Obstacle Avoidance for Robotic Manipulator in Joint Space via Improved Proximal Policy Optimization

Wang, Yongliang; Mohades Kasaei, Seyed

Published in:
ArXiv

Obstacle Avoidance for Robotic Manipulator in Joint Space via Improved Proximal Policy Optimization

Yongliang Wang and Hamidreza Kasaei

***Abstract*— Reaching tasks with random targets and obstacles can still be challenging when the robotic arm is operating in unstructured environments. In contrast to traditional model-based methods, model-free reinforcement learning methods do not require complex inverse kinematics or dynamics equations to be calculated. In this paper, we train a deep neural network via an improved Proximal Policy Optimization (PPO) algorithm, which aims to map from task space to joint space for a 6-DoF manipulator. In particular, we modify the original PPO and design an effective representation for environmental inputs and outputs to train the robot faster in a larger workspace. Firstly, a type of action ensemble is adopted to improve output efficiency. Secondly, the policy is designed to join in value function updates directly. Finally, the distance between obstacles and links of the manipulator is calculated based on a geometry method as part of the representation of states. Since training such a task in real-robot is time-consuming and strenuous, we develop a simulation environment to train the model. We choose Gazebo as our first simulation environment since it often produces a smaller Sim-to-Real gap than other simulators. However, the training process in Gazebo is time-consuming and takes a long time. Therefore, to address this limitation, we propose a Sim-to-Sim method to reduce the training time significantly. The trained model is finally used in a real-robot setup without fine-tuning. Experimental results showed that using our method, the robot was capable of tracking a single target or reaching multiple targets in unstructured environments. The video is available at https://youtu.be/xm5lNAgYaWM**

I. INTRODUCTION

Goal reaching task is considered as one of the fundamental capabilities for robotic manipulators, which is accomplished by motion planning in traditional methods [1]. When the targets and obstacles change at random, the motion planning task for high-degree of freedom manipulators will become notoriously challenging in such uncertain environments as mathematical models that are complex and difficult to establish [2]. Meanwhile, traditional control approaches are often unable to navigate in unstructured environments. Besides, most methods are based on solving inverse kinematics or dynamics equations to map the task space to joint space for collision avoidance, which requires high accurate robot dynamics model and is not easy to generalize for diverse tasks and various robots [3], [4].

Deep reinforcement learning (DRL) is now frequently used in robotic manipulation in place of analytical techniques in traditional control systems [5], [6]. For instance, Adarsh Sehgal $et\ al.$ proposed a deep deterministic policy gradient (DDPG) and hindsight experience replay (HER) based method using of the genetic algorithm (GA) to fine-tune the parameters values. They experimented on six robotic manipulation tasks and got better results than baselines [7]. Franceschetti $et\ al.$ proposed an extensive comparison of the trust region policy optimization (TRPO) and deep Q-Network with normalized advantage functions (DQN-NAF) with respect to other state of the art algorithms, namely DDPG and vanilla policy gradient (VPG) [8]. Unlike our work, these research only focus on single target position reaching.

For multi-target trajectory planning, Wang $et\ al.$ introduced an action ensembles based on poisson distribution (AEP) to PPO, their method could be easily extended to realize the task that the end-effector tracks a specific trajectory [9]. For space robots, the workspace is enough to complete task, but for industrial robots, it is insufficient. Thus, the algorithm requires further development. In another work, Kumar $et\ al.$ proposed a simple, versatile joint-level controller via PPO. Experiments showed the method capable of achieving similar error to traditional methods, while greatly simplifying the process by automatically handling redundancy, joint limits, and acceleration or deceleration profiles [10]. Nevertheless, the output of neural network is velocity of the end-effector. Additionally, the majority of DRL-based research completes learning in task space rather than joint space, which is prone to produce a weak results for reaching tasks. Furthermore, such approaches still need to calculate the inverse kinematics and cannot accomplish reaching tasks when obstacles are close to the manipulator's links.

In this paper, we propose an improved PPO to tackle these problems, which enables the 6-DoF manipulator to accomplish reaching multi-target without colliding with obstacles. In comparison to the previous works [11], [12], [13], [14], our main contributions are threefold: 1) Introducing an action ensembles method to enhance the efficiency of the policy. 2) Designing an adaptive discount factor for PPO, which makes the policy join in value function update directly. 3) Calculating the distance between obstacles with manipulator's links utilizing a type of geometry method, as part of state representation, which is beneficial for reaching task with obstacles. Experimental results showed that the proposed approach performed better than other baselines in various test scenarios.

