of the URDF file, the "virtual linkage" can be defined by saving the input interface information or output interface information in its corresponding URDF file [11]. Take the T-joint module as an example, as shown in the figure below, where the orange "virtual links" indicate the output interfaces, and the blue "virtual links" indicate the input interfaces and the coordinate system definition rule of URDF requires that the geometry of the "virtual links" is defined in the URDF file of each "virtual link". The URDF rules for defining the coordinate system require that a coordinate system be established at the geometric center of each "virtual link", with the axes of the coordinate system parallel to the direction of the normal to the surface of the "virtual link" (in the direction of the z-axis). For output interfaces, the axis of the coordinate system points outwards, while for input interfaces, the axis of the coordinate system points inwards. (Fig.4)

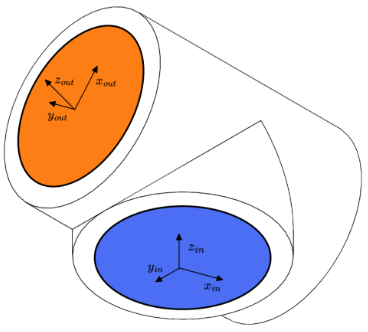

Fig.4 Coordinate definition of the T-joint module

To assemble multiple modules into a complete robot, it is necessary to define the assembly rules between URDF files. Taking the assembly between two T-joint models as an instance, the assembly is completed when the output coordinate system {out} of module {i-1} coincides with the input coordinate system {in} of module {i}. As shown in the figure below (Fig.5), the red arrow indicates the path, i.e., the output "virtual link" (orange) of module {i-1} is connected to the input "virtual link" (blue) of module {i} to complete the assembly.

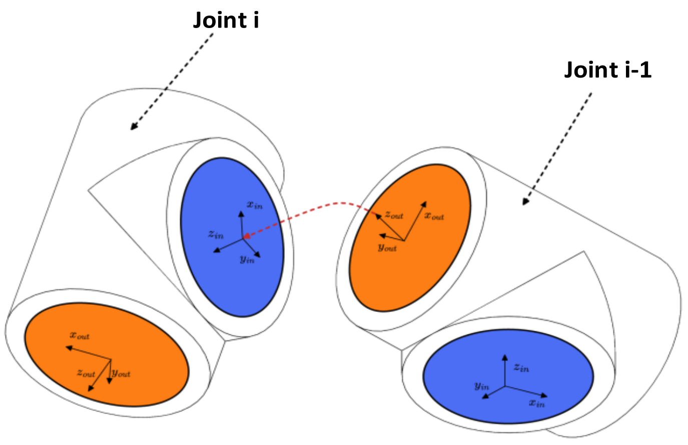

Fig.5 Assembly rules for modules

2.2 Modular Robot Design Based on Deep reinforcement learning

The design methodology in this paper aims to generate the optimal tandem robotic arm configuration to perform the task based on the given robotic arm module library and the task in different scenarios, and the optimal configuration is determined based on factors including the number of modules and mass of the complete robotic arm. The reconstruction process of the tandem robotic arm is shown in Fig. In this process, the base module is added first, the configuration C is formed by adding new joint modules to the base module, the new configuration C' is formed by continuing to add modules from the configuration library to the configuration C, and the configuration that ends only with the end-effector module is defined as the complete configuration. (Fig.6)

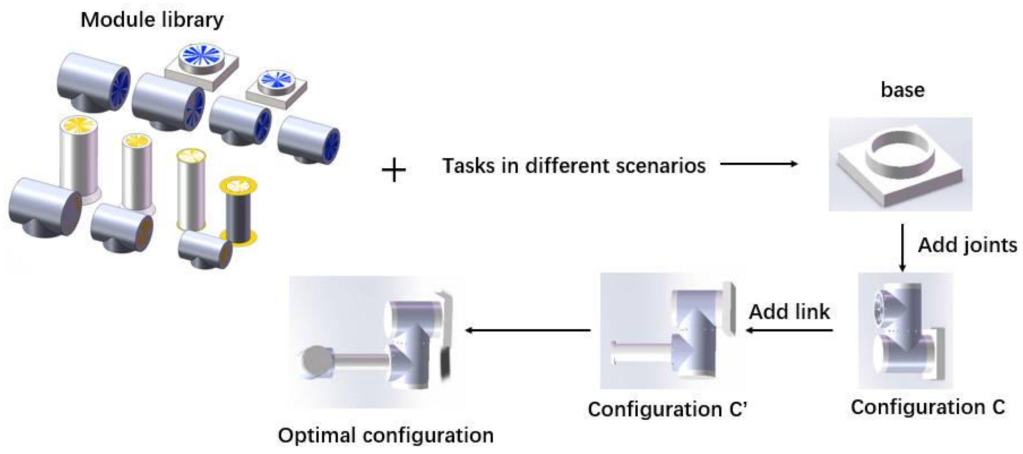

Fig.6 Reconfiguration process diagram of the tandem robotic arm

The configuration design problem of a modular tandem robotic arm can be effectively modeled and solved using the Finite Markov Decision Process (MDP). A quaternion structure is established, where S represents the set of state values of the environment, defined by the different configurations of the robotic arm. The action space A is defined by the available robot module library. R denotes the reward value obtained by adopting an action in a particular state, and π represents the strategy function.At each moment t, the reinforcement learning intelligence needs to select an action, i.e., the index value of a module $a_t \in \{1, 2, ..., N_m\}$, and add the corresponding module to the current configuration C. The only part of the state s_t that changes is the configuration C, so the state at the next moment depends only on the previous moment's state and the action of adding the module. At this point a new state S_{t+1} is produced, containing the new configuration C' and a reward from the environment r_t . The expected payoff of the reward that the Agent receives for choosing an action in the state is:

$$G_t = \sum_{t=0}^{+\infty} \gamma^t R_{t+1}$$

γ is the discount factor used to calculate the cumulative reward R, where the discount factor $0 \le \gamma \le 1$, and R_{t+1} is the value of the environmental reward obtained at time step t. The value function of strategy π taken at state st is the state value function. The state value function for taking strategy π at state it is:

$$V_\pi(s) = E_\pi[G_t \mid s_t = s]$$

Based on the strategy π, action at is used, then the value function of the action in the state it is:

$$Q_\pi(s,a) = E_\pi[G_t \mid s_t = s, a_t = a]$$

The core made of the algorithm uses reinforcement learning to estimate the optimal state-action-value function $Q^\star$, which can be defined by the Bellman optimality equation, the

$$Q^\star(s_t, a_t) = \max_\pi E[R_{t+1} + \gamma \underset{a' \in A}{max} Q^\star(S_{t+1}, a')]$$

The data and environment in reinforcement learning are dynamic, and through continuous interaction with the environment, the intelligent agent can iteratively explore and attempt different actions. The merit of these actions is evaluated based on the reward signals obtained.

