Proceedings of the 2018 IEEE International
Conference on Intelligence and Safety for Robotics
Shenyang, China, August 24-27, 2018

Deep Reinforcement Learning Algorithm for Object Placement Tasks with Manipulator

Wei Guo[1], Chao Wang[1], Yu Fu[1], Fusheng Zha[1]*

***Abstract*—To settle the problem that the household robot needs to have the flexibility of object types and target poses when performing object placement tasks, a deep reinforcement learning algorithm is utilized in this paper. By using this algorithm robot can learn the placement method for different object types autonomously under the policy search part, and a convolutional neural network (CNN) policy is trained to make the robot adapt to different target poses. When performing these tasks, the placement error can also be reduced by modifying the sampling method and the weight form of cost function. Finally, the learning ability and flexibility of the deep reinforcement learning algorithm is tested by simulation.**

I. Introduction

Although object placement is one of the most basic housework, it is very difficult for robots to perform this task indoors. First, because the robot needs to deal with unpredictable and various kinds of objects, it is unrealistic to make corresponding programming for each object. Therefore, the robot must have the ability to learn by themselves [1]. At the same time, because the target pose may vary from each other, the robot also needs to have a strong adaptability to the placement tasks [2].

Current conventional manipulator control method, such as position control, compliance control [3], and machine vision based control methods [4], cannot meet the requirements of self-learning capability for the household robot. Although reinforcement learning algorithm can make manipulator learn and accomplish the specified task through the interaction with environment, such as balancing the inverted pendulum [5], collaborating with people and accomplish the assembly task [6], etc. However, when the target pose changes, reinforcement learning algorithm must learn again to get a new controller, so the adaptability of the algorithm is poor [7].

With the development of deep learning in the last decade and the proposal of the deep reinforcement learning algorithm [8], manipulators have been able to learn the prescribed task autonomously and have shown the flexibility of target position [1], object type [9] and space obstacle [10] under the control of deep reinforcement learning algorithm. When performing object placement tasks, the completion result can be good under force-controlled manipulators, however at present the placement error still needs to be reduced when the agent becomes position-controlled manipulators under deep reinforcement learning algorithm.

Based on the analysis above, we use UR10 (a 6-DOF position-controlled manipulator) as the agent of our deep reinforcement learning algorithm and investigate object placement tasks in household environment. By modifying the action and using the improved cost function and sampling method, the placement error can be reduced under this position-controlled manipulator. The main contents of this paper include: (1) Research and improvement methods of the deep reinforcement learning algorithm based on guided policy search [11]; (2) Research on the CNN policy of the UR10 agent; (3) Simulations of the deep reinforcement learning algorithm.

II. Deep Reinforcement Learning Algorithm

In this section, a deep reinforcement learning algorithm is built based on guided policy search and the action is modified according to the position-controlled feature of UR10. At the same time, performance of the algorithm is improved by modifying the cost function and sampling method.

A. Brief Introduction of the Modified Guided Policy Search Algorithm

In reinforcement learning algorithm, the learning process of object placement tasks can be regarded as optimizing policy parameters θ so that the expected cost of the policy is reduced:

$$\min_{\theta} \; E_{\pi_\theta}\left[\sum_{t=1}^{T} c(s_t, a_t)\right] \tag{1}$$

where $c(s_t, a_t)$ is the cost under state s_t and action a_t at time step t and T is the length of an episode.

By adding a constraint term $p(a_t \mid s_t) = \pi_\theta(a_t \mid s_t)$ to the original problem and using BADMM algorithm [12], guided policy search breaks down the original problem into a policy search process and a supervised learning problem. In the policy search process, first a Gaussian mixture model is fitted and used as the priori to fit a local environment dynamic model:

$$p(s_{t+1} \mid s_t, a_t) = N(f_{sat}[s_t, a_t] + f_{ct}, F_t) \tag{2}$$

where f_{sat} and f_{ct} is the coefficient and F_t is the covariance matrix.