II. PRELIMINARY

We aim to develop an efficient method for manipulators in reaching tasks with obstacle avoidance, which necessitates

Yongliang Wang and Hamidreza Kasaei both are with Department of Artificial Intelligence, Bernoulli Institute, Faculty of Science and Engineering, University of Groningen, The Netherlands

Emails: {yongliang.wang, hamidreza.kasaei}@rug.nl

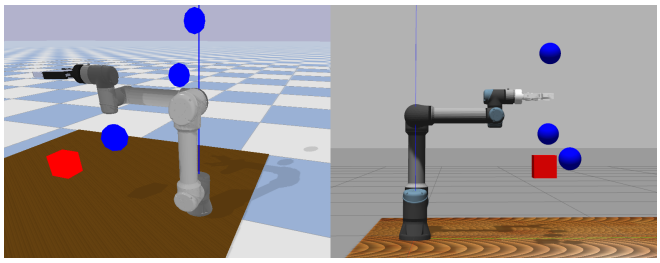

Fig. 1. The environment of 6-DoF manipulator in the Pybullet (*left*) and the Gazebo (*right*): The goal is shown by a red block and obstacles are highlighted by blue spheres.

the manipulator to safely interact with the environment numerous times. As Gazebo owns better compatibility than Pybullet in combining with ROS, we initially design our training process in Gazebo. Nevertheless, DRL methods suffer from long training time in Gazebo. To overcome such a time-consuming problem, we develop a fairly similar environment in Pybullet to initially train the model there, and then transfer and evaluate the learnt model in Gazebo through a Sim-to-Sim transfer. Finally, using a Sim-to-Real transfer process, we assess the efficiency of the learned model in real robot using the trained model.

A. Environment

As previously stated, we develop two simulation environments, one in Gazebo, and the other in Pybullet. As shown in Fig. 1, in both environments, a UR5e robot equipped with robotiq_140 is used as the manipulator. During training and testing phases, we randomly set the pose of the target (shown in red) and obstacles (shown in blue) in workspace.

B. Proximal Policy Optimization

PPO, one of the state-of-the-art on-line DRL methods, is a type of policy gradient training that alternates between sampling data through environmental interaction and optimizing a clipped surrogate objective function using stochastic gradient descent [15]. The clipped surrogate objective function improves training stability by limiting the size of the policy change at each step. In PPO, the clipped surrogate objective function is designed as follows:

$$\begin{aligned} \boldsymbol{L^{CLIP}}(\theta^{\pi}) = \dot{\mathbb{E}}_t[min(r_t(\theta^{\pi})\hat{A}_t, \\ clip(r_t(\theta^{\pi}), 1-\epsilon, 1+\epsilon)\hat{A}_t)] \end{aligned} \tag{1}$$

$$\hat{A}_t = \delta_t + (\gamma\lambda)\delta_{t+1} + \cdots + \cdots + (\gamma\lambda)^{T-t+1}\delta_{T-1} \tag{2}$$

$$\delta_t = r_t + \gamma V(s_{t+1}) - V(s_t) \tag{3}$$

$$V(s) = \mathbb{E}_{s,a\sim\pi}[G(s)|s] \tag{4}$$

$$G(s) = \sum_{i=t}^{\infty} \gamma^{i-t} r(s_i) \tag{5}$$

where θ^{π} is the parameters of policy neural network, $\hat{A}_t$ represents the generalized advantage estimator (GAE) and is used to calculate the policy gradient. The reward value at t is shown by r_t, and the ϵ is a constant between 0 and 1, which is set to 0.2 in the baseline algorithm. $\gamma = 0.99$, $V(s)$ refers to the expected return of state s and G represents the discounted cumulative reward. Likewise, $V_{target}(s)$ is the target value. Additionally, the value loss function is expressed as follows:

$$\boldsymbol{L}_V(\theta^V) = \mathbb{E}_{s,a\sim\pi}[(V(s) - V_{target}(s))^2] \tag{6}$$

C. Sim-to-Sim and Sim-to-Real Transfer

We train the policy in Pybullet first and then transfer the learnt policy from Pybullet to Gazebo in order to overcome the time-consuming training process in Gazebo for DRL approaches. Finally, we deploy the model in our real-robot to evaluate how well it performed in real-world circumstances. Although there are so many research about Sim-to-Real in the filed of learning of navigation and manipulation policies [16], [17], most of research mainly focused on bridging the Sim-to-Real gap in domain adaptation [18], [6]. Furthermore, visual cues were considered as input information directly or indirectly [19]. However, in our work, obstacle avoidance is accomplished in joint space. On the other hand, our primary goal is to achieve the best accuracy in simulation for downstream applications in real-world scenarios. Sim-to-Sim transfer is a productive method for reducing training time and evaluate the robustness of the proposed approach over noises and inaccurate robot's model before deploying the learnt model on a real-robot platform. Therefore, we take into account both Sim-to-Sim and Sim-to-Real transfers, allowing us to quickly train and test the proposed model in various tasks and domains.

