An Open-Source Multi-Goal Reinforcement Learning Environment for Robotic Manipulation with Pybullet*

Xintong Yang[1[0000−0002−7612−614X]], Ze Ji[1[0000−0002−8968−9902]**], Jing Wu[2[0000−0001−5123−9861]], and Yu-Kun Lai[2[0000−0002−2094−5680]]

[1] Centre for Artificial Intelligence, Robotics and Human-Machine Systems (IROHMS), School of Engineering, Cardiff University, Cardiff, UK
{yangx66, jiz1}@cardiff.ac.uk

[2] School of Computer Science and Informatics, Cardiff University, Cardiff, UK
{wuj11, laiy4}@cardiff.ac.uk

Abstract. This work re-implements the OpenAI Gym multi-goal robotic manipulation environment, originally based on the commercial Mujoco engine, onto the open-source Pybullet engine. By comparing the performances of the Hindsight Experience Replay-aided Deep Deterministic Policy Gradient agent on both environments, we demonstrate our successful re-implementation of the original environment. Besides, we provide users with new APIs to access a joint control mode, image observations and goals with customisable camera and a built-in on-hand camera. We further design a set of multi-step, multi-goal, long-horizon and sparse reward robotic manipulation tasks, aiming to inspire new goal-conditioned reinforcement learning algorithms for such challenges. We use a simple, human-prior-based curriculum learning method to benchmark the multi-step manipulation tasks. Discussions about future research opportunities regarding this kind of tasks are also provided.

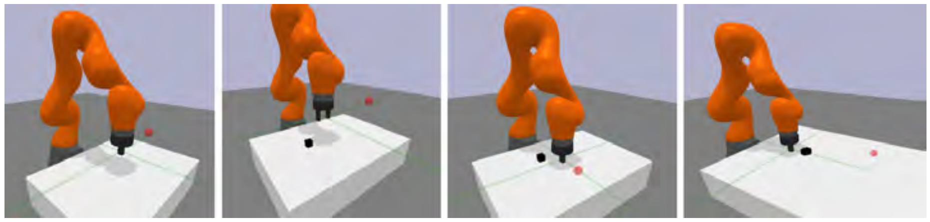
Keywords: Deep Reinforcement Learning · Simulation environment · Pybullet · Robotic Manipulation · Multi-goal learning · Continuous control.

1 Introduction

Due to the difficulties of reinforcement learning in real-world environments [5], developing simulation environments for robotic manipulation tasks becomes increasingly important. In addition to the requirement of being realistic, such simulation is also required to be efficient in generating synthetic data for training deep reinforcement learning (DRL) agents. Currently, the most popular physics engines in DRL research are Mujoco [13,16,17] and Pybullet [3,4,15]. Mujoco is known to be more efficient than Pybullet [6], but it is not open-sourced.

* The authors thank the China Scholarship Council (CSC) for financially supporting Xintong Yang in his PhD programme (No. 201908440400).

** Corresponding author



(a) From left to right: KukaReach, KukaPickAndPlace, KukaPush and KukaSlide.

(b) From left to right: BlockRearrange, ChestPush, ChestPickAndPlace, BlockStack

Fig. 1. The robotic arm manipulation tasks. (a) Single-step tasks, reproduced from the original OpenAI Gym multi-goal manipulation tasks [1,8] (described in section 2.1). (b) Multi-step tasks (described in section 2.2).

The cost of a Mujoco institutional license is at least $3000 per year [9], which is often unaffordable for many small research teams, especially when a long-term project depends on it. To promote wider accessibility to such resource and support DRL research in robot arm manipulations, we introduce an open-source simulation software, **PMG**, **P**ybullet-based, **M**ulti-goal, **G**ym-style [2]. It is written in Python, the most popular language in recent machine learning research[3].

The manipulation tasks proposed by [1,13] focus on goal-condition reinforcement learning (GRL) in sparse reward scenarios. GRL aims to train a policy that behaves differently when given different goals, for example, picking up different objects. While in sparse reward cases, the agent only receives a reward signal when a goal is achieved. This is motivated by the fact that providing task completion information is often easier and less biased than hand-designing a behaviour-specific reward function for most real-world robotic tasks [5].

We implement the four basic tasks (Fig. 1a) proposed in [1] using Pybullet and reproduce the performances achieved by the Deep Deterministic Policy Gradient (DDPG) algorithm with Hindsight Experience Replay (HER) [1,8].

