Hole-Peg Assembly Strategy Based on Deep Reinforcement Learning

Xiaodong Zhang[1](✉) and Pengchao Ding[2]

[1] Beijing Key Laboratory of Intelligent Space Robotic Systems Technology and Application, Beijing Institute of Spacecraft System Engineering CAST, Beijing 100086, China
[2] Institute of Technology, Harbin 150001, Heilongjiang, China

Abstract. Hole-peg assembly using robot is widely used to validate the abilities of autonomous assembly task. Currently, the autonomous assembly is mainly depended on the high precision of position, force measurement and the compliant control method. The assembly process is complicated and the ability in unknown situations is relatively low. In this paper, a kind of assembly strategy based on deep reinforcement learning is proposed using the TD3 reinforcement learning algorithm based on DDPG and an adaptive annealing guide is added into the exploration process which greatly accelerates the convergence rate of deep reinforcement learning. The assembly task can be finished by the intelligent agent based on the measurement information of force-moment and the pose. In this paper, the training and verification of assembly verification is realized on the V-rep simulation platform and the UR5 manipulator.

Keywords: Deep reinforcement learning · Assembly · TD3

1 Introduction

At present, an increasing number of robots are used in a variety of assembly tasks. There are mainly two strategies for a robot to finish the assembly task, One is to preview the assembly task by teaching the robot to fulfil the task and optimize the teaching path, but it is difficult to perform the high precision assembly tasks directly; The second is to design an impedance control method for assembly strategy and make adjustments to find the best parameters, however it often can't get the good performance in unknown situations.

In this paper, deep reinforcement learning strategy is designed which can accomplish the better adaptive abilities in unknown situations. A deep reinforcement learning algorithm is adopted based on the assumption that the environmental state information of the next moment can be inferred through the current environmental state information and the actions adopted, the algorithm simulates the human learning process through designing a reward mechanism, and then finding the optimal strategy to get the reward by constant trial and error. In recent related researches, this algorithm has made some progress in intelligent grasping [1–4] and intelligent tracking of robots [5–8]. However,

F. Sun et al. (Eds.): ICCSIP 2020, CCIS 1397, pp. 15–32, 2021.
https://doi.org/10.1007/978-981-16-2336-3_2

these scenes have low requirements on the motion accuracy of the robot, which cannot directly prove its applicability in the field of high-precision assembly.

Recently, many researches have been proposed on assembly [9–15]. J. Xu [10] proposed a new strategy based on traditional force control to reduce training time and used fuzzy reward mechanism to avoid the network optimization of algorithm entering local optimal, but the traditional force control actually limits the performance of the policy to some extent. Inoue. T [13] introduced a kind of LSTM neural network structure to improve the performance of network in tasks. Luo Jianlan [15] uses iLQG reinforcement learning algorithm to combine force position hybrid control and space force control, and then uses MGPS method to optimize the parameters of network, but this approach does not significantly improve the autonomous intelligence of the robot. Fan Yongxiang [9] combines the advantages of GPS and DDPG [16] to design a new framework in which the supervised learning provides guidance trajectory and the reinforcement learning carries out strategy optimization. Mel Vecerik [11] reduces the amount of work required to elaborate the reward in the reinforcement learning algorithm by adding human demonstrations. There are some problems in DDPG algorithm include the slow convergence speed and large amount of super parameter adjustment. Many scholars [9–11, 13, 14, 17] have adopted DQN algorithm, however this algorithm usually need to discretize relevant data which cannot deal with the continuity problem of large data volume.

In order to prove the feasibility of deep reinforcement learning in assembly task, this paper adopts the hole-peg assembly task, and this assembly is also the most basic assembly form of industry assembly task. At the same time, the neural network structure of TD3 algorithm [18] is improved to increase the robustness of the algorithm, a certain angle deviation is added between the axis and the hole, and the precision of the sensor in the simulation is reduced.

The rest of this article is organized as follows: The second section mainly elaborates and analyzes the tasks in detail. The third section describes the methods used in detail. The fourth section is a detailed analysis of deep reinforcement learning algorithm in V-rep training and testing process. The fifth section is the main conclusions of this paper and some prospects for the future work.

2 The Description of Problems

In the high-precision hole-peg assembly task, the corresponding assembly process is divided into the hole-searching stage and the hole-inserting stage. The assembly strategies and difficulties of these two parts are quite different, so the spatial hierarchical strategy is adopted to decompose the assembly task, the first one is the deep reinforcement learning strategy in the hole-searching stage, which is to complete the movement from any position out of the chamfering to the actual position of the hole. The second is the deep reinforcement learning strategy during inserting the hole which is to complete all the subsequent tasks. These two tasks are distinguished by determining the z position of the axis. Therefore, the researches and training in this paper are mainly carried out around these two parts. Firstly, it is the hole-searching stage. In this stage, the angle offset is so small that it is often ignored, as a result of which, the main movement dimension of the hole is three translational dimensions while the rotation action dimension is corrected by the internal position control strategy. Then it is the hole-inserting phase when the position and angle observations are so small that they usually need to be amplified. In this case, the action dimension needs not only the translational dimension but also the rotational dimension.