III. STRATEGY FOR LEARNING

We adopt the PPO to accomplish obstacle avoidance with the mapping from task space to joint space. For reinforcement learning, one of the important aspects is to devise a good learning strategy, which includes selecting appropriate state and action representation. The strategy is implemented as a deep policy, which is designed as a multi-layer perception network with two hidden-layers.

A. State and Action Representation

It is crucial for DRL methods to chose appropriate state and action space. Most researchers prefer to represent both states and actions in task space, which is ineffective for avoiding collision between links and obstacles. To accomplish collision avoidance in whole workspace, we consider the position of 6 joints, end-effector, and targets as part of state representation. Furthermore, the errors in X, Y, and Z axes, and distance between obstacles and the five links are also considered as state representation. It is worth mentioning that we do not consider the distance of the obstacles to the base link. Therefore, the state is represented as a vector: $s \in \mathbb{R}^{19}$. For action representation, we consider the position of the six joints in order to avoid complex and time-consuming inverse kinematics calculations and map from task space to joint space. In the following subsections, we discuss the state and action spaces in more detail.

1) States in Reaching Task without Obstacles: In the case of obstacle free goal reaching task, we represent the state as:

$$\boldsymbol{s}_t =< \boldsymbol{q}_t, \boldsymbol{p}_e, \boldsymbol{p}_t, \boldsymbol{error} > \tag{7}$$

where $\boldsymbol{q}_t = (q_{t1} \dots q_{t6})$ is the position of the six joints, $\boldsymbol{p}_e = (p_{ex}, p_{ey}, p_{ez})$ represents the position of the end-effector, $\boldsymbol{p}_t = (p_{tx}, p_{ty}, p_{tz})$ is referred to the target position. $\boldsymbol{error} = (e, e_x, e_y, e_z)$ is the error vector including absolute distance and distances in X, Y, Z axes, respectively.

2) States in Reaching Task with Obstacles: When there are obstacles in the environment, the state is represented as:

$$\boldsymbol{s}_t =< \boldsymbol{q}_t, \boldsymbol{p}_e, \boldsymbol{p}_t, \boldsymbol{error}, \boldsymbol{d}_{obs} > \tag{8}$$

where $\boldsymbol{d}_{obs}$ is the shortest distances between obstacles and links in space. As depicted in Fig. 2, to calculate the distance between the obstacle and each link in joint space, we transform it into a geometric problem to find the shortest distance between any point in space and different links. We provide an example on the left side of Fig. 2 to make it clearer. Let a three-dimensional line be specified by two points, $p_1 = (x_1, y_1, z_1)$ and $p_2 = (x_2, y_2, z_2)$, where $\cdot$ represents the dot product. Therefore, a vector along the line is given by the following equation:

$$\boldsymbol{v} = \begin{bmatrix} x_1 + (x_2 - x_1)t \\ y_1 + (y_2 - y_1)t \\ z_1 + (z_2 - z_1)t \end{bmatrix}$$

The squared distance between a point on the line with parameter t and a point $p_0 = (x_0, y_0, z_0)$ is therefore:

$$\begin{aligned} d^2 = &[(x_1 - x_0) + (x_2 - x_1)t]^2 \\ &+[(y_1 - y_0) + (y_2 - y_1)t]^2 \\ &+[(z_1 - z_0) + (z_2 - z_1)t]^2 \end{aligned} \tag{9}$$

Set $d(d^2)/dt = 0$ and solve for t to obtain the shortest distance:

$$\boldsymbol{t} = -\frac{(x_1 - x_0) \cdot (x_2 - x_1)}{|x_2 - x_1|^2} \tag{10}$$

The shortest distance can then be calculated by plugging Eq. (10) back into Eq. (9). Thus, as shown on the right side of Fig. 2, we consider each link of the robot as the line and the obstacle as the point. We then calculate the shortest distance between every obstacle and link.

3) Actions: In both cases, the action space is represented by a vector, $\boldsymbol{a}_t =< \dot{\boldsymbol{q}}_t >$, where $\dot{\boldsymbol{q}}_t = (\dot{q}_{t1} \dots \dot{q}_{t6})$ represents the position of the six joints.