2.3 Deep reinforcement learning DDQN

Q-learning is a reinforcement learning algorithm based on value iteration, which employs a tabular method to compute the valuation of Q-values corresponding to each possible state-action pair. In modular robotic arm configuration design, the high-dimensional continuous state and action space makes the traditional Q-value estimation based on the table method difficult. To solve this problem, Deep Q Network (DQN) was developed. By fitting the Q-value function through a neural network, it can directly input the states and output the Q-value of each action, thus avoiding the space complexity problem of maintaining a Q-table. Meanwhile, the nonlinear fitting ability of deep neural networks can better approximate complex Q-value functions.

Traditional DQNs use two neural networks with the same structure but different parameters. One network is used to update the parameters in real time while the other network is used to update the target Q-value. The update formula for the target Q-value is:

$$Q_{t\arg et} = R + \gamma \max_{a'} Q(s', a'; w^-)$$

However, the common DQN algorithm is to find the largest Q value for each action from the target Q network, which will select the overestimated value with high probability, which in turn leads to the overestimation problem. To solve this problem, the Deepmind team proposed DDQN in 2015, which solves this problem by independently training two separate Q-networks [10]. Double DQN introduces two separate neural networks (which also have the same structure and different parameters as DQN): one for selecting actions (action selection network), and the other for evaluating the value of the selected actions (target network). Instead of using the target network directly to select the maximum action value for the next state when calculating the target Q value, the action selection network is used to select the action:

$$a^* = \arg\max_a Q(s_{t+1}, a; w).$$

This action is used as the action for calculating the target Q value, and the selection of the optimal action is accomplished using the target network. The calculation of the target Q in Double DQN is shown in equation ():

$$Y_t^{DoubleQ} = R_{t+1} + \gamma \cdot Q\left(S_{t+1}, \arg\max Q(S_{t+1}, a; w^-)\right).$$

In this paper, the principle of Double DQN is applied to the modular robotic arm configuration design to solve the high dimensional and complex problem of robotic arm module reconfiguration.

3. METHOD

3.1 Double DQN for Module Selection

The paper aims is to find the optimal configuration of a robotic arm for different tasks so that it can accurately reach the specified task point. The robotic arm consists of a set of N_{m} modules with module index $m \in 1,2,...N_m$. Thus, the configuration optimization problem for a modular reconfigurable robot can be formulated as selecting a sequence of modules, the configuration consisting of these modules in a sequential order, which is capable of accomplishing the given task. The metrics and functions used in this paper for evaluating the optimality of the robotic arm are given below.

(1) Target point accessibility

The workspace objective (positional pose) is described by the set $T = [p, \hat{n}]$, which $p \in R^3$ denotes the coordinates of the end-effector in space and $\hat{n} \in R^3, \| \hat{n} \| = 1$ denotes the orientation of the end-effector. A sequence of modules C is selected, and to evaluate whether the configuration can reach the given objective, the position error ϵ_p and attitude error ϵ_n are set according to the joint accuracy of the robot, and definitions $p_{ee}(C, p, \hat{n})$ and $\hat{n}_{ee}(C, p, \hat{n})$ denote the output postures of the inverse kinematics solution corresponding to the positive kinematics for the given objective. The "reachability" function of the configuration C for a given pose is defined by Eq:

$$reach(C,T) = \begin{cases} 10 & \| p - p_{ee}(C, p, \hat{n}) \| \le \epsilon_p \ and \\ & 1 - \hat{n} \cdot \hat{n}_{ee}(C, p, \hat{n}) \le \epsilon_n \\ 0 & others \end{cases}$$

(2) Conformational state

When the added module is an end-effector, the smart body reaches the termination state, and the reward returned by the environment is given by the "reachability" function in Eq. The "reachability" evaluation is only for the complete configuration, and for the configuration that does not have an end-effector added to it in the termination state A penalty value of -1 is returned (the maximum number of modules in this paper is set to Nmax=15).

$$r_{terminal} = \begin{cases} -1 & \text{Number of modules}(C') == N_{\max} \text{ and m is not end-effector} \\ reach(C',T) & \text{m is an end-effector} \end{cases}$$

(3)Constraints

Due to the influence of the mechanical structure and joint driving ability and other factors, the joint angle of each joint of the robotic arm is a certain range of constraints, and the constraints of the range of joint angle are defined as:

$$q_{\min} \le q_i \le q_{\max}$$

where: qi is the inverse kinematic solution of the reconfigurable robotic arm; qmin and qmax are the lower and upper bounds of the joint turning angle, respectively.

(4)Optimal Configuration

The ultimate goal of evaluating the robotic arm configuration C is to find the optimal configuration C* that achieves the goal. At the same time, it is desired that the robot has fewer drive modules and a lighter mass. From this, the objective function F is defined by Eq:

$$F(C,T) = reach(C,T) - W_J N_J(C) - W_M M(C)$$

where $N_J(C)$ denotes the number of drive joints in a given robotic arm configuration C, $M(C)$ denotes the total mass of the configuration C, and W_J and W_M are the weighting factors for the number of drive modules and the total mass of the configuration C, respectively, to trade-off between multiple objectives. The optimal configuration C* is defined as given by Eq.

$$C^{\star} = \underset{C}{argmax} \sum_{i=1}^{N_T} F(C,T_i)$$

In this paper, the above metrics and functions are applied to the interaction between the intelligent body and the environment to gradually learn how to maximize the cumulative rewards by choosing the optimal actions. The training interaction process is shown in Figure (Fig.9), in which the DDQN intelligent body selects the next robot module to be assembled based on the list of states and submodule constraints provided by the simulation training environment. The environment immediately adds the module to the current configuration and returns the state, reward signal, termination signal, and list of submodules for the next moment.