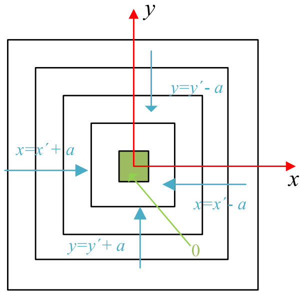


Fig. 1. The position error of the end of the shaft relative to the hole

Due to the influence of visual positioning error, the position and posture of the end of the shaft relative to the hole are measured, which is obtained by combining the forward kinematics calculation of the manipulator with the visual positioning to reduce the whole value to the origin so as to improve the ability of deep reinforcement learning to resist parameter perturbation. As shown in the Fig. 1, in the x direction, when $-a < x < a$, the value of p_x is zero; when $a < x$, the value of p_x is $x - a$; when $a > x$, the value of p_x is $x + a$. In the y direction, when $-a < y < a$, the value of p_y is zero; when $a < y$, the value of p_y is $x - a$; when $a > y$, the value of p_y is $y + a$. By this treatment, the interference error of the arm in forward operation can be reduced.

3 A Detailed Description of the Methods

In this section, two strategies of hole-searching and hole-inserting are elaborated and then the TD3 algorithm is introduced. Assume that when the current state is known, the future state is independent of the past state among the assemblies in this assembly task. So translate the solution to the assembly problem into a "Markov decision process", where the decision part is determined by the agent and the assembly process can be represented as a sequence where states and actions alternate chronologically. In this case, each action is expected to work toward the ultimate goal—maximized long-term reward, for which the value of each action contributing to the ultimate goal should be quantified. As a result, make the agent continuously interact with the environment to explore the value of each action, Q. Based on this value, an optimal strategy π is found, which determines the best action a according to the current state s to obtain the maximum reward.

$$\pi(s) = \mathrm{argmax}_a Q(s, a)$$

At the same time, in order to alleviate the sparsity problem in the exploration of reinforcement learning, the assembly process is divided into the hole-researching and hole-inserting phases, and different treatments were made for different stages.

3.1 Hole-Searching Phase

The environmental state information needed to observe is:

$$S = [p_x, p_y, p_z, \alpha, \beta, \gamma, F_x, F_y, F_z, T_x, T_y, T_z] \tag{1}$$

Where p_x, p_y, p_z are the positions relative to the hole after treatment; α, β, γ are the angles relative to the normal line of the hole after treatment; F_x, F_y, F_z, T_x, T_y are the force information detected by the six-dimensional force-moment sensor which can indirectly reflect the contact situation between the axle and hole; x, y, z are three axes in space. However, angle information is controlled by the internal strategy regulator in the hole-searching stage, for which our specific operation is to delete the three dimensions of the state information observed by the agent γ and T_z basically have no practical effect on this assembly task, so which can be deleted. Reducing dimensions can not only reduce computation and simplify neural network model, but also effectively alleviate the problem of dimension explosion. At this time the state S changes into:

$$S = [p_x, p_y, p_z, F_x, F_y, F_z] \tag{2}$$

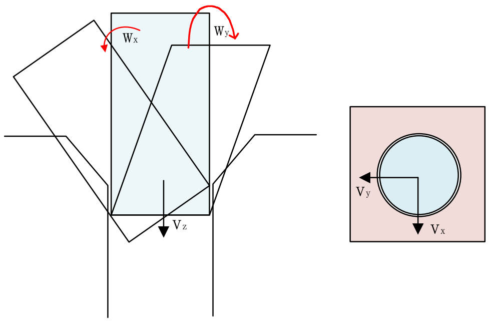


Fig. 2. The motion directions of peg

As shown in the Fig. 2, there are mainly five directions of motion in the assembly of circular hole shaft, which are v_x,v_y, v_z, w_x and w_y. The action of the agent in this stage are mainly 3-dimensional translational motions:

$$a_0 = [v_x, v_y, v_z] \tag{3}$$

Where v_x, v_y, v_z are the translational velocities respectively in the x, y and z axes, which are the three dimensions of the output of actor network. When this action is transmitted to the controller, it is denoted as:

$$a_0 = [v_x, v_y, v_z, 0, 0, 0] \tag{4}$$

The knowledge-driven action at this stage is just a simple proportional controller algorithm:

$$a_1 = \left[v_x, v_y, v_z, w_x, w_y, 0\right]$$

$$\begin{aligned} v_x &= k_{x1}p_x + k_{x2}F_x \\ v_y &= k_{y1}p_y + k_{y2}F_y \\ v_z &= b_1 \\ w_x &= k_{x3}\alpha + k_{x4}T_x \\ w_y &= k_{y3}\beta + k_{y4}T_y \end{aligned} \tag{5}$$

The reinforcement learning is optimized and evaluated based on reward. At this stage, the reward designed is divided into two parts: one is the reward after successful assembly, the other is the penalty value of failure to complete the task.

The punishment at this stage is divided into two parts, force punishment and positional punishment, where the force penalty p_f is the two norm of F and the position penalty p_{pos} is the two norm of pos:

$$penalize = p_f K + p_f(1 - K)\ 0 < K < 1$$

$$p_f = f(F), F = \| F_x\ F_y\ F_z \|$$

$$p_{pos} = f(pos), pos = \| p_x\ p_y\ p_z \|$$

$$f(x) = \begin{cases} -x/xmax & xmax > x > 0 \\ -1 & x > xmax \end{cases} \tag{6}$$

Where the function $f(x)$ is a linear limiting function with respect to x and $xmax$ is the maximum value of x. By adjusting the parameter K, not only ensure that penalize is between (0,1], but also adjust the proportion of each part in penalize. The reward value after successful assembly is just related to the value of the steps learned to complete the assembly action and the maximum speed of assembly is achieved by setting the reward value:

$$reward = 1 - \frac{n}{nmax}\ n \in (0, nmax] \tag{7}$$

Where n is the value of step in each episode, whose maximum value is nmax.