B. Reward Function

The following function represents how we calculate the reward for various situations:

$$\boldsymbol{R}(s, a) = -[\omega_1 e^2 + \ln(e^2 + \tau_e) + \omega_2 \sum_{i=1}^{n} \psi_i] \tag{11}$$

$$\psi_i = max(0, 1 - ||d_i||/d_{max}) \tag{12}$$

where $e = \|p_t - p_e\|$ refers to the euclidean distance between the target pose and the end-effector. The middle term $(\ln(\cdot))$ encourages the end-effector error tend to be zero, and the ψ_i represents the penalties of obstacle avoidance, ω_1, and ω_2 are two coefficients, and τ_e represents the threshold on error between the end-effector and the target pose. Based on trial and errors, we set $\omega_1 = 10^{-3}$, $\omega_2 = 0.1$, $d_{max} = 0.05$ and $\tau_e = 10^{-4}$.

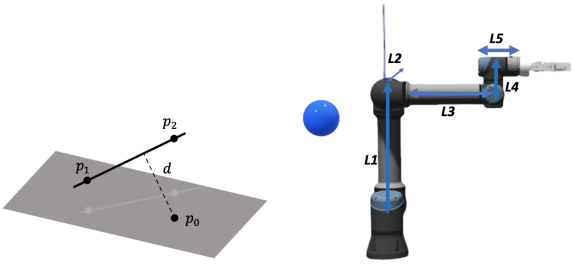

Fig. 2. The distance calculation between a point and a line in space (*left*). $\boldsymbol{d}_{obs}$ is obtained by calculating the distance between the obstacle (blue) and 5 links.

C. Neural Network Structure

In our system, the Actor and Critic (AC) neural network consist of three layers, where each layer consists of 256 neurons. The first two layers use $tanh$ activation function. The only distinction between actor and critic networks is that the critic network generates only a single scalar value, while the actor produces a vector of six values, representing the robot joints' position. For both networks, we consider the Adam optimizer. The overall framework of our approach is depicted in Fig. 3.

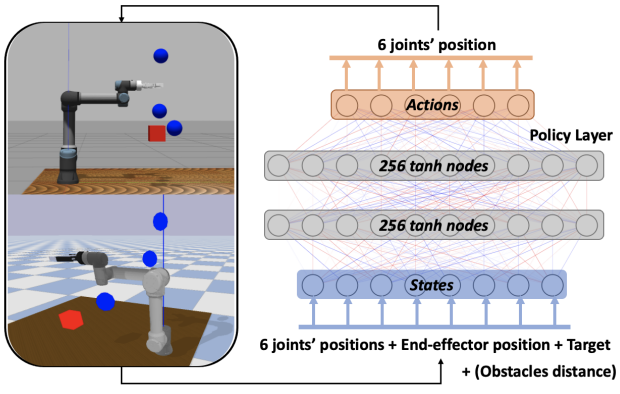

Fig. 3. The framework of strategy for learning: The state of robot consists of the current joint angles q_t , the position of end-effector p_e, and the target position p_t, which extends with the distance of three obstacles when the task is with obstacles. The grey layers are the network structure. The action layer produces the desired joint angles.

IV. IMPROVED PROXIMAL POLICY OPTIMIZATION

Original PPO does not perform well in solving complex robotic problems (i.e., the discussed reaching task while there are obstacles in the environment). In particular, we observed that PPO took too long time to be trained in the

entire workspace, and the accuracy was poor for reaching task, even without collision avoidance. To overcome these limitations, we propose two improvements for the PPO to achieve better results, which are discussed in detail in the following subsections.

A. Action ensembles Based on Poisson Distribution

For most tasks that applied learning methods in robotics, Gaussian distribution is utilized to describe the optimal policy distribution, which is on the grounds that Gaussian distribution is more realistic. In reaching task, the distribution of choosing action can be considered as: $\pi_\theta(a_t|s_t) \sim N(\mu_\theta(s_t), \delta_\theta)$, where δ represents the uncertainty and unstable of distribution to output optimal action. To some extent, average the multiple outputs can solve this problem but it will limit the exploration ability at the initial steps and make the policy easily prone to local optima.

To make the policy robust, balance the exploration and avoid inclining to optima, we select the number of samples through Poisson distribution [9]. In particular, the specific calculation is defined as:

$$i \sim clip(Poisson(\beta), 1, 2\beta), \quad \beta = 1 + \alpha\frac{e_n}{e_a} \tag{13}$$

$$\boldsymbol{a}_{t,j} \sim N(\mu_\theta(s_t), \delta_\theta), \quad \boldsymbol{a}_t = \underset{j}{mean}(\boldsymbol{a}_{t,j}) \tag{14}$$

where β indicates the Poisson distribution mean, $\alpha = 5$, $j \in [1, j)$, e_n and e_a represent the number of episodes at the current episode and the final episode, respectively.