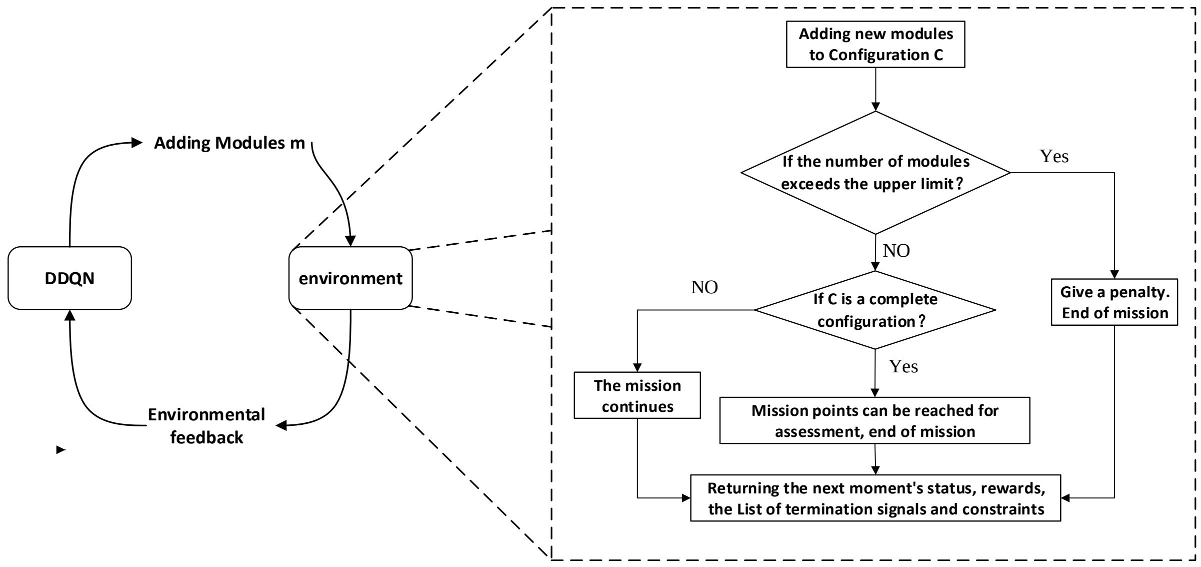
Adding Modules m
DDQN
environment
Environmental feedback
Adding new modules to Configuration C
If the number of modules exceeds the upper limit?
Yes
NO
Give a penalty. End of mission
NO
If C is a complete configuration?
Yes
The mission continues
Mission points can be reached for assessment, end of mission
Returning the next moment's status, rewards, the List of termination signals and constraints

Fig.7 Flowchart of optimal configuration design

3.2 Modified Double DQN with SumTree

The traditional empirical playback mechanism blocks the relevance of the training samples, and the uniform sampling is not conducive to filtering out the quality samples, which is not beneficial to the convergence of the algorithm. In addition, the traditional sampling method needs to sort all the samples, which is very power-consuming. For better sampling, this paper adopts the preferred empirical playback method based on SumTree for solving the problem. SumTree is a special binary tree data structure (Fig.10). There are four layers of node structure in SumTree, the top node is called the root node. The middle two rows are called parent nodes and the bottom row is called leaf node. All empirical samples are stored in the leaf nodes, which also store the priority of the samples.

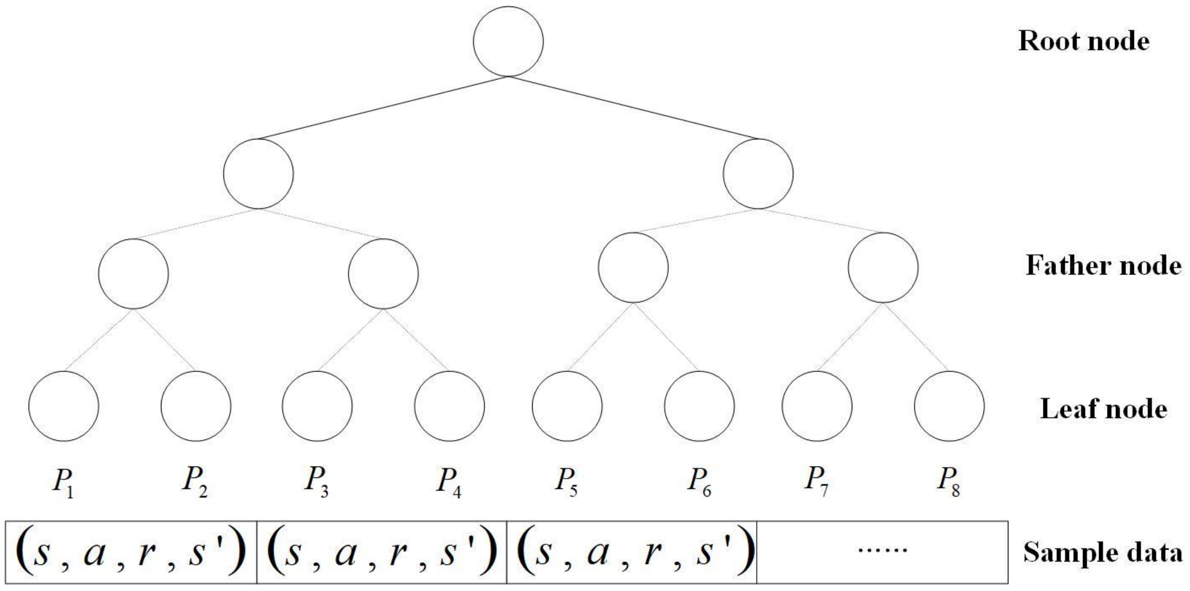

Fig.8 Sumtree structure diagram

SumTree sampling mainly trains the samples based on the priority, which depends on the size of the temporal difference error, the larger the value of the TD error indicates the stronger the back propagation effect of the neural network, and the larger the TD error the higher the priority. The empirical priorities and the probability of their selection are as follows:

$$p_i = |\delta_i| + \epsilon$$

$$p_{(i)} = \frac{p_i^\alpha}{\sum_{i=1}^{k} p_i^\alpha}$$

Where p_i is the priority of the experience, ϵ is the TD error, and is a very small number that prevents the probability from being equal to 0. $p_{(i)}$ is the probability that the sampling experience is selected; α is a hyperparameter that controls the sampling preference in uniform and greedy sampling; And $\alpha \in (0,1)$. When it is 0, sampling is uniform; When it is 1, sampling is greedy.

When sampling with a probability distribution that prioritizes playback, the estimate of the action value function is biased. Since the sampling distribution and the distribution of the action value function are two completely different distributions, to correct for this bias, it is necessary to multiply by a significance sampling coefficient.

$$\omega_i = \left(\frac{1}{N} \cdot \frac{1}{P(i)} \right)^\beta$$

In this paper, the Double DQN algorithm with preferential experience replay is used to solve the multitasking problem in modular robotic arm reconfiguration. Priority Experience Replay (PER) is utilied to reduce the amount of experience required for learning. In the training collection, mini-batches are given priority weights, which helps to reduce the number of iterations in the training process and the training time of the model (Fig.11). In this paper, the effectiveness and performance advantages of the method are verified through subsequent simulation experiments.