When the assembly of this segment is completed, the reward is the value of *reward*. In addition, the penalize is used to ensure that the value of the reward is within (−1,1).

3.2 Hole-Inserting Phase

At this stage, the value of pose information is relatively small, but the deviation is large, so the state information of environment observed by agent is changed to the power information:

$$S = [p_z, F_x, F_y, F_z, T_x, T_y] \tag{8}$$

At the same time, the actions of agents are in 5 dimensions:

$$a_0 = \left[v_x, v_y, v_z, w_x, w_y\right] \tag{9}$$

Where v_x, v_y, v_z are the translational velocities respectively in the x, y and z axes; w_x, w_y are the rotation speed along the x and y axis respectively. In this stage, the knowledge-driven action is as follows:

$$\begin{aligned} a_1 &= \left[v_x, v_y, v_z, w_x, w_y, 0\right] \\ v_x &= k_{x5}F_x \\ v_y &= k_{y5}F_y \\ v_z &= b_2 \\ w_x &= k_{x6}T_x \\ w_y &= k_{y6}T_y \end{aligned} \tag{10}$$

Then let knowledge-driven action provide supplementary guidance to agent exploration and the reward of this phase is also divided into two parts:

$$penalize = \tfrac{p_f + p_M}{2}K + p_h(1-K)\ 0 < K < 1$$

$$p_f = f(F), F = \| F_x\ F_y\ F_z \|$$

$$p_M = g(M), M = \| M_x\ M_y\ M_z \|$$

$$p_h = f_1(h), h = |p_z|$$

$$g(x) = \begin{cases} -\sqrt[3]{x/xmax} & xmax > x > 0 \\ -1 & x > xmax \end{cases}$$

$$f_1(x) = \begin{cases} -1.2x/xmax + 0.6 & xmax > x > 0 \\ -0.6 & x > xmax \end{cases}$$

$$reward = 1 - \tfrac{k}{kmax}\ k \in (0, kmax] \tag{11}$$

When the assembly of the segment is completed, the reward is the value of the reward, otherwise the penalize is adopted to ensure that the value of the reward is within (−1,1).

In order to get the long-term reward of every action in the current state, the reinforcement learning algorithm is designed as follows:

$$R_i = r_i + \gamma r_{i+1} + \gamma^2 r_{i+2} + \cdots + \gamma^{T-i} r_T = r_i + \gamma R_{i+1} \tag{12}$$

Where r_i is the reward value at the current moment; r_T is the reward at time T; γ is the decay rate of the effect of future reward value on current reward; R_i is the current long-term returns; R_{i+1} is the long-term return at the next moment. Calculate the value of long-term returns by iterating. However, the long-term returns bring about the rapid accumulation of fitting errors and variances when using neural network fitting. For this reason, long-term returns at the expense of longer-term returns:

$$R_i = \frac{r_i + \gamma R_{i+1}}{1 + \gamma} \tag{13}$$

Since the current size of the reward is within the range of −1 to 1, the value of the long-term reward can be effectively limited to the range of −1 to 1 by designing such a long-term reward. Compared with original formula, it can effectively reduce the influence caused by the accumulation of deviation, especially the fitting deviation for the undetected space.

In order to solve the problem that there are so many data in the continuous space of high latitude, DDPG reinforcement algorithm uses a neural network to fit the action-value function $Q(s, a)$, which means the value of taking action a under state s. In order to make the critic network fit $Q(s, a|\theta^Q)$ well, the optimized loss function of network is set to:

$$L\left(\theta^Q\right) = E_{\pi'}\left[\left(Q\left(s_i, a_i|\theta^Q\right) - y_i\right)^2\right] \tag{14}$$

$$y_i = r(s_i, a_i) + \gamma Q'\left(s_{i+1}, \mu'\left(s_{i+1}|\theta^{\mu'}\right)|\theta^{Q'}\right) \tag{15}$$

Where $r(s_i, a_i)$ is the reward of taking action a_i under states_i; $\mu'\left(s_{i+1}|\theta^{\mu'}\right)$ is the action a_{i+1} obtained by using actor neural network to fit the strategy function under the states_{i+1}; $Q'\left(s_{i+1}, a_{i+1}|\theta^{Q'}\right)$ is a neural network whose structure is identical to $Q(s, a|\theta^Q)$, which is used to calculate the value of taking action a_{i+1} in states_{i+1}; γ is the attenuation value of the influence of future value. Similar to the long-term return, get the value in the case of (s_i, a_i) throughy_i. Then by continuous training, a better actor-critic function $Q(s, a)$ can be fitted.

As mentioned above, there are four neural networks involved in the DDPG structure: the critic network, the target critic network, the actor network and the target actor network. The Actor network is the control strategy network whose main function is to calculate the corresponding actions according to the information of the sensor. The structures of the two target networks are completely similar to the corresponding network structure, but there is a certain delay for them on parameter update compared with the corresponding network. The update strategy adopted is the moving average of the corresponding network parameters:

$$\theta = \tau\theta' + 1 - \tau\theta \tag{16}$$

The main functions of the Target network are to calculate the value and to attain the action of the next moment in the time series. This dual network method can effectively alleviate the coupling problem of time. The update mode of actor network parameter θ^μ is the gradient direction of maximum value and the calculation method adopts the chain rule of gradient:

$$\begin{aligned} \nabla_{\theta^\mu} J &\approx \mathbb{E}_\pi[\nabla_{\theta^\mu} Q^\mu(s_t, \mu\left(s_t|\theta^\mu\right)|\theta^\mu) \\ &= \mathbb{E}_\pi[\nabla_{a_t^\mu} Q^\mu(s_t, a_t)\nabla_{\theta^\mu}\left(\mu\left(s_t|\theta^\mu\right)\right) \end{aligned} \tag{17}$$