Then by using LQG algorithm [13], the Q-function and value function are alternately calculated from the last time step of a trajectory to obtain the controller parameters:

Research supported by National Natural Science Foundation of China (No.U1713222, No.61773139 and No.61473015), the Natural Science Foundation of Heilongjiang Province, China(Grant No.F2015008) and Shenzhen Peacock Plan (No.KQTD2016112515134654).

1 State Key Laboratory of Robotics and System, Harbin Institute of Technology, Harbin, China.

* Corresponding author, e-mail: zhafusheng@hit.edu.cn

$$\begin{aligned} Q_{sa,sat} &= \tilde{c}_{sa,sat} + f_{sat}^T V_{s,st+1} f_{sat} \\ Q_{sat} &= \tilde{c}_{sat} + f_{sat}^T V_{st+1} + f_{sat}^T V_{s,st+1} f_{ct} \\ V_{s,st} &= Q_{s,st} - Q_{a,st}^T Q_{a,at}^{-1} Q_{a,st} \\ V_{st} &= Q_{st} - Q_{a,st}^T Q_{a,at}^{-1} Q_{at} \end{aligned} \tag{3}$$

where the subscripts denote the gradient and Hessian matrix for the state or action vector.

And the controller of manipulator can be obtained as follows:

$$p(a_t \mid s_t) = N\left(K_t s_t + k_t; Q_{a,at}^{-1}\right) \tag{4}$$

where the coefficients in the average term are:

$$K_t = -Q_{a,at}^{-1} Q_{a,st}, \quad k_t = -Q_{a,at}^{-1} Q_{at} \tag{5}$$

Finally, the CNN policy is optimized under the supervision of current controller using stochastic gradient descent. The details of guided policy search can be found in reference [11].

According to the position-controlled feature of UR10, action of the controller must be modified. Otherwise, the manipulator will acquire excessively discrete states when sampling, making policy search algorithm divergent. The modified action of controller is:

$$a_t = N\left(K_t s_t + k_t; Q_{a,at}^{-1}\right) + \theta_t \tag{6}$$

where θ_t denotes the joint angle vector of manipulator at time step *t*.

By adding the joint angles of manipulator at current time step, the modified controller converts absolute joint values into relative joint values, so that joint states of manipulator at adjacent moment will not change violently and the sampling region of manipulator is limited to the vicinity of trajectories from last optimization, making the policy search algorithm converge eventually.

B. *the Improved Cost Function of Object Placement Tasks*

For the object placement task with manipulators, the cost function of policy search is defined as follows:

$$c(s_t, a_t) = \alpha_1 d_t^2 + \alpha_2 \log\left(\rho + d_t^2\right) \tag{7}$$

where d_t is the distance between the object and target position, α_1, α_2 and ρ are hyperparameters. By simulation under UR10 agent, these hyperparameters are finally taken as α_1=10, α_2=0.1, ρ=10^{-5}.

The cost function here consists of two parts, when the end effector is far from the target position, the first term will make it approach target position quickly; and when the end effector is near target position, the logarithm term will force the controller to produce more precise actions and improve the precision of the final result.

Based on equation (7), this algorithm gives weight ω_t to the cost obtained at each sampling time and segments the cost weights as follows:

$$\omega_t = \begin{cases} \dfrac{1}{2T} t & t \le \dfrac{T}{2} \\ \dfrac{3}{2T} t - \dfrac{1}{2} & \dfrac{T}{2} < t \le T \end{cases} \tag{8}$$

where the value of T is taken as 20 for UR10.

For the trajectory obtained by policy search algorithm, the importance of actions in the initial stage is different compared to the end stage. At the end of a trajectory, actions performed by manipulator will directly determine the completion result of a task. Therefore, giving higher weights to the cost of these time steps can make the optimization process of controller better and improve the final performance of manipulator.

At the same time, since the action at the last time step will directly determine the completion result of a task, this algorithm also gives an independent weight $\tilde{\omega}_T$ to the cost at the last time step, and the value of $\tilde{\omega}_T$ is taken as 10 through simulation. After adding the weights above, the sum of cost in a trajectory becomes:

$$c(\tau) = \sum_{t=1}^{T} \omega_t c(s_t, a_t) + \tilde{\omega}_T c(s_T, a_T) \tag{9}$$

where τ is the trajectory.