B. AC Architecture with Policy Feedback

PPO uses the standard AC architecture, which means that the critic network estimates the value function that the actor network uses to improve policy performance. However, the policy does not participate in the update of value function directly, which increases the instability of DRL algorithms. Thus, inspired by [20], we include the policy in the value function update. Using this strategy, the critic network can recognize policy differences rapidly. To put the strategy into action, we utilize an adaptive clipped discount factor:

$$\gamma(s, a; \eta) = clip(\pi(s, a), \eta, 1) \quad \eta \in (0.55, 0.99) \tag{15}$$

in which $\pi(s, a)$ represents the policy and the adaptive γ can join in the update of critic network.

C. Improved PPO

According to previous advancements, the loss functions of actor-critic networks can be represented as follows:

$$\begin{aligned}\boldsymbol{L}(\theta^\pi) = \dot{\mathbb{E}}_t[&min(r_t(\theta^\pi)A_t, \\ &clip(r_t(\theta^\pi), 1-\epsilon, 1+\epsilon)A_t)]\end{aligned} \tag{16}$$

$$\boldsymbol{L}(\theta^V) = \dot{\mathbb{E}}_t[(R_t^\pi - V_t)^2] \tag{17}$$

Fig. 4 is the framework of the improved PPO. Algorithm 1 contains the pseudocode of the improved PPO.

By combining all the proposed strategies together, the entire algorithm for robot to avoid obstacles while reaching the goal is summarized in Algorithm 2.

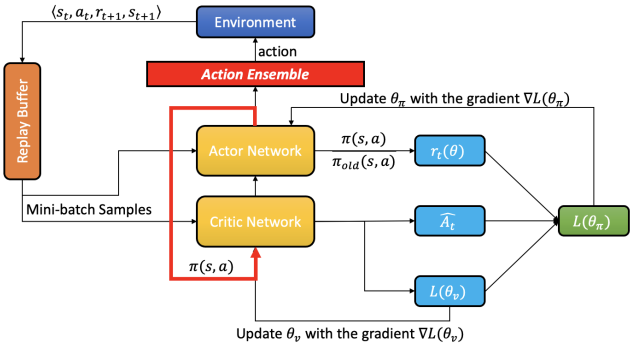

Fig. 4. The framework of the improved PPO: The action ensemble (*shown in red block*) and the policy feedback (*red line*) are the improvements that we proposed.

Algorithm 1 Improved PPO Pseudocode

Orthogonal initialize the actor and critic networks
Initialize optimizer as Adam with learning rates
Set λ, $a_{max} = 3.14$, clip parameter: ϵ
Set AEP parameter: $\alpha = 5$
Set Policy Feedback parameter: $\eta \in [0.55, 0.99]$
while *True* **do**
  Get s_t from the environment
  Sample a_t from the baseline policy $\pi_{b\theta}(a_t|s_t)$
  Using $\boldsymbol{a}_t \sim AEP(\mu_\theta(s_t), \delta_\theta)$ to publish $\boldsymbol{a}_t$
  Execute a_t, get $< s_t, a_t, \pi_\theta, r_t, s_{t+1} >$ and store them into buffer D
  Calculate clipped discount factor $\gamma(s, a; \eta)$ by Eq.(16)
  Compute reward: $R_t^\pi = \sum_{i=t}^{T} r_i \prod_{j=t}^{n} \gamma(s, a; \eta)$
  Compute advantage function: $A_t^\pi = r_t + \gamma V_{t+1}^\pi - V_t^\pi$
  Use Eq.(17) to accumulate gradients with respect to θ_π
  Use Eq.(18) to accumulate gradients with respect to θ_V
end

V. EXPERIMENTS

To illustrate the efficiency of the improved PPO, we performed a serious of experiments in simulation and real-robot settings. We divided our experiments to four steps. In the first round of experiment, we compared the proposed approach with two baseline methods. Then, we measured the success rate for completing the goal reaching task with and without obstacles. Afterwards, the trained model was transferred from Pybullet to Gazebo, and also used directly on real-robot without fine-tuning. We used the same code and network in both real and simulation experiments. Note that all tests were performed with a PC running Ubuntu 20.04 with a 3.20 GHz Intel Xeon(R) $i7$, and a Quadro P5000 NVIDIA.