The TD3 algorithm used in this paper is improved on the basis of DDPG algorithm and the overall framework is shown in Fig. 3. There are three main improved strategies. As for the first strategy, change the critic network to a pair of independent Q-value neural

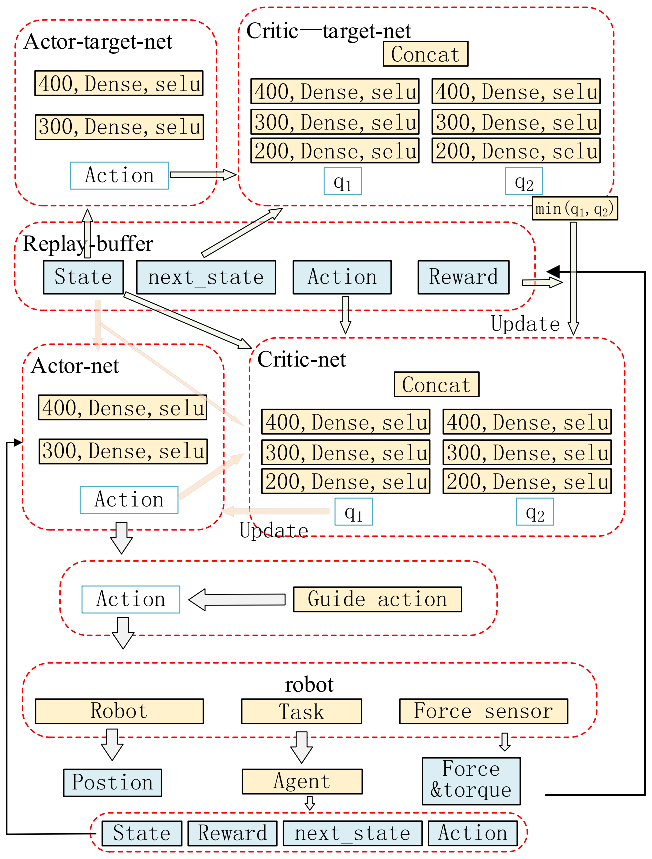


Fig. 3. Frame diagram of TD3 deep reinforcement learning strategy

networks and when optimize the network parameters, the smaller one of the two Q values is used to calculate the loss value, so as to avoid overestimation of errors:

$$y = r_i + (1 - Done) * \gamma * \min_{j=1,2} Q^{'}\left(s_{i+1}, a_{i+1}|\theta_j^Q\right) \tag{18}$$

On the second one, the main cause of the error is the deviation of the estimation of the value function and regularization parameters are often used to eliminate deviations in machine learning. Therefore, a small area around the target action in the action space should be smoothed, which means adding some noise to the action when calculating the Q value of the target action:

$$\mu^{'}\left(s_{i+1}|\theta^{\mu^{'}}\right) + \epsilon$$

$$\epsilon \sim clip\left(\mathcal{N}\left(0, \tilde{\sigma}\right), -c, c\right) \tag{19}$$

About the third, delay the update of actor network, which reduce the cumulative error caused by multiple updates while reducing unnecessary repeated updates, thus solving the time coupling problems of value function and policy function.

The algorithm is divided into two threads, the first action thread is to explore the environment and storing the experience, the second learning thread is to train the network with the recovered experience. The Action thread refers to the part in bottom left of Fig. 3, Firstly, reset the assembly's initial environment including setting the initial deflection Angle within ±2°, setting the position deviation within ±1 mm during the training jack phase and getting the feedback from the environment during the initial exploration. Secondly, select the directing actions or agent output actions with certain probability through the action selector, which is obtained through the actor network in the agent in the top of Fig. 3. Then the robot is asked to perform that action and interact with the environment to obtain new status information, which will be processed by the agent to obtain the reward value and then immediately store them in replay buffer. Finally, start a new cycle of above-mentioned state - action - state process. Replay buffer uses the FIFO method to store information and improve queue structure by taking out a block of memory in advance to explicitly store the information and the corresponding address, which speeds up the transfer of data between CPU and GPU at the expense of memory cache. At the same time, its reading method is adopted randomly to eliminate the correlation among the data and prevent overfitting.

Table 1. The learning thread of algorithm1

```
Algorithm 1 action thread
Initialize replay buffer: R
for k=1 to M do
    Initialize the noise distribution N
    $S_1$ = env.reset()
    when the data stored in R reaches Ns
        a start signal is sent to the learning thread
    for t = 1 to T do
        At a certain probability $\epsilon$ choice guiding action $a_g$
    or $a_t = \mu(s_t|\theta^\mu) + N_t$
    $s_{t+1}, reward_t, done, \_ = env.step(a_t)$
        let R store $s_t, s_{t+1}, a_t, reward_t, done$
        t = t+1
    end for
    k = k+1
    when k is a multiple of 100
        test the current actor network performance
end for
Sends a termination signal to the learning thread
```

The learning thread refers to the upper part of the Table 1. For updating the critic network, agent continuously replays the experience from replay buffer and obtains a smaller current Q value from the target network. After a few steps of delay, the actor network is updated by using the gradient of the critic network, and the specific content is shown in Table 2.