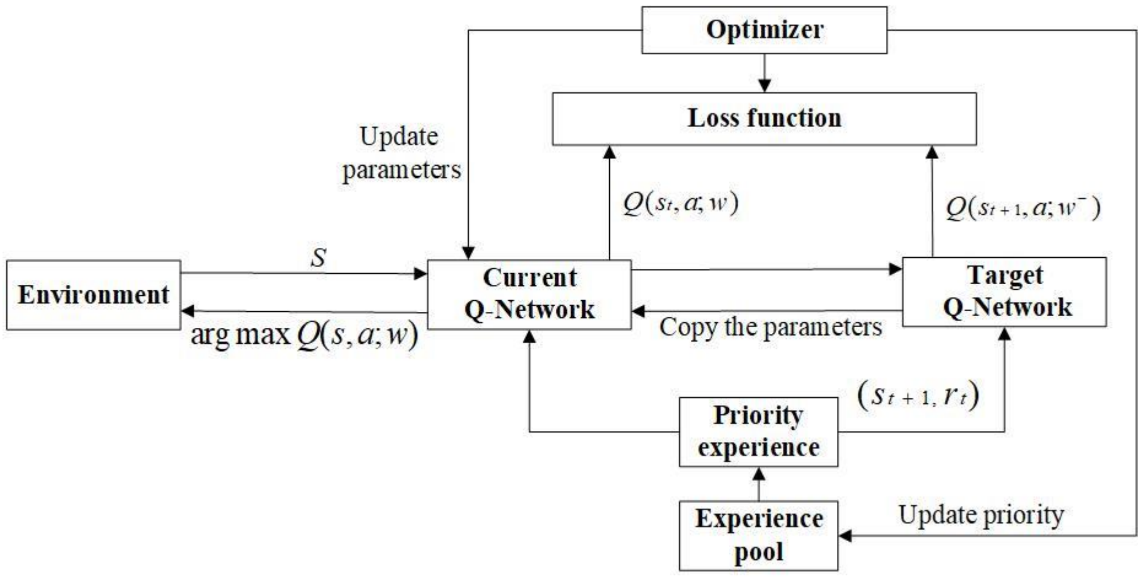

Fig.9 Schematic of the DDQN algorithm with prioritized experience playback

3.3 Modular Robotic Arm Algorithm Based on Improved DDQN Algorithm

In this paper, the Double DQN algorithm with preferred experience playback is applied to modular robotic arm configuration design, and the flow of the algorithm model is shown in Figure (Fig.10) . To obtain the optimal configuration under different tasks, the algorithm takes the state information, constraints, and reward signals of the current configuration as the inputs and is fitted by neural network to get the Q-value corresponding to each module selection. A module a is selected according to the greedy strategy, the new configuration obtained is evaluated and a return is obtained r. The Q value is updated by the return value to decide the module to be selected for the next stage of work, and then the four variables mentioned above are deposited into a quaternion of samples 〈s, a， r， s'〉 , and the multiple samples from the sample pool. A certain batch of samples is taken from the sample pool according to the priority weights to train the neural network and update the network parameters. Then the module selection is carried out again and the cycle is repeated until the termination state is reached. Through this iterative training process, the algorithm can continuously optimize the configuration selection strategy and find the optimal robotic arm configuration under different tasks.

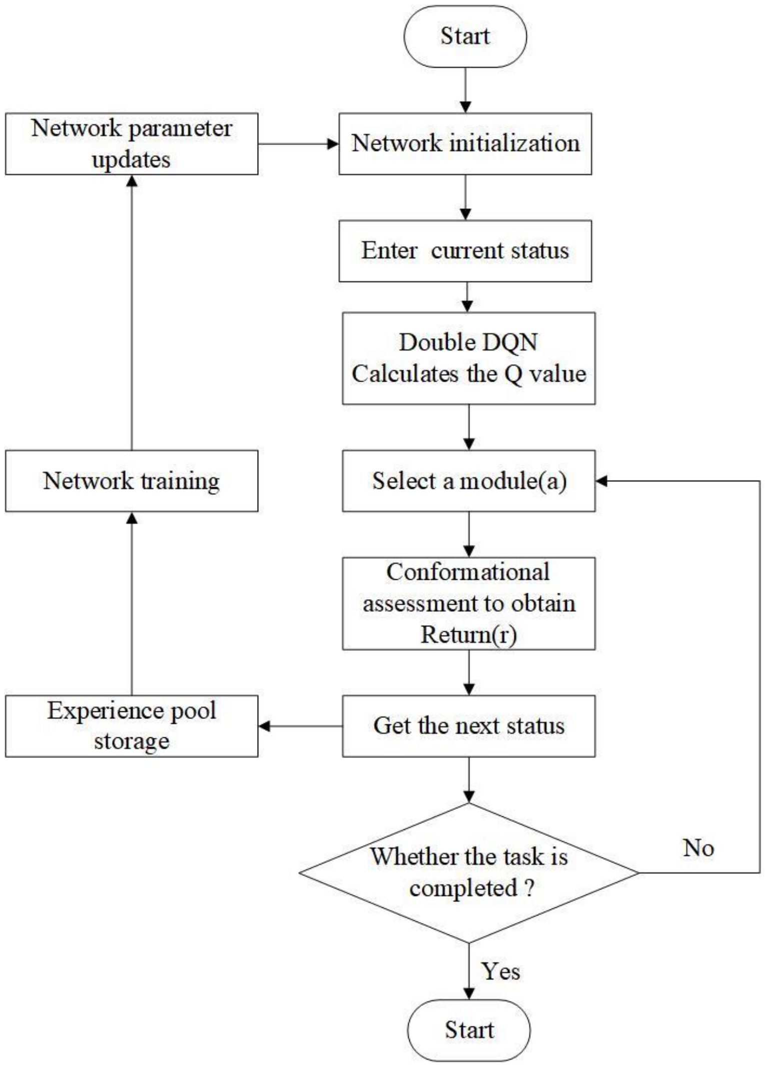
Start
Network parameter updates
Network initialization
Enter current status
Double DQN Calculates the Q value
Network training
Select a module(a)
Conformational assessment to obtain Return(r)
Experience pool storage
Get the next status
Whether the task is completed ?
No
Yes
Start

Fig.10 Flowchart of the application of Double DQN algorithm based on priority experience playback

In this paper, we propose a DDQN modular robotic arm reconfiguration algorithm with prioritized experience playback and using SumTree as the experience playback pool, the pseudo-code of this algorithm is shown in Algorithm 1:

Algorithm1：Double DQN with proportional prioritization

```
1: Input： $s_t$ =( $C_t$ ， $T_t$ ， $O_t$ )
2: Result: optimal configuration $C^*$
3: Initialize replay memory $\mathrm{H}=\varnothing, \Delta=0, p_1=0$
4: Set minibatch $k$ , step size $\eta$ , replay period $K$ and size $N$ , exponents $\alpha, \beta$ , budget T
5: Observe $S_0$ and choose an action $A_0 \sim \pi_w(S_0)$
6: for t=1 to T do
7:     Observe $(S_t, R_t, \gamma_t)$ in H with maximal priority $p_t = \max_{i<t} p_i$
8:     Store transition $(S_{t-1}, R_{t-1}, \gamma_t, S_t, R_t)$
9:     if $t \equiv 0$ mod $K$ then
10:        for i=1 to $k$ do
11:            Sample transition $i \sim P（\mathrm{i}）= p_i^\alpha / \sum_{i=1}^{k} p_i^\alpha$
12:            Compute the importance-sampling weight $\omega_i = (N \cdot P(i))^{-\beta}$
13:            Compute TD-error $\delta_i = R_i + \gamma_i Q_{target}(S_i, \arg\max_a Q(S_i, a)) - Q(S_{i-1}, A_{i-1})$
14:            Update transition priority $p_i \leftarrow |\delta_\mathrm{i}|$
15:            Accumulate weight-change $\Delta \leftarrow \Delta + \omega_i \cdot \delta_i \cdot \nabla_\theta Q(S_{i-1}, A_{i-1})$
16:        end for
17:        Update weights $w \leftarrow w + \eta \cdot \Delta$ , reset $\Delta = 0$
18:        From time to time copy weights into a target network $w^- \leftarrow w$
19:    end if
20:    Choose an action $A_t \sim \pi_w(S_t)$
21: end for
```

4. NUMERICAL EXPERIMENTS

4.1 Simulation Environment

In this paper, a custom simulation environment for robot configuration optimization was created using the OpenAI gym reinforcement learning platform and the Pybullet physics engine. The simulation environment defines the obstacle (left) and the task space (right), the obstacle consists of a cube with a prism length $l_O = 0.25\mathrm{m}$, and the task space is defined as a space $2\mathrm{m} \times 2\mathrm{m} \times 1\mathrm{m}$ containing the possible locations of the obstacle and the target (Fig.).

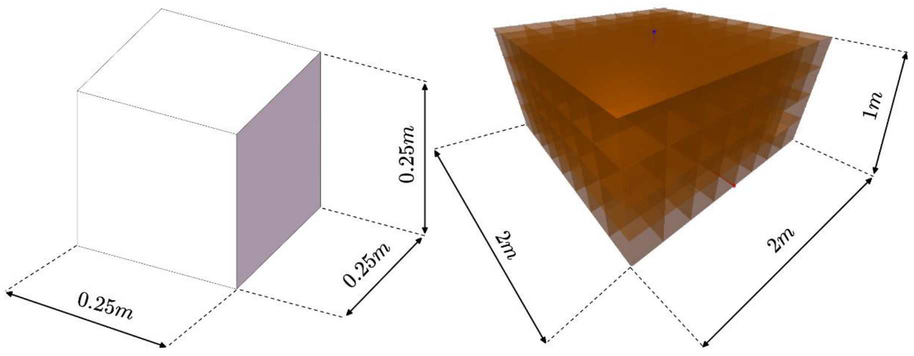

Fig.11 Obstacles and Task Space in Simulated Environments

4.2 Experimental parameter setting

The code is deployed on the Aliyun ECS cloud server for training and the experimental environment is shown in Table (Table 2.)

Table 2. Experimental operating environment

Experimental environment	
Systems	*Ubuntu 20.04*
CPU	*2-core (vCPU)）*
Computer memory	*2 GiB*
Deep learning framework	*Pytorch 1.3.1*
Python	*3.7.15*

In this experiment, the input to the neural network consists of the current configuration C, the workspace T, and the constraints O. These three tensors are concatenated by the Concatenate operation after a layer of fully-connected layers to generate a combined input tensor. The three fully connected layers are activated using ReLU as the activation function, which is applied after the output of each layer, setting negative values to zero and keeping positive values constant. The final output is a state-behavior value for each module Q. The state-behavior value Q is used to guide the decision-making process, helping the intelligence to select the next robot module to be assembled and to maximize the cumulative rewards within the given constraints. The structure of the neural network is described. (Fig.8)

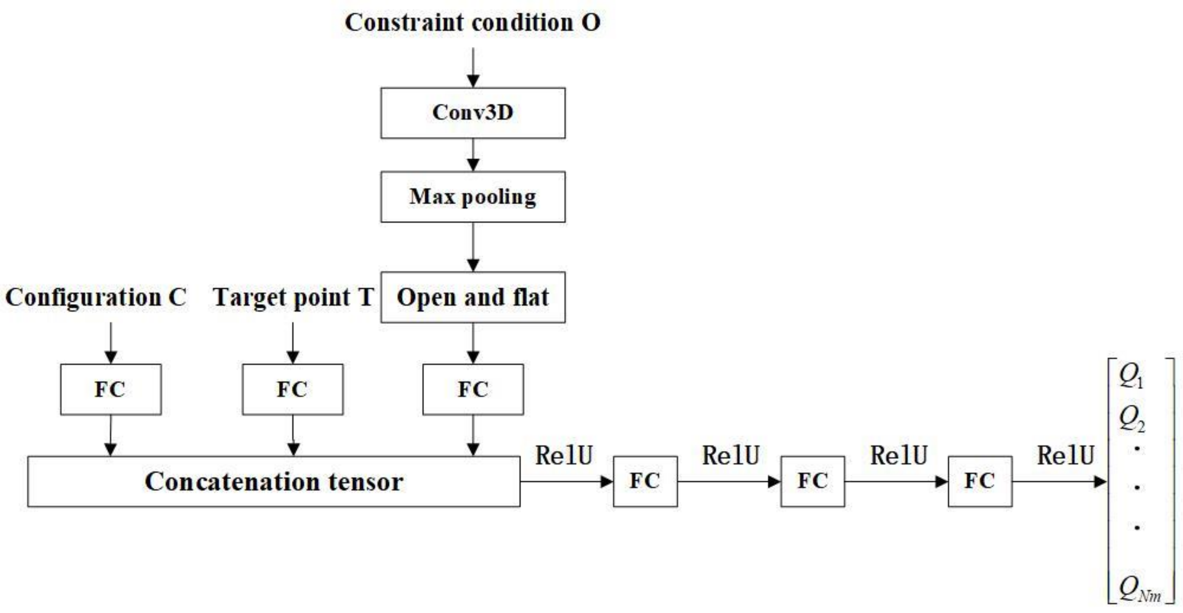

Fig.12 Schematic diagram of neural network structure

To verify the effectiveness of the DDQN modular robotic arm reconstruction algorithm with prioritized experience playback proposed in this paper, the performance of the algorithm combining DDQN and PER was compared with that of the traditional DQN algorithm before training. Both algorithms use the same hyperparameters, which are shown in Table (Table 3.).