C. *the Improved Epsilon-greedy Sampling Method*

In this part the sampling method of guided policy search is improved. By using ε-greedy method, the algorithm can make full use of the CNN policy and achieves better convergence when performing object placement tasks.

In each condition of the object placement task, the original sampling method is:

Run controller $p(a_t \mid s_t)$ N times.

The modified sampling method becomes:

1) Run controller $p(a_t \mid s_t)$ once (no action noise);

2) Run CNN policy $\pi_\theta(a_t \mid o_t)$ once (no action noise);

3) Calculate the cost of the two trajectories obtained above, and choose the sampling method corresponding to lower cost as the main method of ε-greedy;

4) Remove the trajectory with higher cost;

5) Run ε-greedy method N-2 times (with action noise).

In each condition, manipulator first samples once under current controller and CNN policy, because the algorithm needs to calculate the cost of these two trajectories to select the main sampling method of ε-greedy, so noise cannot be added during these two sampling processes. After calculating the cost and choosing the main sampling method, the trajectory with higher cost is then removed from samples. Finally, the algorithm will run the new ε-greedy method and samples N-2 times. Action noise needs to be added this time to explore the state space around current trajectories. In these two sampling methods above, N is the number of samples taken in each condition, where N=5 is taken.

In ε-greedy method, the value of hyperparameter ε needs to be set carefully. Since the purpose of exploration in the vicinity of current optimal trajectory is achieved by adding action noise when using ε-greedy method. Therefore, the value of ε does not need to be set very low, where ε=0.95 is taken in this paper.

III. Convolutional Neural Network Policy

In this section, we use controllers optimized by policy search to train a CNN policy, making manipulator adapt to the target pose of object placement tasks under this policy.

A. Convolutional Neural Network Policy Structure

When constructing the CNN policy, first we need to determine the input form of the network. For the supervised controller optimized by policy search algorithm, the input is joint information (joint angles, joint velocities) of manipulator and the end effector information (end effector poses, end effector velocities), and the output is joint values. In object placement tasks, the target pose and end effector information can be obtained through images easily, but since the joint information cannot be obtained intuitively through images, it is used directly as the input of CNN policy. Based on the analysis above, the structure of CNN policy is shown as Figure 1.

Figure 1. Convolutional neural network policy structure

At each time step, some feature points are automatically extracted when the original image is passed through convolutional layers and the fully connected layer 1 (FC1); then these feature points are combined with the joint information and used as the input to subsequent fully connected layers; finally the joint values of manipulator can be obtained after FC3. The details of CNN policy in Figure 1 are shown in Table I, where the type of pooling is max pooling.

TABLE I. Layer parameters of the convolutional neural network

Name	Output Dim.	Filter Size	Stride
Conv1	200x200x12	3x3	1
Pool1	100x100x12	2x2	2
Conv2	100x100x10	3x3	1
FC1	26	——	——
FC2	20	——	——
FC3	6	——	——

B. Introduction of FC1

After Conv2, if combining the feature maps with joint information directly as the input of fully connected layers, the CNN policy will be difficult to converge. The reason is that low-dimensional joint information of the manipulator is overwhelmed by high-dimensional image features. At the same time, because the input dimension of fully connected layers is too large, it is difficult to learn the relationship between the input and joint values when training.

Therefore, this algorithm adds FC1 when constructing the CNN policy, which has two main effects. First, the addition of FC1 can greatly reduce the input dimension of subsequent fully connected layers, making the joint information of manipulator not overwhelmed by image features. As the CNN policy shown in Figure 1, the dimension of output is drastically reduced to 26 after FC1, which reaches the same level as the dimension of joint information (12 for UR10 agent). Second, the image features can be numerically matched with joint information by adding FC1, making it easier to perform stochastic gradient descent.