Table 2. The learning thread of algorithm 2

```
Algorithm 2 learning thread
Initialize actor network $\mu(s|\theta^{\mu})$
Initialize critic network $Q(s,a|\theta_1^Q)$, $Q(s,a|\theta_2^Q)$
Initializes the target network:$Q' \leftarrow Q,\ \mu' \leftarrow \mu$
Repeat
  for i=1 to I do:
    for t =1 to T do:
      randomly obtain the minibatch data $(s_i, s_{i+1}, a_i, r_i)$ from R and the size of mini-batch is R$_{batch}$
      $\epsilon \sim clip(\mathcal{N}(0,\tilde{\sigma}), -c, c)$
      $a_{i+1} = \mu'(s_{i+1}|\theta^{\mu'}) + \epsilon$
      $y' = r_i + (1 - Done) * \gamma * \min_{j=1,2} Q'(s_{i+1}, a_{i+1}|\theta_j^Q)$
      $y = \frac{y'}{1 + (1 - Done) * \gamma}$
      $\theta_i \leftarrow \text{argmin}_{\theta_i} \frac{1}{N}\sum_i (y - Q(s,a|\theta_i^Q))^2$
    end for
    update actor network:
    $\nabla_{\theta^{\mu}} J \simeq N^{-1} \sum \nabla_a Q(s,a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^{\mu}} \mu(s|\theta^{\mu})|_{s_i}$
    update target network:
    $\theta_i^{Q'} \leftarrow \tau\theta_i^Q + (1-\tau)\theta_i^{Q'}$
    $\theta^{\mu'} \leftarrow \tau\theta^{\mu} + (1-\tau)\theta^{\mu'}$
  end for
  wait until the star signal from the action thread is received
until receive the termination signal sent by the action thread
```

4 Simulative Training

This section mainly introduces the specific situation of our simulation training. The main robot for high precision assembly is the UR5 six-axis manipulator. The main simulation experiment platform used in the training is v-rep. The main accessory information of the computer used in the training is the GPU of NVIDIA 1660Ti with the video memory frequency of 12000 MHz and video memory of 6G and the CPU with main frequency of the 3.7 GHz AMD 2700×.

The first step is to import the axle hole assembly model and UR5 manipulator into the V-rep, as shown in Fig. 4. The diameter of the hole is 25.0 mm and the depth is 36.0 mm. The chamfering part has a maximum diameter of 40 mm and a depth of 2 mm, and the diameter of the shaft during training is 24.95 mm and its length is 10 mm.

During the training, the 2.78 version of bullet engine is adopted for the dynamic engine and select the high-precision mode. Besides, the simulation time step of the training was set to 5.0 ms and the maximum number of steps in each episode was set to 20 in the hole-seeking stage and 450 in the hole-inserting stage. The maximum Replay buffer storage is set to 1 million and the random sample size is 1024.

In order to simplify the calculation, the influence of gravity is ignored, which means setting the gravity constant G of the simulation environment to 0 and adding the gravity compensation to the dynamics control of the manipulator in the subsequent experiments. At the same time, the controller uses the internal velocity Jacobian matrix to convert the velocity of the end workspace into the velocity of each joint in the joint space in real time.

To ensure the safety of the assembly, the maximum contact force in z axis direction is set as $F_{zmax} = 50\,\text{N}$. At the same time, in order to ensure that the simulation process is as close as possible to the real situation, a threshold value is set for the absolute deviation of the Angle and position of the axis in the inserting stage.

When it is greater than the threshold, namely, $abs(\theta) \leq 4^{\circ}$ and $abs(\Delta p) \leq 2\,\text{mm}$, the inserting process is stopped and restart a new process. At the same time, during the training process, the motion range of the axle is restricted: in the searching stage, the ranges of x and y direction are set as (−9,9) mm and that of z-direction is (27.0, 40.5) mm; in the inserting phase, the range of z-direction is (0,2 8.5) mm. When the assembly process is restarted due to an uncompleted task, the reward value is set to −1.

Before formal training, in order to reduce the error accumulation caused by the large deviation and variance of the network in the initial training, the data set of initial instruction action is designed and the data set of initial reward to preliminarily train the actor network and the critic network.

To do this, according to the random generation of uniform distribution, generate a set of states within a certain range and include a state set which contains as many states as possible that may be involved in the experiment, and then obtain the action set through the directive action generator whose input is the random state set. Therefore, the elements in the two sets can be saved into state-action data sets one by one in order to train the actor network by the state-action data sets. The obtaining way of the initial reward data set is: according to the uniform distribution, in a certain range, randomly generate a data set which includes as much state action information as possible that may be involved in the experiment. At the same time, simplify the system of reward designed in the assembly process, as a result of which, the reward is fixed as 0.7 when *done*, and the reward set when restarting the assembly process is not taken into account. Then based on that, the reward data set is built and the critic network is initially trained with this data set. However, when training the critic network, set the reward decay rate to zero, which means that we only care about the immediate reward rather than the long-term return.

As for the probability ϵ, which is selected to guide action, we adopt intelligent adaptive annealing method, namely, to make the corresponding changes according the observation of the test data. Specifically, when the completion degree is greater than N for three consecutive times, the probability ϵ becomes half of the original value and N = N + 1. The initial value of ϵ is 0.5 and the initial value of N is 5. Specific algorithm is shown in Table 3.