Table 3. DDQN algorithm hyperparameters

Parameter	***Reference value***	***Description***
DDQN		
epoch	*100*	*Training rounds*
step_per_epoch	*150*	*The number of actions performed in an epoch*
replay_size	*1000000*	*Experience pool capacity*
gamma	*0.95*	*Discount factor*
epsilon_start	*1*	*epsilon Starting value*
epsilon_decay	*0.0001*	*epsilon Attenuation rate*
epsilon_end	*0.1*	*epsilon Termination value*
q_lr	*0.00025*	*Learning rate*
batch_size	*12*	*Sample size was drawn from the experience pool each time*
start_steps	*120*	*Start updating the network after the first few moves*
max_ep_len	*15*	*The maximum value of an episode*
update_freq	*100*	*Frequency of synchronization of target and main network parameters*

Parameter	***Reference value***	***Description***

Priority experience playback		
α	*0.4*	*Controls the priority queue of experience*
β	*0.6*	*Offsetting the effect of priority experience replay on convergence results*

The parameters that can be adjusted in the simulation environment are shown in the table, where the reference values are also the parameters used in the training environment in this paper:

Table 4. Simulation environment adjustable parameters

Parameter	***Reference value***	***Description***
Visualization	*False*	*Whether the process of module assembly and evaluation is displayed*
RobotPos	$[0,0,0]^T$	*Position of the robot base*
ϵ_P	*0.1*	*End position accuracy error*
ϵ_N	*0.1*	*End attitude accuracy error*
W_J	*0.1*	*The weighting factor for the number of modules in Eq.*
W_M	*0.1*	*Mass weighting factor in the equation*
N_{max}	*15*	*Maximum number of modules that make up the robot*
P_O	*0.005*	*Obstacle generation probability*
E_X	*[-0.5, 0.5]*	*X-axis range of the task space (Unit: m)*
E_Y	*[-0.5, 0.5]*	*Y-axis range of the task space (Unit: m)*
E_Z	*[0, 1]*	*Z-axis range of the task space (Unit: m)*

A training environment was built using the PyBullet simulation engine, and the simulation runtime interface is shown in Figure (Fig.13). In which a multi-degree-of-freedom robotic arm is generated at the origin, yellow dots indicate randomly generated target points, and orange cubes indicate obstacles. (a) Adding the base at the origin. (b)(c)(d) is the process of adding the robotic arm module. (e)(f) is the process of acquiring the target point for the complete configuration.

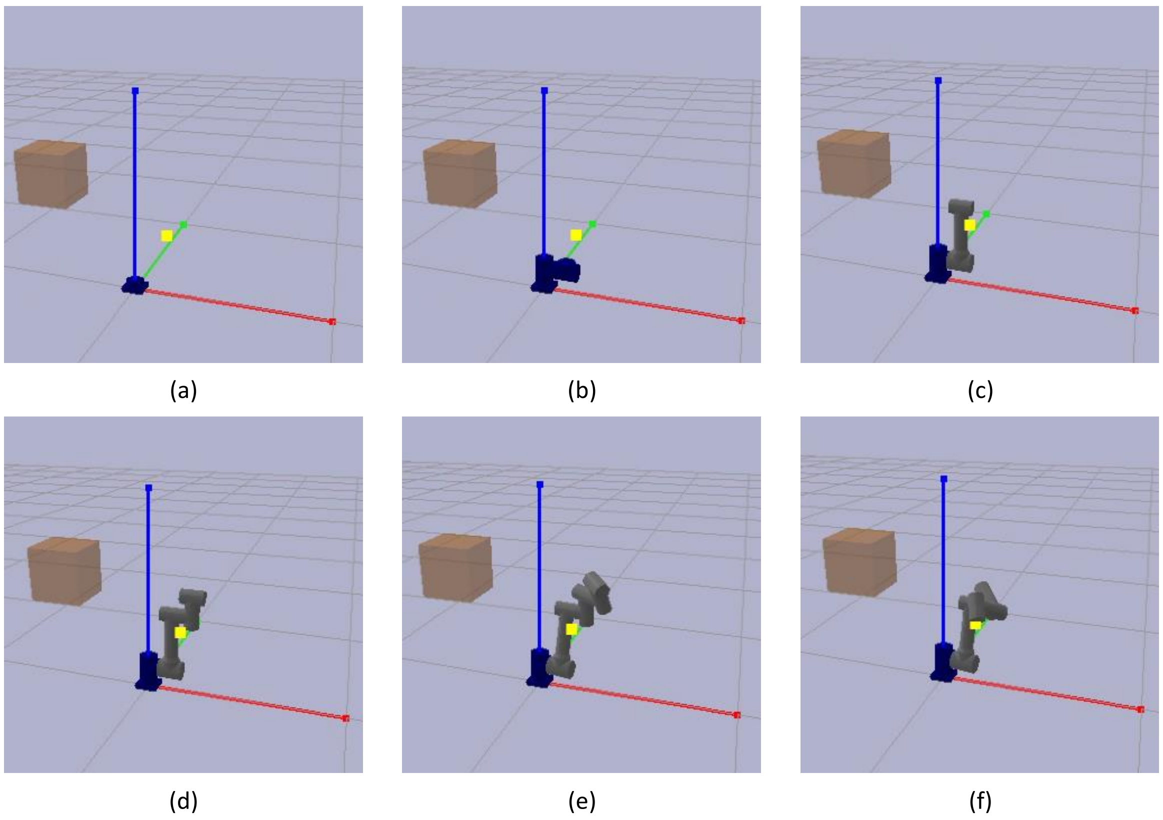

Fig.13 Simulation runtime interface

5. RESULTS AND DISCUSSION

A total of 80 epochs were performed in this experiment, each containing 150 interactions. In the same training environment, we used two different deep reinforcement learning algorithms for training and plotted the variation of the loss function (see Fig. 14). As can be observed from the figure, both algorithms were able to reach a converged state after a certain number of training rounds. The traditional DQN algorithm starts to converge when the number of training steps is close to 1900 steps, but there are large loss fluctuations before that. The improved DDQN algorithm, on the other hand, converges much faster and can reach a stable convergence around 500 steps, and its average loss value remains below 2.5. Under the same training environment, the DDQN algorithm demonstrates faster convergence speed and better stability than the traditional DQN algorithm, and its average loss value is also smaller. This indicates that the improved algorithm learns and optimizes the Q-value function more efficiently during the training process, which enhances the learning ability and performance of the agent.