A. Sim-to-Sim Transfer

We performed two round of experiments to measure the training time as well as the performance of the robot in goal reaching task. We ran one round of experiments in Gazebo environment and the other round in Pybullet for 10^6 time

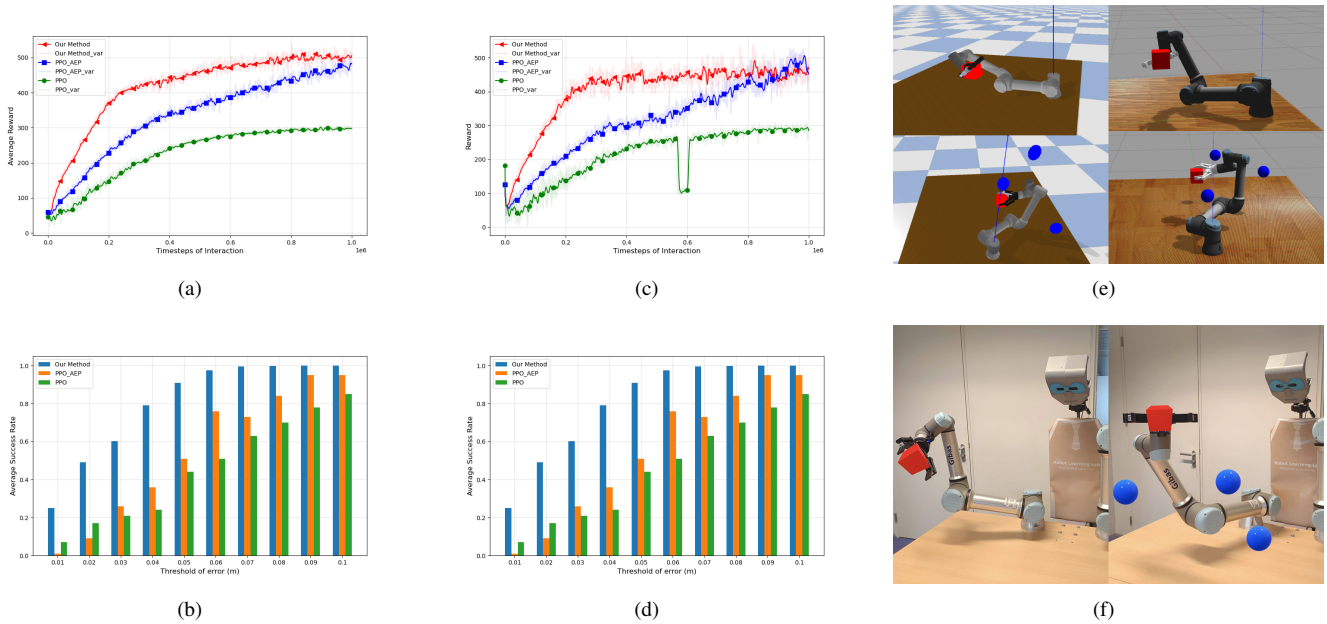

Fig. 5. Summary of experiments: (a) Comparison result for reaching task without obstacles utilizing 5 random seeds; (b) Comparison of success rate on reaching task without obstacles; (c) Comparison result for reaching task with obstacles utilizing 1 random seed; (d) Comparison of success rate on reaching task with obstacles; (e) The train and test simulation experiments in Pybullet and Gazebo; (f) The real robot experiments.

Algorithm 2 The Manipulator Control via Improved PPO

for *number of epochs* **do**
  Set i_epoch as random seed
  Orthogonal initialize the actor and critic networks
  Set λ, maximum action, learning rates, clip parameter ϵ
  for *number of episodes* **do**
    Set a target position p_t randomly
    for *numbers of maximum time steps* **do**
      Get s_t from the environment
      Chose the action using $\boldsymbol{a}_t \sim AEP(\mu_\theta(s_t), \delta_\theta)$
      Clip $\boldsymbol{a}_t$ to ensure safety for the environment
      Execute a_t, get $< s_t, a_t, \pi_\theta, r_t, s_{t+1} >$ and store them into buffer D
    end
  end
  for *every time step t* **do**
    for k *epochs* **do**
      Select a minibatch b_k in D
      Utilize the Algorithm 1 to Update networks
    end
  end
end

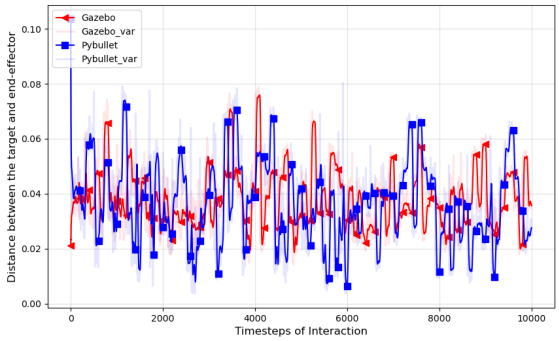

Fig. 6. The distance error for reaching task without obstacles in Gazebo and Pybullet using the same trained model obtained from Pybullet.