The details of CNN policy when using FC1 are as follows:

1) Convert each feature map after Conv2 into a row vector and concatenate these vectors in order;

2) Pass the vector obtained in 1) through a standard fully connected layer, where the activation function is ReLU to ensure that the output is non-negative;

3) Combine the feature point vector obtained after 2) with the vector of joint information and serve as the input to subsequent fully connected layers.

In FC1, the output dimension is a hyperparameter of CNN policy, which needs to be adjusted in different environments. In the simulation of this paper, the output dimension is taken as 26. Although the number of image features is greatly reduced after FC1, each new feature extracted is a linear combination of all input features. And after adding FC1 in the simulation, CNN policy also achieves better convergence.

C. Loss Function of Convolutional Neural Network Policy

When training CNN policy, the loss function is:

$$l_t = D_{KL}\left(\pi_\theta\left(a_t \mid o_t\right) \| \, p\left(a_t \mid s_t\right)\right) + \lambda R(W) \qquad (10)$$

where λ is the regularization term of the convolutional neural network.

The loss function of CNN policy consists of two parts: a Kullback–Leibler (KL) divergence term describing the similarity between CNN policy and supervised controller, and an L2 regularization term to prevent overfitting, where the coefficient λ is taken as 0.005.

Since the supervised controller of CNN policy is a linear-Gaussian distribution, the KL divergence term in loss function can be expanded when assuming that the form of CNN policy is also a linear-Gaussian. After ignoring the terms in the expansion that do not contain the mean value of controller and CNN policy, the simplification form of loss function can be shown as follows:

$$l_t = \frac{1}{2}\left(\mu_{pt} - \mu_{\pi_\theta t}\right)^T Q_{a,at}\left(\mu_{pt} - \mu_{\pi_\theta t}\right) + \lambda R(W) \qquad (11)$$

where μ_{pt} and $\mu_{\pi_\theta t}$ are the mean values of controller and CNN policy at time step t respectively.

IV. SIMULATIONS

A. Simulations of the Improvement Methods Above

In this part, the improvement methods of cost function, sampling and CNN policy are tested under the robot peg insertion model [14] in MuJoCo simulation package [15], as shown in Figure 2.

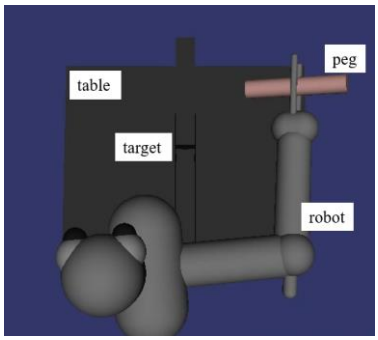

Figure 2. Peg insertion model in MuJoCo simulation package

In each following simulation, four positions of target are defined, and at each position the robot executes the task five times under the corresponding algorithm. The simulation results obtained are as follows.

1) Simulations on the Improvement Method of Cost Function

In addition to comparing the placement error with original algorithm, here the linear weight form which is commonly used in policy search has also been compared. The average placement error in the corresponding weight form and target position is recorded in Table II. It can be seen that the error is minimal at all positions under the weight form in this paper.

TABLE II. Placement error under different weights (cm)

	Position 1	Position 2	Position 3	Position 4
Default	0.99	1.55	1.13	2.97
Linear	0.62	0.48	0.35	0.29
This Paper	**0.20**	**0.25**	**0.33**	**0.25**

2) Simulations of the Modified Sampling Method

The average placement error in the corresponding sampling method and target position is recorded in Table III. It can be seen that the error is obviously reduced using the modified sampling method in position 1, 3 and 4; and although the placement error under the modified sampling method is worse than original method in position 2, the size of these two errors is almost equal.

TABLE III. Placement error under different sampling methods (cm)

	Position 1	Position 2	Position 3	Position 4
Default	0.69	**0.53**	1.57	0.47
This Paper	**0.49**	0.56	**0.78**	**0.37**

When using the modified sampling method, the number of trajectories sampled by controller and CNN policy is also recorded, as shown in Table IV. Compared with the original method, it can be seen that in 30% of cases the modified ε-greedy method can get better trajectories using CNN policy as the main sampling method.