Table 3. Test threat of Algorithm 3

```
Algorithm 3 test thread
Initialize $N_{done} = 0, R_{sum} = 0, N_{step} = 0, N = 5, N_1 = 0$
for k=1 to 10 do
  S_1 = env.reset()
  for t = 1 to T do
  $a_t = \mu(s_t|\theta^\mu)$
  $s_{t+1}, reward_t, done, \_ = env.step(a_t)$
  $R_{sum} = R_{sum} + reward_t$
    If the assembly process ends, $N_{step} = N_{step} + t$;
      If the assembly task is completed, $N_{done} = N_{done} + 1$
    t = t+1
  end for and calculate the sum of the rewards
  k = k+1
end for
if $N_{done} > N : N_1 = N_1 + 1$
  if $N_1 > 3,\ \epsilon = \epsilon/2,\ N = N + 1, N_1 = 0$
```

4.1 Inserting Phase

In this part, a comparative experiment on the improvement of long-term returns is made. The long-term returns are changed only and the other improvements remain unchanged. The main content of the comparison includes the completion rate, the average reward value of each step and the number of steps to complete the task.

From the Fig. 4(a), the degree of completion increased steadily and finally reached 100%. From the Fig. 4(b), the average reward value of each step shows a steady increase in volatility. From the Fig. 4(c), the number of steps required to complete the task steadily decline until the training is finally stopped, and the number of steps required reduce by half compared with the early training and by three quarters compared with the pure guiding actions, resulting in two times and three times higher efficiency respectively.

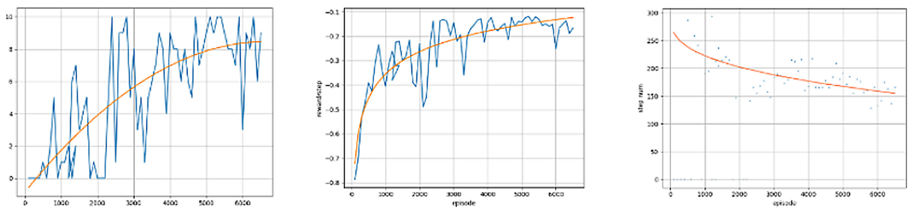

(a) success rate (b) r/step (c) step number

Fig. 4. Data graph after changing long-term returns

From Fig. 5, with long-term returns unchanged, the first completion case appears at the training step of 2000 and then rapidly increases to 100%, the number of steps to complete the task has reached a relatively small value since the beginning of the completion of the task and has declined in volatility. The comparison shows that the algorithm can learn the successful strategy more steadily and faster after changing the long-term return.

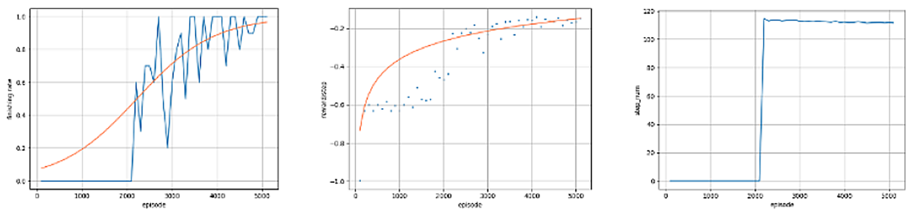
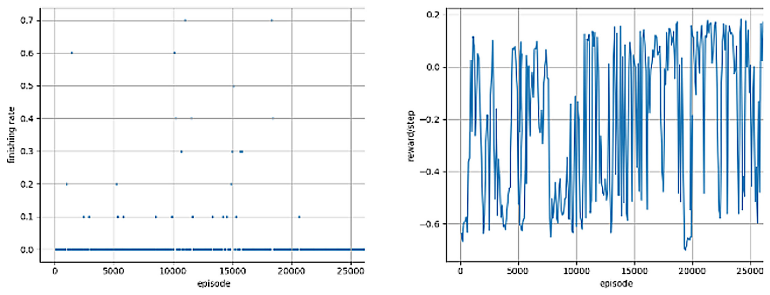

(a) success rate (b) r/step (c) step number

Fig. 5. Data graph without changing long-term returns

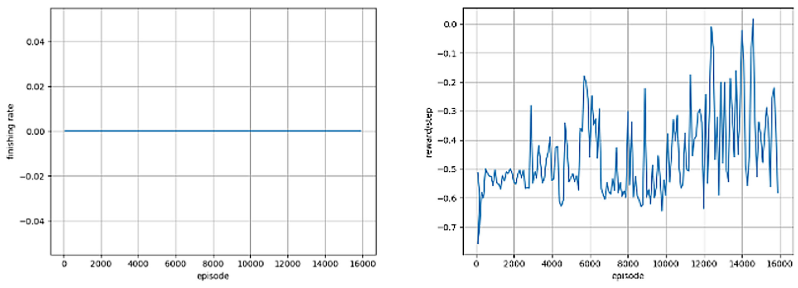
In order to verify the effect of the intelligent annealing method, a comparative experiment was made with the undirected action. The data for comparison are involved in two aspects including completion rate and average reward of each action step, as shown in Fig. 6. It can be seen from the figure that there are only a few completion cases in the 25,000 training steps while the average reward value of each action step fluctuates slowly. Compared with the previous experiments, it can be concluded that the effect of intelligent annealing method is very significant.

Then, in order to verify the effect of network pre-training, experiments without pre-training and the data are simulated and the cooperation are completed about the rate and the average reward of each action step as shown in Fig. 7. It can be seen from the figure that there are no case of completion in the training steps within 16,000, while the reward values of each action step slowly increase with some fluctuates. Compared with the previous experiments, it can be concluded that network pre-training has a very significant effect.