TABLE I
TRAINING TIME COMPARISON

Task	Time steps	Gazebo	Pybullet
Reaching without obstacles	10^6	50 h 06 min	2 h 40 min
Reaching with obstacles	10^6	**	11 h 28 min

steps. Table I summarizes the time spent during training in Gazebo and Pybullet. By comparing these results, it is clear that training the agent in Gazebo required much more time than training in Pybullet. Regarding the accuracy of reaching task, we ran 100 obstacle-free reaching tasks using the same model. Fig. 6 showed that the agent could achieve equivalent performance, with the average error of 0.04 ± 0.02 m.

We also transferred the model that has been trained in Pybullet to the Gazebo to check the possibility of Sim-to-Sim Transfer. We ran similar experiments (100 obstacle-free reaching tasks), and compared the obtained results with the results of the model that was trained in Pybullet. Experimental results showed that the agent could achieve equivalence performance in both Pybullet and Gazebo. This experiment showed the usefulness of Sim-to-Sim transfer regarding training time and goal reaching accuracy.

B. Comparison with Other Methods

As the output of the neural network was a multivariable and PPO was one of widely used state-of-the-art on-policy algorithm in the field of robotic control, we chose the following methods as baseline: PPO [15] and PPO-AEP [9].

Fig. 5 shows the comparison between the proposed method with the baseline methods. It should be noted that the baselines used the same neural network, parameters, state-action representation and reward function. The learning curves, as shown in Fig. 5 (a) and (c), indicated that our method could get higher reward in the same condition. Fig. 5 (b) and (d) also illustrated that the success rate of our method reached the higher value in reaching task with and without obstacles.

C. Reaching Task without Obstacle Avoidance

The previously mentioned strategy was used to train the neural network, which was then used in a UR5e robot arm. Based on our experiments, the trained neural network in Pybullet could be directly used in Gazebo and real robot. Considering the workspace in our real-robot setup, the whole workspace was defined as a quarter spherical annulus with major radius 0.95m, minor radius 0.4m. We also established a set of constraints for six joints in order to avoid collisions between the manipulator and the table, as well as self collisions among all joints during test.

For observing the performance of reaching task without obstacle avoidance, the success rate was produced by recording the number of times the robot could reach to the target pose within a predefined threshold, ranging from 1cm to 10cm with the interval of 1cm. For each of the threshold value, we performed 100 experiments by randomly setting the target point in each experiment. We repeated such experiments with five random seeds and reported the average success rate for each threshold value. The obtained results is summarized in Fig. 5 (b). By comparing the results, it is visible that our approach outperformed the other baselines. In particular, by setting the threshold to 5cm, our method achieves 100 percent success rate.

D. Reaching Task with Obstacle Avoidance

As our main goal was to propose an efficient method of collision avoidance for robotic manipulators, we tested our method in an environment with random obstacles and a target. In this round of experiments, 3 spherical obstacles with size 0.05m were placed randomly in the workspace. The learning curve of our method in reaching task with obstacles is shown in Fig. 5 (c). We didn't average out the rewards as in the previous section because of the longer training time (≈ 11.28 hours). Although our method could end the training faster it didn't reach the highest value at the end of the training as shown in Fig. 5 (c). We hypothesis that it happened because we just ran the experiment once and it was an occasional training session. Fig. 5 (d) was the success rate obtained by the same method as the previous task, which showed our method still performed best.

E. Sim-to-Real Transfer

To validate the efficient of our algorithm in real robot, we performed 10 real-robot experiments of reaching task with and without obstacles. Fig. 5 (e) and (f) show two snapshots of the robot during these experiments. Results demonstrated that our algorithm performs as good as it worked in simulation. A video summarizing these experiments has been attached to the paper as a supplementary material.