TABLE IV. Number of trajectories sampled by controller and CNN policy

	Controller	CNN policy	Ratio of CNN policy
Simluation 1	100	60	37.50%
Simluation 2	112	48	30.00%
Simluation 3	108	52	32.50%
Simluation 4	116	44	27.50%
Simluation 5	100	60	37.50%
Total	536	264	33.00%

3) Simulations on the Influence of FC1 in CNN Policy

In this simulation first it is necessary to adjust the structure and hyperparameters of CNN policy according to current environment. An additional pooling layer is added after Conv2 and the output dimension of FC1 is set to 15 for this simulation. The learning curves of CNN policy is shown in Figure 3.

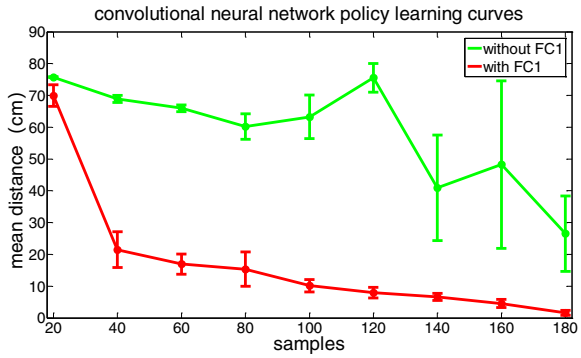

Figure 3. Convolutional neural network policy learning curves

In Figure 3, error bars indicate one standard deviation. We can see that when the CNN policy does not include FC1, the average distance between peg and target position after 180 samples still exceeds 20cm and the standard deviation is also very large. After adding FC1, the optimization process of CNN policy becomes very smooth and the average distance between peg and target position becomes less than 1cm after 180 samples. The CNN policy has achieved a good convergence.

B. Simulations of the Policy Search Algorithm

In this part the adaptability of policy search algorithm to object types and target poses is tested using UR10 manipulator in Gazebo simulator.

1) Simulations of Tasks with Different Object Types

The adaptability of policy search algorithm to object types is tested by setting three kinds of object placement tasks. The tasks performed by the manipulator are: hanging hangers, book placement and peg insertion.

Constraints of the tasks above are different from each other. In the task of hanging hangers, there are strict constraints on the height position and the manipulator needs to complete the task within allowable height. While in the book placement task, there are strict constraints on the width position of the book and the manipulator needs to complete the task within

allowable width. Finally in the peg insertion task, there are strict constraints in all three directions and the requirement for the learning trajectory is also most demanding. The learning curves of these three tasks under policy search algorithm are shown in Figure 4.

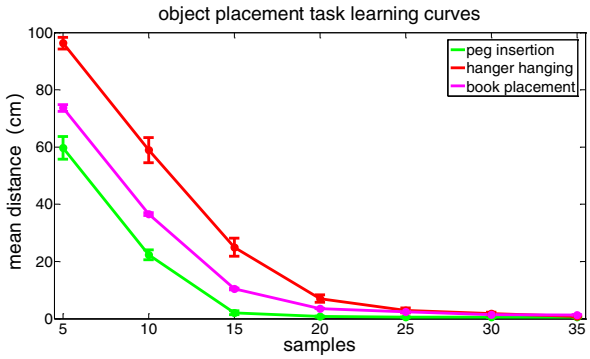

Figure 4. Learning curves of different object placement tasks

In Figure 4, error bars indicate one standard deviation. We can see that with the iteration of policy search algorithm, distance between the object and target position continues to decrease and controller that satisfies the constraints can be acquired after 35 samples in all the three tasks. The completion result of these tasks can be seen in Figure 5 and a good adaptability of policy search algorithm to object types has been shown by this simulation.

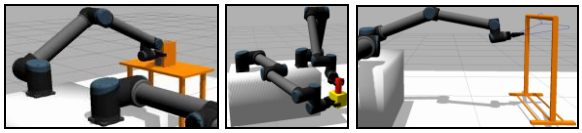

Figure 5. Completion result of book placement, peg insertion and hanger task

2) Simulations of Tasks with Different Target Positions

The adaptability of policy search algorithm to different target positions is tested through book placement and peg insertion tasks. The placement error of these two tasks at different positions is shown in Table V.