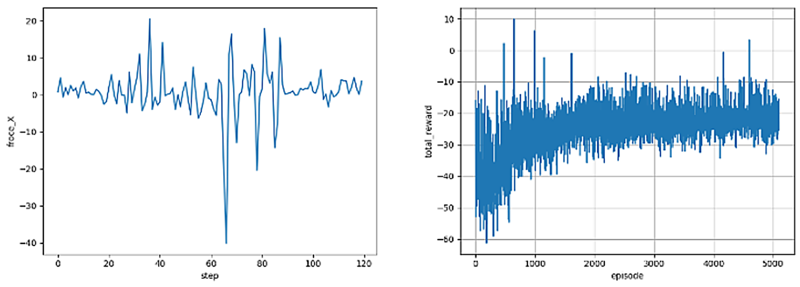
(a) success rate (b) r/step

Fig. 6. Data changes without guidance to explore



(a) success rate (b) r/step

Fig. 7. Data changes without network pretraining



(a) force (b) reward

Fig. 8. Some other related data graphs

From the data information corresponding to each moment in assembly activities, as shown in Fig. 8(a), it can be found that the contact force is maintained in a relatively small range to ensure the rapid completion of the task. In Fig. 8(b), it shows that the change of total reward value in the training process whose reward value declines first and then rises rapidly in the volatility.

4.2 Performance of Anti-Jamming

This part mainly studies the anti-interference ability of strategies derived from reinforcement learning. This paper mainly studies the change of the measured contact force after adding the interference signal to the control signal or the acquired sensor signal. The interference signals tested are common Gaussian interference signals and mean interference signals. In the absence of interference signal, the change of contact force in x, y and Z direction is shown in Fig. 9.

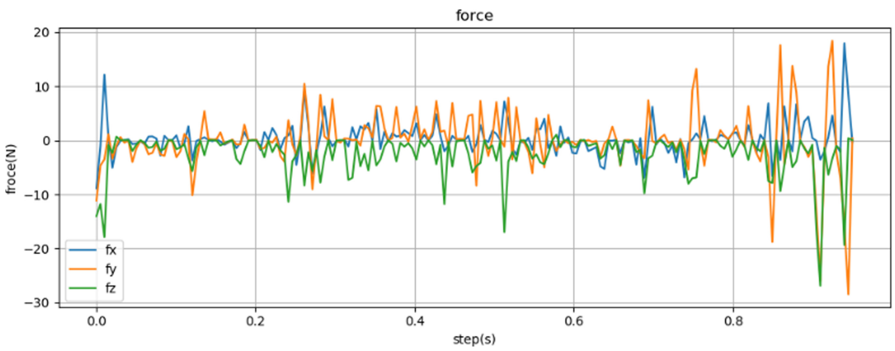
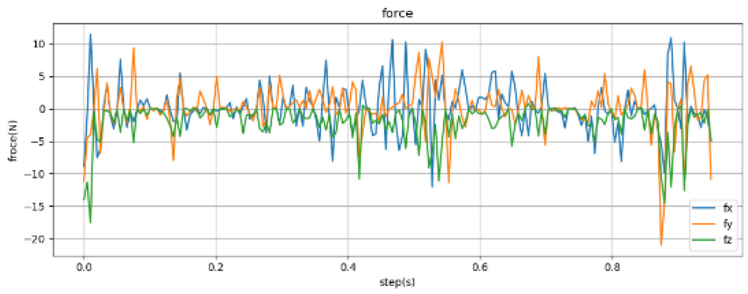
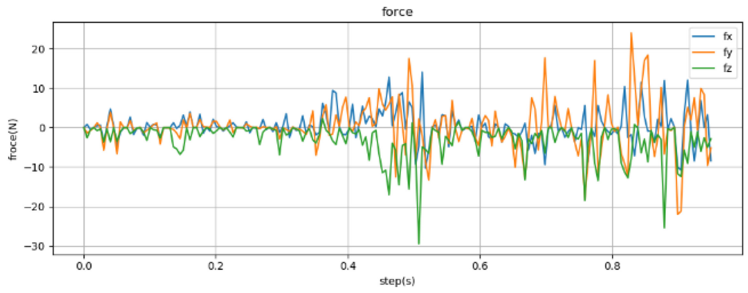


Fig. 9. The contact forces in the x,y, and Z directions

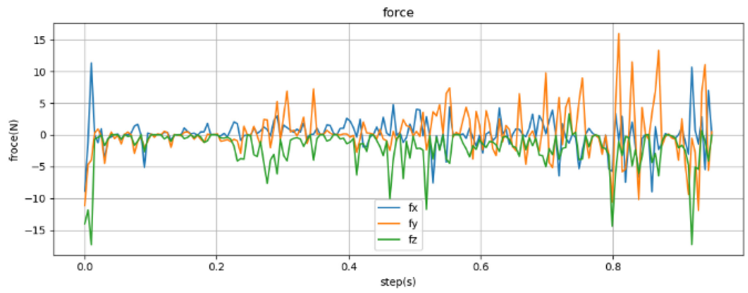
When interference signal is added to the control signal, the change of contact force in the direction of X, Y and Z is shown in Fig. 10. By comparing the Fig. 9 and Fig. 10(a), it is clear that the amplitude of contact force variation under Gaussian noise increases to a certain extent globally but the maximum amplitude does not change significantly, which means the contact force is still within the acceptable range although the fluctuation increases, so it proves that the strategies has the capability to resists Gaussian interference signal which is added to controller. By comparing Fig. 9 and Fig. 10(b–d), it can be concluded that the strategies has the capability to resists the Gaussian or mean interference signal which are added to controller or force sensor.