In addition, we further propose a set of new tasks that focus on multi-step manipulations in longer horizon with sparse rewards (Fig. 1b). To improve readability, the original set of tasks is named 'single-step tasks' and the new set of tasks is named 'multi-step tasks'. The multi-step tasks are developed with the aim to inspire new learning algorithms that can handle tasks where the reward

[3] The source codes are available at https://github.com/IanYangChina/pybullet_multigoal_gym.

signals only appear near the end of the task horizon [5,18]. Beside the delayed rewards, these tasks also require multiple steps to complete, and some of the steps are strongly dependent. For example, a block cannot be placed into a chest unless the chest is opened. This characteristic requires a learning algorithm to reason about the relationships between steps.

To facilitate comparison in future research, we benchmarked the performances on the four multi-step tasks by training the aforementioned DDPG-HER agent [1] with a simple human prior-based curriculum. Potential research directions in this regard are also discussed. To sum up, our contributions in this article are:

- Reproducing the multi-goal robotic arm manipulation tasks [13] using Pybullet, making it freely accessible.
- Reproducing the Hindsight Experience Replay performances [1] on the Pybullet-based environments.
- Proposing a set of new environments for multi-goal multi-step long-horizon sparse reward robotic arm manipulations.
- Benchmarking the multi-step tasks and proposing future research opportunities.

The rest of this paper includes the details of the proposed environments and programming APIs (section 2); the reproduction results of the DDPG-HER agent on the single-step tasks, the benchmark results of the multi-step tasks and discussions of challenges and future research (section 3); and finally the conclusion (section 4).

2 Environment

2.1 Single-step tasks

As shown in Fig. 1a, the single-step tasks are:

- KukaReach, where the robot needs to move the gripper tip to a goal location.
- KukaPickAndPlace, where the robot needs to pick up the block and move it to a goal location[4].
- KukaPush, where the robot needs to push the block to a goal location on the table surface.
- KukaSlide, where the robot needs to push the cylinder bulk with a force such that the bulk slides to a goal location that is unreachable by the robot.

Different from the original environments, which use a Fetch robot, we use a Kuka IIWA 14 LBR robot arm equipped with a simple parallel jaw gripper. This does not affect training as only Cartesian space control (gripper movement

[4] In training, the PickAndPlace goals are generated either on the table surface or in the air, with even probability, as suggested by [1]

and finger width) are used in the original tasks. We plan to support more robot arms in the future.

In addition to the gripper frame control mode, our environments also support joint space control, which results in a 7 dimensional action space for the KukaReach and KukaPush tasks and an 8 dimensional one for the other two tasks (with one extra dimension for controlling the gripper finger width). Such a control mode has been largely ignored in most DRL-based manipulation works, possibly due to its high dimensionality. However, this control mode is important in scenarios that involve collision avoidance. A manipulation policy should not only consider end-effector control, but also learn to control each joint more explicitly when the surroundings are crowded by objects or other agents, e.g., humans. We leave the design of tasks for this specific direction to future work.

The tasks provide two reward functions. The dense reward function uses the negative Euclidean distance between the achieved and desired goals. The sparse reward function gives a reward of 0 when a goal is achieved and -1 everywhere else. We further provide RGB-D images as an optional observation representation. Users can easily define different camera view-points for rendering observations and goals.

Note that, we did not change the design of these four tasks, but reproduce them using a different physics engine. For more details of the task, such as the state and the action spaces, we refer the readers to the original paper [13]. The APIs and programming style are slightly different and are described in section 2.3.

2.2 Multi-step tasks

Fig. 1b visualises the four challenging multi-step tasks developed by the authors, aiming at sparse reward long-horizon manipulations. Briefly, they are:

- BlockRearrange, where the robot needs to push the blocks to random positions. Gripper fingers are blocked in this task.
- ChestPush, where the robot needs to first open the sliding door (in black colour) of the chest and then push the blocks into the chest. Gripper fingers are blocked in this task.
- ChestPickAndPlace, where the robot needs to first open the sliding door (in black colour) of the chest and then pick and drop the blocks into the chest.
- BlockStack, where the robot needs to stack the blocks into a tower in a given order that is randomly chosen.