TABLE V. Placement error of book placement and peg insertion task

Placement task	Taregt position	Placement error (cm)
Peg insertion	Figure 6.1	0.50
	Figure 6.2	0.52
	Figure 6.3	0.50
Book Placement	Figure 6.4	2.26
	Figure 6.5	1.81
	Figure 6.6	1.76

The completion result of these two tasks at different positions is shown in Figure 6. As can be seen from Table V and Figure 6, the manipulator gets a good result when executing peg insertion tasks and the placement error is also mainly axial error. While in the book placement task, although policy search algorithm can also get the controller that meets the constraints at different positions, the accuracy along length direction of the book is not very good in this simulation.

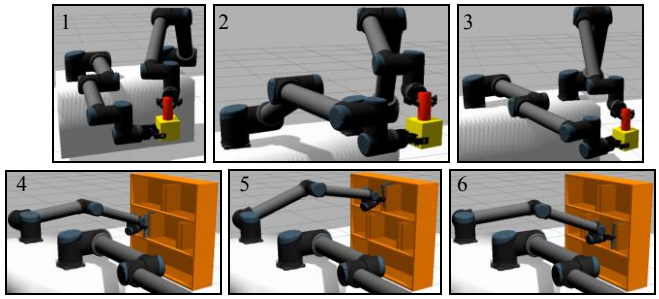

Figure 6. Performance of UR10 at different target positions

3) Simulations of Tasks with Different Target Attitudes

The adaptability of policy search algorithm to different target attitudes is tested through book placement tasks. The target attitude and corresponding placement error is shown in Table VI.

TABLE VI. Attitude error of book placement task

Placement task	Target attitude	Placement error
Book Placement	30°	0.61°
	60°	1.08°
	90°	0.89°

The completion result for different target attitudes is shown in Figure 7. As can be seen from Table VI and Figure 7, the placement error is about 1° in all situations and the algorithm has shown a good adaptability to target attitudes.

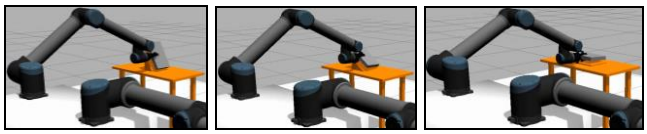

Figure 7. Performance of UR10 at different target attitudes

C. Simulations of the Convolutional Neural Network Policy

In this part the generalization ability of CNN policy is tested through the peg insertion task. In Gazebo simulator, the raw image is obtained by Kinect camera and the target position is adjusted by an auxiliary manipulator during training and testing. Because the workspace of UR10 is very large, volume of the peg and target is correspondingly enlarged so that the pixel number of these objects in raw image can be increased. Environment of this simulation is shown in Figure 8.

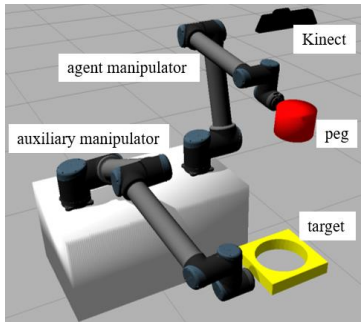

Figure 8. Simulation environment of CNN policy in Gazebo

A square area with sides of 20cm in the workspace of two manipulators is selected to train and test the CNN policy.

During training, the center of target is placed at vertexes of the square area by auxiliary manipulator and the CNN policy is trained at these positions. During testing, besides the four vertexes which have been trained, five extra positions are selected randomly within the square area to test the generalization ability of CNN policy. The learning curve of CNN policy is shown in Figure 9.