VI. CONCLUSIONS

In this paper, we presented an efficient DRL method for manipulator obstacle avoidance at the joint-level. The learning strategy was designed as a multi-layer neural network optimized by an improved PPO. Particularly, to begin with, we designed a special calculation for the shortest distance between obstacles and links of the manipulator. Then, we proposed two improvements for the original PPO, which outperformed other relative baseline methods. Finally, we utilized a sim-to-Sim and Sim-to-Real transfer method to train and verify our method rapidly, which could guide other researchers in testing their modification without suffering time-consuming. To sum up, our proposed method was efficient for reaching tasks with obstacles and it is simple to generalize to various robots and robotic tasks. In the continuation of this work, we would like to look into the feasibility of using the proposed approach in dynamic scenarios where the pose of the target and the obstacles changes over time.

REFERENCES

[1] P. Martin and J. D. R. Millán, "Reinforcement learning of sensor-based reaching strategies for a two-link manipulator," in *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems. IROS'96*, vol. 3. IEEE, 1996, pp. 1345–1352.

[2] E. Aljalbout, J. Chen, K. Ritt, M. Ulmer, and S. Haddadin, "Learning vision-based reactive policies for obstacle avoidance," in *Conference on Robot Learning*. PMLR, 2021, pp. 2040–2054.

[3] D. Zhou, R. Jia, H. Yao, and M. Xie, "Robotic arm motion planning based on residual reinforcement learning," in *2021 13th International Conference on Computer and Automation Engineering (ICCAE)*. IEEE, 2021, pp. 89–94.

[4] S. Rodriguez, X. Tang, J.-M. Lien, and N. M. Amato, "An obstacle-based rapidly-exploring random tree," in *Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006.* IEEE, 2006, pp. 895–900.

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

[6] Y. Zhu, R. Mottaghi, E. Kolve, J. J. Lim, A. Gupta, L. Fei-Fei, and A. Farhadi, "Target-driven visual navigation in indoor scenes using deep reinforcement learning," in *2017 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2017, pp. 3357–3364.

[7] A. Sehgal, N. Ward, H. La, and S. Louis, "Automatic parameter optimization using genetic algorithm in deep reinforcement learning for robotic manipulation tasks," *arXiv preprint arXiv:2204.03656*, 2022.

[8] A. Franceschetti, E. Tosello, N. Castaman, and S. Ghidoni, "Robotic arm control and task training through deep reinforcement learning," in *International Conference on Intelligent Autonomous Systems*. Springer, 2022, pp. 532–550.

[9] S. Wang, X. Zheng, Y. Cao, and T. Zhang, "A multi-target trajectory planning of a 6-dof free-floating space robot via reinforcement learning," in *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2021, pp. 3724–3730.

[10] V. Kumar, D. Hoeller, B. Sundaralingam, J. Tremblay, and S. Birchfield, "Joint space control via deep reinforcement learning," in *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2021, pp. 3619–3626.

[11] Y. Gu, Y. Cheng, K. Yu, and X. Wang, "Anti-martingale proximal policy optimization," *IEEE Transactions on Cybernetics*, 2022.

[12] H.-L. Hsu, Q. Huang, and S. Ha, "Improving safety in deep reinforcement learning using unsupervised action planning," in *2022 International Conference on Robotics and Automation (ICRA)*. IEEE, 2022, pp. 5567–5573.

[13] P. Sadhukhan and R. R. Selmic, "Multi-agent formation control with obstacle avoidance using proximal policy optimization," in *2021 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*. IEEE, 2021, pp. 2694–2699.

[14] T. Kobayashi, "Proximal policy optimization with relative pearson divergence," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2021, pp. 8416–8421.

[19] A. Pashevich, R. Strudel, I. Kalevatykh, I. Laptev, and C. Schmid, "Learning to augment synthetic images for sim2real policy transfer," in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2019, pp. 2651–2657.

[15] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[16] T. Zhang, K. Zhang, J. Lin, W.-Y. G. Louie, and H. Huang, "Sim2real learning of obstacle avoidance for robotic manipulators in uncertain environments," *IEEE Robotics and Automation Letters*, vol. 7, no. 1, pp. 65–72, 2021.

[17] K. Bousmalis, A. Irpan, P. Wohlhart, Y. Bai, M. Kelcey, M. Kalakrishnan, L. Downs, J. Ibarz, P. Pastor, K. Konolige *et al.*, "Using simulation and domain adaptation to improve efficiency of deep robotic grasping," in *2018 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2018, pp. 4243–4250.

[18] K. Fang, Y. Bai, S. Hinterstoisser, S. Savarese, and M. Kalakrishnan, "Multi-task domain adaptation for deep learning of instance grasping from simulation," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2018, pp. 3516–3523.

[20] Y. Gu, Y. Cheng, C. P. Chen, and X. Wang, "Proximal policy optimization with policy feedback," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 2021.