These tasks require the robot to learn different combinations of behaviours and provide different numbers of step dependencies. For example, the BlockStack task has more dependent steps with the increase of the number of blocks to be stacked. The complexity of these tasks increases with more dependent steps and blocks, as shown in Table. 1. Moreover, the number of blocks involved in a task affects its task horizon, and thus its exploration difficulty. Detailed task information is provided in supplementary material section 1.

Table 1. Multi-step tasks summary

Task	Needed behaviours	Step dependency	Num. of blocks
BlockReaarange	pushing	0	2 to 5
ChestPush	pushing	1	1 to 5
ChestPickAndPlace	pushing, picking, dropping	1	1 to 5
BlockStack	pushing, picking, placing	≥ 2	2 to 5

With the challenge of sparse reward in mind, the extreme case of these tasks is that the environment only gives a task completion signal (e.g., a reward of 0) when the ultimate goal (e.g., all the blocks are stacked) is achieved, and provides a reward of -1 everywhere else. In this case, the task is extremely difficult for any naive reinforcement learning algorithm, even the one with hindsight experience replay (see section 3.2). This is because the reinforcement learning agent has an extremely low probability of seeing a meaningful reward value. Compared to the single-step tasks, which only feature the sparse reward problem in a short task horizon, these multi-step tasks can be used to investigate more difficult problems, such as

- How to explore efficiently for multi-step tasks with sparse and delayed rewards?
- How to represent and learn the dependencies among task steps?
- How can ideas such as curriculum learning, option discovery and hierarchical learning help in these tasks?

One possible research direction for these problems is to create a curriculum that provides the learning algorithm with goals starting from easy to difficult [11,12]. In this paper, we design a human-prior based curriculum for the multi-step tasks. It simply generates goals that require increasing time horizons to achieve, e.g., from stacking two blocks to five. However, the results show that such a simple curriculum is not efficient enough for longer horizon tasks (see section 3.2). To tackle these problems, more efficient methods need to be developed. Section 3.3 provides more discussion on future research opportunities.

2.3 APIs and Programming style

In OpenAI Gym, users create environment instances by specifying a unique task ID pre-registered in the package [2,13]. In contrary, we provide users with an API to make environments more intuitively. As shown in Code 1, the `make_env(...)` function provides arguments to setup a specific environment instance. Supplementary material section 2 provides a detailed explanation of these arguments. Currently, only eight tasks are prepared, including four single-step tasks and four multi-step tasks.

We provide an argument to activate image observations and goals, while the original Gym environment requires users to rewrite some of the code to achieve

Code 1 Create an environment instance

```
# Original OpenAI Gym style
import gym
env = gym.make("FetchReach-v0")
# Our style
import pybullet_multigoal_gym as pmg
env = pmg.make_env(
    # task args
    task='block_rearrange', joint_control=False, num_block=2, render=False,
    binary_reward=True, max_episode_steps=50, distance_threshold=0.05
    # image observation args
    image_observation=False, depth_image=False, goal_image=False,
    visualize_target=True,
    camera_setup=camera_setup, observation_cam_id=0, goal_cam_id=1,
    # curriculum args
    use_curriculum=True, num_goals_to_generate=1e6)

# Interaction loop
obs = env.reset()
while True:
    action = env.action_space.sample()
    obs, reward, done, info = env.step(action)
    if done:
        obs = env.reset()
```

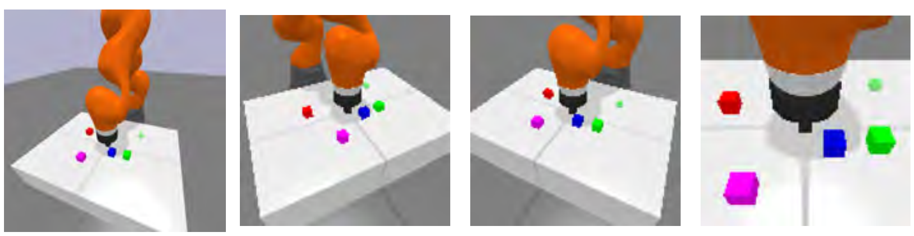
this. In addition, users can easily customise cameras for observation or goal images by defining a list of Python dictionaries and passing it to the `camera_setup` argument. An example is given in Code 2. Intuitively, the setup example defines two cameras, and in Code 1 they are used for capturing observation and goal images respectively, by setting the `cam_id` arguments to 0 and 1. Alternatively, users can pass -1 to the `cam_id` arguments, activating an on-hand camera looking at the gripper tip position. Fig. 2 shows a scene and three images rendered with the above-mentioned cameras.