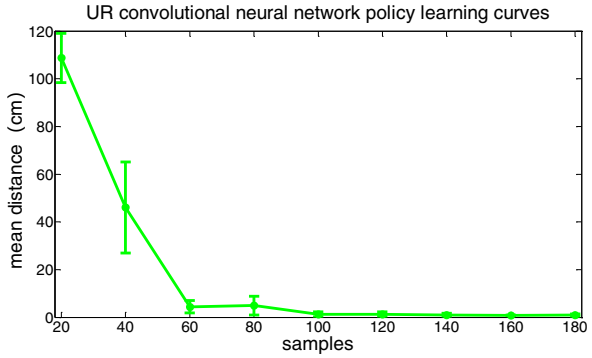

Figure 9. UR10 convolutional neural network policy learning curves

As can be seen from Figure 9, because the supervised controller is very bad in initial iterations of the algorithm, parameters of CNN policy are also very terrible and the manipulator even goes far away from target position under CNN policy in the first iteration. With the optimization of controller, CNN policy has also been effectively trained and finally achieves convergence after 100 samples.

The test result of CNN policy after training is shown in Table VII. It can be seen that the manipulator has executed the task successfully in all training positions and has got over 90% accuracy in testing positions which have not appeared during training. The CNN policy shows a good generalization ability in this task.

TABLE VII. Test result of the convolutional neural network policy

	Train				Test				
Position	1	2	3	4	1	2	3	4	5
Result	5/5	5/5	5/5	5/5	3/5	5/5	5/5	5/5	5/5

V. CONCLUSION

In this paper, a deep reinforcement learning algorithm is utilized for the object placement task in household environment. After modifying the action according to the position-controlled feature of UR10, the manipulator can learn prescribed tasks autonomously in different situations through policy search algorithm and shows a good flexibility of target positions under CNN policy. At the same time, the placement error can be reduced by using the modified cost weights and sampling method.

REFERENCES

[1] S. Levine, N. Wagener, and P. Abbeel. "Learning contact-rich manipulation skills with guided policy search." 2015(2015):156-163.

[2] Montgomery, William, et al. "Reset-Free Guided Policy Search: Efficient Deep Reinforcement Learning with Stochastic Initial States." (2016):3373-3380.

[3] Whitney, D. E. "Force feedback control of manipulator fine motions."*Asme J Dynamic Systems Measurement & Control* 99.2(1977).

[4] Hartley, Richard, and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, 2003.

[5] Deisenroth, Marc Peter, and C. E. Rasmussen. "PILCO: a model-based and data-efficient approach to policy search." *International Conference on International Conference on Machine Learning* Omnipress, 2011:465-472.

[6] Vogt, David, et al. "A system for learning continuous human-robot interactions from human-human demonstrations." *IEEE International Conference on Robotics and Automation*, 2017.

[7] Kober, Jens, and J. Peters. "Reinforcement Learning in Robotics: A Survey." *International Journal of Robotics Research* 32.11(2013):1238-1274.

[8] Mnih, Volodymyr, et al. "Playing Atari with Deep Reinforcement Learning." *Computer Science* (2013).

[9] Devin, Coline, et al. "Deep Object-Centric Representations for Generalizable Robot Learning." (2017).

[10] Haarnoja, Tuomas, et al. "Composable Deep Reinforcement Learning for Robotic Manipulation." *IEEE International Conference on Robotics and Automation*, 2018.

[11] S. Levine, et al. "End-to-end training of deep visuomotor policies." *Journal of Machine Learning Research* 17.1(2016):1334-1373.

[12] Wang H, Banerjee A. Bregman Alternating Direction Method of Multipliers[J]. *Mathematics*, 2013:2816-2824.

[13] Tassa, Yuval, T. Erez, and E. Todorov. "Synthesis and stabilization of complex behaviors through online trajectory optimization." *Ieee/rsj International Conference on Intelligent Robots and Systems* IEEE, 2012:4906-4913.

[14] Chelsea Finn, Marvin Zhang, Justin Fu, William Montgomery, Xin Yu Tan, Zoe McCarthy, Bradly Stadie, Emily Scharff, Sergey Levine. Guided Policy Search Code Implementation. 2016. Software available from rll.berkeley.edu/gps.

[15] E. Todorov, T. Erez, and Y. Tassa. MuJoCo: A physics engine for model-based control. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2012.