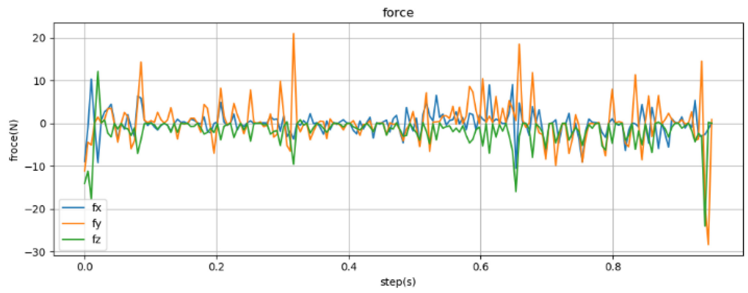
(a) Add Gaussian noise to the control signal



(b) Add mean noise to control signal



(c) Add Gaussian noise to the force sensor



(d) Add a constant value bias to the force sensor

Fig. 10. The change of contact force

5 Conclusions

This paper presents an assembly method based on TD3 deep reinforcement learning algorithm, and designs an adaptive annealing guide method in the exploration process, which greatly accelerates the convergence rate of deep reinforcement learning. In addition, this paper improves the form of traditional reward function to reduce the possibility of divergence of the deep reinforcement learning algorithm, and the effect of the improved algorithm in accelerating convergence is proved by several comparison experiments. Finally, the interference signals is added to prove that the network trained by deep reinforcement learning algorithm and validate the abilities of anti-interference signals in the simulation.

A future direction is to introduce LSTM networks to the deep reinforcement learning to perceive time-ordered data for further improving the control effect which is similar to the performance of differential and integral terms in traditional control theory. The other direction is to study the generalization ability of reinforcement learning for multiple assemblies with different assembly shapes or more complex assembly process.

References

1. Quillen, D., Jang, E., Nachum, O., et al.: Deep reinforcement learning for vision-based robotic grasping: a simulated comparative evaluation of off-policy methods (2018). arXiv Robot
2. Zeng, A., Song, S., Welker, S., et al.: Learning synergies between pushing and grasping with self-supervised deep reinforcement learning. Intell. Robots Syst. 4238–4245 (2018)
3. Gu, S., Holly, E., Lillicrap, T., et al.: Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates. In: International Conference on Robotics and Automation, pp. 3389–3396 (2017)
4. De Andres, M.O., Ardakani, M.M., Robertsson, A., et al.: Reinforcement learning for 4-finger-gripper manipulation. In: International Conference on Robotics and Automation, pp. 1–6 (2018)
5. Yang, P., Huang, J.: TrackDQN: visual tracking via deep reinforcement learning. In: 2019 IEEE 1st International Conference on Civil Aviation Safety and Information Technology (ICCASIT). IEEE (2019)
6. Kenzo, LT., Francisco, L., Javier, R.D.S.: Visual navigation for biped humanoid robots using deep reinforcement learning. IEEE Robot. Autom. Lett. **3**, 1–1 (2018)
7. Yang, Y., Bevan, M.A., Li, B.: Efficient navigation of colloidal robots in an unknown environment via deep reinforcement learning. Adv. Intell. Syst. (2019)
8. Zeng, J., Ju, R., Qin, L., et al.: Navigation in unknown dynamic environments based on deep reinforcement learning. Sensors **19**(18), 3837 (2019)
9. Fan, Y., Luo, J., Tomizuka, M., et al.: A learning framework for high precision industrial assembly. In: International Conference on Robotics and Automation, pp. 811–817 (2019)
10. Xu, J., Hou, Z., Wang, W., et al.: Feedback deep deterministic policy gradient with fuzzy reward for robotic multiple peg-in-hole assembly tasks. IEEE Trans. Ind. Informat. **15**(3), 1658–1667 (2019)
11. Roveda, L., Pallucca, G., Pedrocchi, N., et al.: Iterative learning procedure with reinforcement for high-accuracy force tracking in robotized tasks. IEEE Trans Ind. Informat. **14**(4), 1753–1763 (2018)
12. Chang, W., Andini, D.P., Pham, V., et al.: An implementation of reinforcement learning in assembly path planning based on 3D point clouds. In: International Automatic Control Conference (2018)

13. Inoue, T., De Magistris, G., Munawar, A. et al.: Deep reinforcement learning for high precision assembly tasks. In: Intelligent Robots and Systems, pp. 819–825 (2017)
14. Vecerik, M., Hester, T., Scholz, J., et al.: leveraging demonstrations for deep reinforcement learning on robotics problems with sparse rewards (2017). arXiv: Artificial Intelligence
15. Luo, J., Solowjow, E., Wen, C., et al.: Reinforcement learning on variable impedance controller for high-precision robotic assembly. In: International Conference on Robotics and Automation, pp. 3080–3087 (2019)
16. Lillicrap, T., Hunt, J.J., Pritzel, A., et al.: Continuous control with deep reinforcement learning (2015). arXiv Learning
17. Mnih, V., Kavukcuoglu, K., Silver, D., et al.: Playing Atari with deep reinforcement learning (2013). arXiv Learning
18. Fujimoto, S., Van Hoof, H., Meger, D. et al.: Addressing function approximation error in actor-critic methods (2018) . arXiv: Artificial Intelligence