Except for the codes that create an environment instance, other user APIs are kept the same as the original multi-goal Gym environment package. In our experiments, the code of training the DDPG-HER agent needs no change from Mujoco to Pybullet, and we successfully reproduce the performances as shown in section 3.1.

3 Benchmark and Discussion

In section 3.1, we reproduced the Hindsight Experience Replay (HER) [1] on the single-step tasks to demonstrate the success of the transfer from the Mujoco-based environments to ours. More specifically, we trained a DDPG agent using the 'future' goal-relabelling strategies, with the same hyperparameters and design

Code 2 A list of camera setup dictionary	Meaning
camera_setup = [	
{	
'cameraEyePosition': [-1.0, 0.25, 0.6],	the 3D coordinates of the camera frame in the world frame
'cameraTargetPosition': [-0.6, 0.05, 0.2],	the 3D coordinates which the camera looks at in the world frame
'render_width': 128,	the width of the rendered image
'render_height': 128	the height of the rendered image
},	
{	
'cameraEyePosition': [-1.0, -0.25, 0.6],	
'cameraTargetPosition': [-0.6, -0.05, 0.2],	
'render_width': 128,	
'render_height': 128	
}	
]	



(a) Scene (b) Camera 1 (c) Camera 2 (d) On-hand camera

Fig. 2. Images rendered using the two cameras defined in Code 2 and the built-in on-hand camera.

proposed in [1], except that we did not use distributed training. In addition, we also trained the same agent on the single-step tasks with joint control. Section 3.2 shows the results of training the DDPG-HER agent on the multi-step tasks. The results serve as a benchmark for future studies. Section 3.3 provides challenges and future research opportunities.

The Pytorch implementation of the algorithm is available here. The experiment scripts are available here. All experiments were run on Ubuntu 16.04 on a workstation with an Intel i7-8700 CPU and an Nvidia RTX-2080Ti GPU. All performance statistics are averaged from 4 runs with different random seeds.

3.1 Reproducing Hindsight Experience Replay on Single-step tasks

For comparison, we ran the same DDPG-HER algorithm [1] with the same hyperparameters on the Mujoco- and our Pybullet-based environments. As shown in Fig. 4, the agent achieved almost the same performances on the both environments [5]. These results demonstrate our successful transplantation of the single-step tasks onto the Pybullet engine. Running an episode of the Reach task in Mujoco took 0.079 ± 0.007 seconds, and in PMG, 0.272 ± 0.011 seconds (averaged over 100 episodes for 10 random seeds).

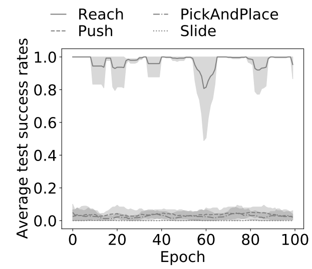
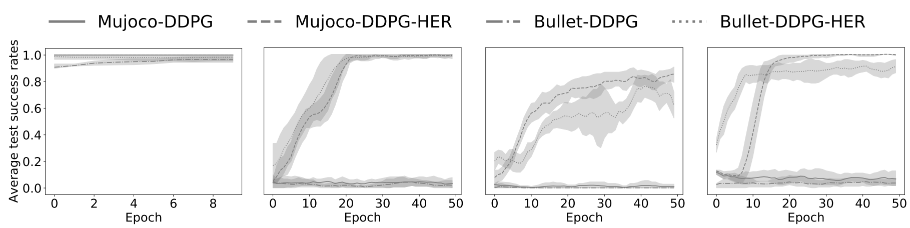


Fig. 3. DDPG-HER performances on Joint space control tasks.

Beside the original tasks, we also ran the experiments with joint space control using the same algorithm. These joint space control tasks differ from the original gripper frame control tasks in that the robot's actions are now joint commands, and the state representation further includes the current joint states. Results show that, in comparison to gripper frame control mode, single-step tasks under joint space control mode are harder to learn (Fig. 3). Its performance on the easiest Reach task also shows higher variance.