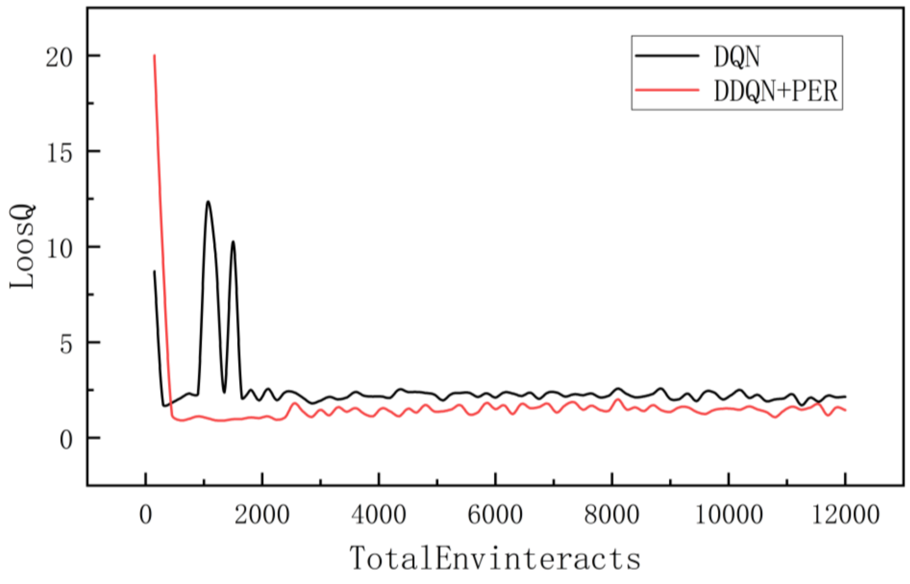

Fig.14 Loss Function Comparison

The maximum Q value of each selection in both algorithms is recorded as shown in Fig. (Fig.15), and it can be seen that the maximum Q values of both algorithms have the same trend and converge to near 12 after about 8000 steps of iteration. However, the average value of the maximum Q value of the DQN algorithm after convergence is 11.689, and the average value of the maximum Q value of the improved DDQN algorithm is 11.302, which indicates that the DDQN algorithm successfully mitigates some of the over-estimation when solving the problem of too high Q value and inaccurate target Q value in network training, and the algorithm is able to more accurately estimate and update the Q value function during the training process. Thereby improving the performance and stability of the algorithm.

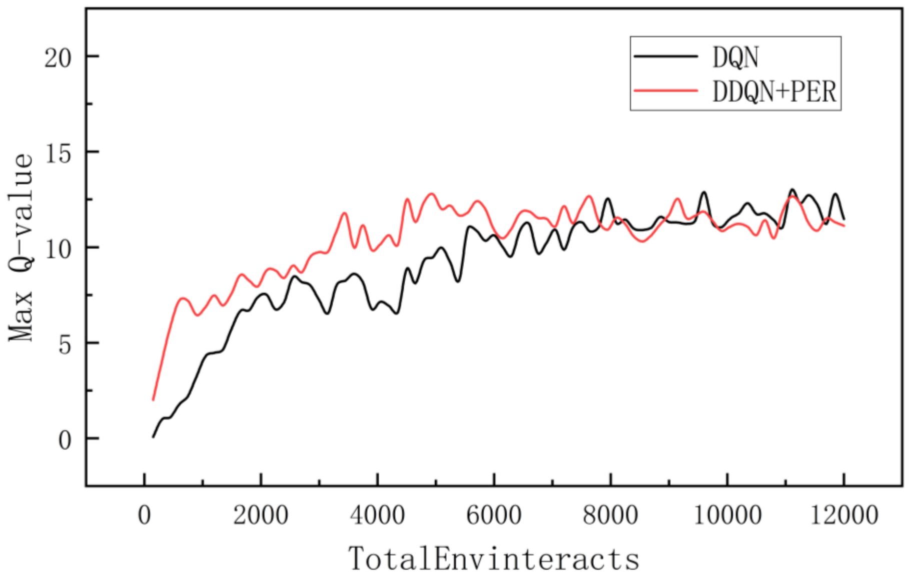

Fig.15 Maximum Q Comparison

The ultimate goal of both deep reinforcement learning algorithms is to maximize the target reward, therefore, comparing the average reward values obtained by the two algorithms, as shown in Figure (Fig.16), it can be observed that both curves show an increasing trend and stabilize after more than 30 training rounds. The average reward value of the DDQN algorithm converges near about 9 and the average reward sum over 80 training rounds is 485.026. while the average reward value of the DQN algorithm converges near about 8 and the average reward sum over 80 training rounds is 378.511. In addition, the DDQN algorithm produced higher average reward values than the DQN algorithm in the vast majority of cases. This suggests that the DDQN algorithm learned a better strategy and was able to select more efficient actions in the environment, thus increasing the overall reward of the agent. This also validates the effectiveness of the improved DDQN algorithm and further demonstrates its better learning ability and adaptability relative to the traditional DQN algorithm.

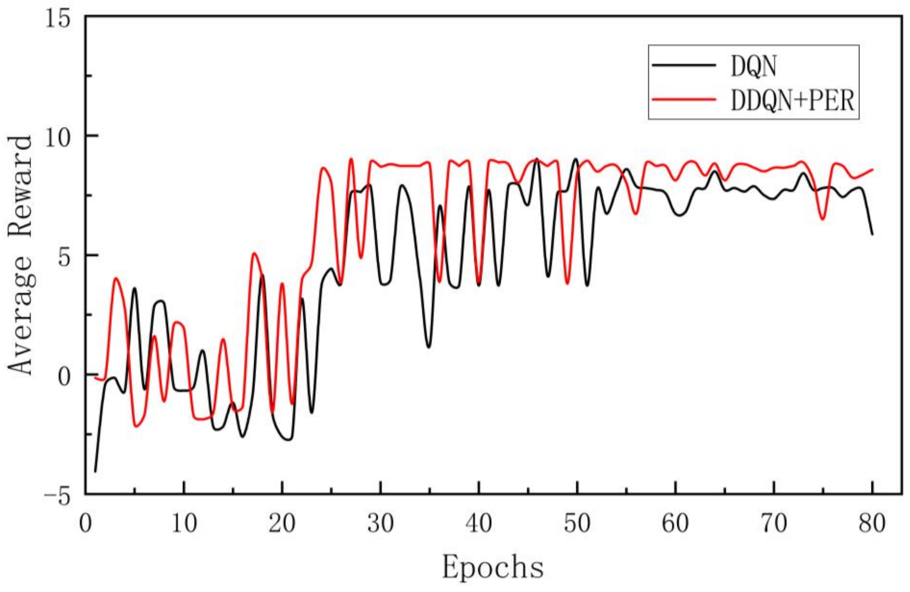

Fig.16 Comparison of average reward values

The study aims to verify the effectiveness of the proposed algorithm on the modular reconfiguration of robotic arms and to evaluate the performance of the new algorithm in solving the modular reconfiguration problem of robotic arms. During the study, a total of 50 groups of randomized tests were generated, with each group of targets containing a grid of up to five random obstacles. In each group, a target point was randomly generated through a normal distribution, and the configuration that successfully reached the target point was defined as the feasible configuration for the following data comparison:

(1) The probability of successfully finding a feasible configuration;

(2) The average number of modules of a feasible configuration;

(3) Average module weight of feasible configurations;

(4) Average time to find a feasible configuration;

The results of the comparative experiment are shown in Table (Table 5.):

Table 5. Comparison Test Results

Algorithm	*The probability of finding a configuration*	*The average module mass*	*The average number of modules*	*The average time (Unit: seconds)*

Traditional DQN algorithms	*24%*	*7.23*	*7.5*	*7.807*
DDQN + PER algorithms	*32%*	*5.92*	*5.75*	*7.984*

The traditional DQN algorithm succeeded in finding complete configurations in 12 out of 50 sets of tests, while the proposed algorithm found 16 sets of complete configurations. Comparing the average module mass of complete configurations, it can be obtained that the average mass of complete configurations obtained by the proposed algorithm in this paper is about 5.92kg, and the average mass of complete configurations obtained by the traditional DQN algorithm is about 7.23kg, and the method in this paper decreases the average mass of robotic arm modules by about 18.11%. Comparing the average number of modules for the complete configuration, the algorithm proposed in this paper has 1.75 fewer average modules compared to the traditional DQN algorithm. Comparing the average time to find a feasible configuration, the two algorithms are comparable, A comparison of the data associated with the two algorithms to find feasible configurations is shown in (Fig.17).

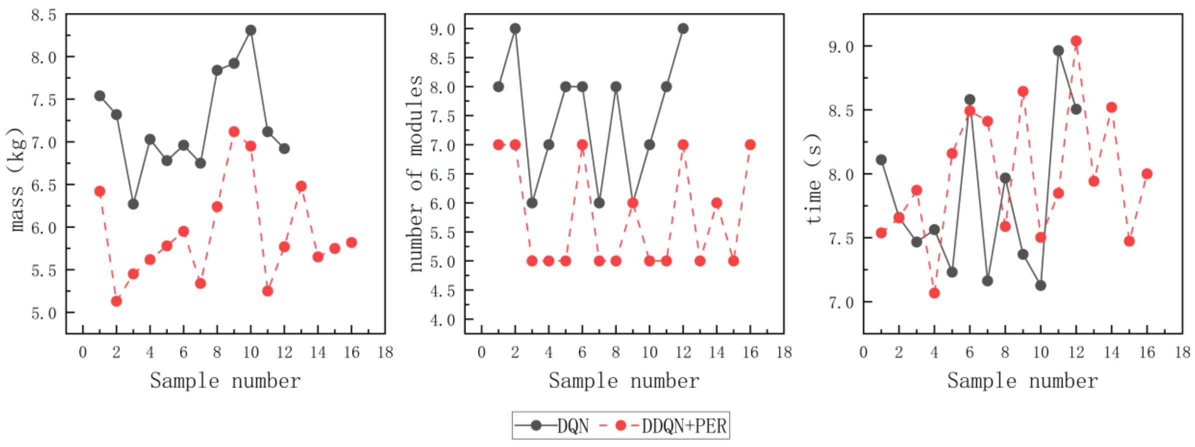

Fig.17 Comparison of data related to the two algorithms to find feasible configurations

From the results of the above comparative tests, we can conclude that compared to the traditional DQN algorithm, the algorithm proposed in this paper has a higher probability of successfully finding feasible configurations, and the feasible configuration modules are lighter in mass and fewer in number. This means that the algorithm in this paper is able to find more feasible configurations, and these configurations have lighter and more compact structures, and this advantage can bring higher efficiency and performance in practical applications.

6. CONCLUSION

In this paper, a DDQN modular robotic arm reconfiguration algorithm with prioritized experience playback is proposed, aiming to resolve the conflict that exists between diversity and economy in robotic arm design. The algorithm is improved by making improvements in terms of task customisation and robot economics, and comparative experiments are conducted with the traditional DQN algorithm. The results show that the algorithm can reduce the risk of overestimation of the Q-value and improves the convergence performance of the algorithm by prioritizing the empirical playback mechanism. This enables the modular robotic arm to achieve a higher success rate in the reconfiguration process while also improving the quality of the optimal configuration to adapt to complex and changing production environments and task requirements.

Future work could consider applying the algorithm to real robotic arm modules through migration learning and other methods to achieve modular robotic arm reconstruction in real scenarios. This will further validate the feasibility and effectiveness of the algorithm and explore its potential and limitations in practical applications.

REFERENCES

1. Wei J. Research on chained reconfigurable robotic arm for space on-orbit operation[D]. Harbin Institute of Technology, 2020.
2. [2]Tu Yuanyuan,Wang Dayi,Zhang Xiangyan et al. Reconfigurability and autonomous reconfiguration methods for spacecraft[J/OL]. Journal of Aeronautics,pp:1-14,2023.
3. [3] Luo H, Li M, Liang G, et al. An obstacle-crossing strategy based on the fast self-reconfiguration for modular sphere robots[C]//2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). IEEE, 2020: 3296-3303.
4. [4] Whitman J, Bhirangi R, Travers M, et al. Modular Robot Design Synthesis with Deep Reinforcement Learning[J/OL]. Proceedings of the AAAI Conference on Artificial Intelligence, 2020, 34(06): 10418-10425.
5. [5] Schaul T, Quan J, Antonoglou I, et al. Prioritized experience replay[J]. arXiv preprint arXiv:1511.05952, 2015.
6. [6] Zhai P, Zhang Y, Shaobo W. Intelligent Ship Collision Avoidance Algorithm Based on DDQN with Prioritized Experience Replay under COLREGs[J]. Journal of Marine Science and Engineering, 2022, 10(5): 585.
7. [7] Liu S, Li X, Huang Yun. Research on NPC traveling route planning based on improved DQN algorithm[J]. Radio Engineering,vol:52(08),pp:1441-1446,2022.
8. [8] Liu J, Wang YH, Wang L, Yin ZZ. A non-cooperative multiuser dynamic power control method based on SumTree sampling combined with Double DQN[J/OL]. Telecommunications Technology,pp:1-9,2023.
9. [9] Liu J, Zhang X, Hao G. Survey on research and development of reconfigurable modular
10. [10]Feder M, Giusti A, Vidoni R. An approach for automatic generation of the URDF file of modular robots from modules designed using SolidWorks[J]. Procedia Computer Science, 2022, 200: 858-864.
11. [11] Van Hasselt H, Guez A, Silver D. Deep reinforcement learning with double q-learning[C]//Proceedings of the AAAI conference on artificial intelligence. 2016, 30(1).