(a) Reach (b) PickAndPlace (c) Slide (d) Push

Fig. 4. Test success rates of the single-step tasks on Mujoco or Pybullet engine.

This is expected as the action space has higher dimensionality. On the other hand, the gripper is constrained to be pointing top-down under the gripper frame control mode, but this constraint is released under the joint space control mode. This makes the tasks harder to learn by increasing the size of its solution space.

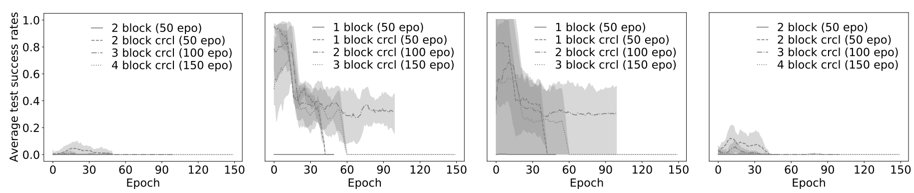
For future research, it is valuable to develop reinforcement learning algorithms that can handle such control tasks with higher action dimensionality and larger solution space, potentially from (depth-) image observations. Investigating harder tasks including collision avoidance and comparing with classic motion planning methods are interesting directions as well.

[5] Note that the Slide task is sensitive to the random seeds in both environments. The agent was unable to learn anything in some cases. It also exhibited higher variance than other tasks.

3.2 Benchmarking Multi-step tasks

This section discusses the performances of the DDPG-HER agent [1] on the multi-step tasks, with and without the use of the proposed simplistic curriculum (supplementary material section 3). We benchmarked the tasks without a chest using $2, 3, 4$ blocks, and the tasks with a chest using $1, 2, 3$ blocks.

We made one modification to the agent for these tasks. The action values predicted by the critic network are clipped within $[-50, 0]$ in the single-step tasks as suggested by [1], because the lowest value is -50 under sparse reward setting, given that the maximum episode timestep is 50 [13]. For the multi-step tasks, we changed the lower bound of the clipped value range to the negative maximum episode timestep for each task.



(a) BlockRearrange (b) ChestPush (c) ChestPick (d) BlockStack

Fig. 5. Test success rates of the multi-step tasks. 'crcl' means 'curriculum'.

As shown in Fig. 5, the DDPG-HER agent learned nothing without the help of the curriculum (blue line in each subplot). When aided by the curriculum, it could achieve the easiest steps (open the chest door) in the ChestPush and ChestPickAndPlace tasks, but failed at later harder steps (success rates quickly drop to near 0 as learning proceeds, shown by the orange, green and red lines). For the BlockRearrange and BlockStack tasks, the agent struggled to learn the easiest steps even with the help of the curriculum. This is because exploring to open the chest door is easier than moving a block around. These results indicate that these sparse rewards multi-step tasks are still unsolvable given the current state-of-the-art reinforcement learning algorithms.

3.3 Challenges and opportunities

This section discusses the challenges and future research opportunities related to the sparse reward multi-step robotic manipulation tasks from two perspectives, including exploration efficiency and representation learning. From each of them, there are several research directions that can be focused on.

Exploration: In sparse reward environments, improving exploration efficiency has long been a research challenge in the field of DRL [14]. However, current research has been restricted within toy problems (e.g., grid world) or the Atari games (e.g., Montezuma's Revenge). These are all 2D tasks with discrete action spaces. Robotic manipulations are tasks in a 3D world, with larger and richer observations and continuous action spaces. It would be valuable to evaluate techniques that work in the 2D tasks on our 3D and continuous action tasks, with the hope to improve them further and transfer to the real-world.

In the multi-goal setting, we have demonstrated the insufficiency of the HER aided by a simplistic goal generation curriculum. It is then potentially fruitful to develop a better curriculum for such tasks. Another interesting direction is to leverage task decomposition for multi-step tasks and make use of hierarchical learning systems [18]. The use of sub-goals is a promising way to tackle the hard exploration problem in such tasks.

Representation learning: Representation for RL agents, especially in sparse reward tasks, has been increasingly active recently. Different from supervised learning tasks, RL agents rely on the reward signals to learn a representation of the environment and the task altogether. This makes it hard to generate and maintain a good representation in sparse reward tasks, in which the representation learnt can easily collapse. Again, current state-of-the-art in this direction has been largely restricted within 2D tasks or tasks with short horizon [7,10], and our environment is a promising testbed for evaluating and improving them in a 3D world with longer task horizons.

4 Conclusion

We propose an open-source robotic manipulation simulation software implementation for multi-goal multi-step deep reinforcement learning. The implementation of the OpenAI multi-goalstyled environment (based on the Mujoco engine) has been achieved using Pybullet. Performance of the popular DDPG-HER algorithm has been reproduced in our work (section 3.1). Except for the original manipulation tasks, named single-step tasks, we designed a set of multi-step tasks with sparse rewards in longer task horizons. We benchmarked the performances of the DDPG-HER agent with and without the use of a simplistic goal generation curriculum (section 3.2), demonstrating the inability of the state-of-the-art algorithms to learn in such long horizon and sparse reward environments. Finally, we provided brief discussions of the challenges and future research opportunities, including EXPLORATION and REPRESENTATION LEARNING in sparse reward reinforcement learning. Our future research will focus on developing sub-goal-based solutions to tackle such multi-step sparse reward robotic manipulation tasks.

References

1. Andrychowicz, M., Wolski, F., Ray, A., Schneider, J., Fong, R., Welinder, P., McGrew, B., Tobin, J., Abbeel, O.P., Zaremba, W.: Hindsight experience replay. In: NIPS. pp. 5048–5058 (2017)
2. Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., Zaremba, W.: Openai gym. arXiv preprint arXiv:1606.01540 (2016)

3. Coumans, E., Bai, Y.: Pybullet, a python module for physics simulation for games, robotics and machine learning. http://pybullet.org (2016–2019)
4. Delhaisse, B., Rozo, L., Caldwell, D.G.: Pyrobolearn: A python framework for robot learning practitioners. In: Conference on Robot Learning. pp. 1348–1358. PMLR (2020)
5. Dulac-Arnold, G., Mankowitz, D., Hester, T.: Challenges of real-world reinforcement learning. ICML (2019)
6. Erez, T., Tassa, Y., Todorov, E.: Simulation tools for model-based robotics: Comparison of bullet, havok, mujoco, ode and physx. In: 2015 IEEE international conference on robotics and automation (ICRA). pp. 4397–4404. IEEE (2015)
7. Laskin, M., Srinivas, A., Abbeel, P.: Curl: Contrastive unsupervised representations for reinforcement learning. In: International Conference on Machine Learning. pp. 5639–5650. PMLR (2020)
8. Lillicrap, T.P., Hunt, J.J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., Wierstra, D.: Continuous control with deep reinforcement learning. ICLR (2016)
9. LLC, R.: Mujoco: advanced physics simulation. http://www.mujoco.org/ (2018)
10. Lyle, C., Rowland, M., Ostrovski, G., Dabney, W.: On the effect of auxiliary tasks on representation dynamics. arXiv preprint arXiv:2102.13089 (2021)
11. Manela, B., Biess, A.: Curriculum learning with hindsight experience replay for sequential object manipulation tasks. arXiv preprint arXiv:2008.09377 (2020)
12. Narvekar, S., Peng, B., Leonetti, M., Sinapov, J., Taylor, M.E., Stone, P.: Curriculum learning for reinforcement learning domains: A framework and survey. Journal of Machine Learning Research **21**(181), 1–50 (2020)
13. Plappert, M., Andrychowicz, M., Ray, A., McGrew, B., Baker, B., Powell, G., Schneider, J., Tobin, J., Chociej, M., Welinder, P., et al.: Multi-goal reinforcement learning: Challenging robotics environments and request for research. arXiv preprint arXiv:1802.09464 (2018)
14. Rashid, T., Peng, B., Boehmer, W., Whiteson, S.: Optimistic exploration even with a pessimistic initialisation. ICLR (2020)
15. Shen*, B., Xia*, F., Li*, C., Martín-Martín*, R., Fan, L., Wang, G., Buch, S., D'Arpino, C., Srivastava, S., Tchapmi, L.P., Vainio, K., Fei-Fei, L., Savarese, S.: igibson, a simulation environment for interactive tasks in large realistic scenes. arXiv preprint arXiv:2012.02924 (2020)
16. Tassa, Y., Doron, Y., Muldal, A., Erez, T., Li, Y., Casas, D.d.L., Budden, D., Abdolmaleki, A., Merel, J., Lefrancq, A., et al.: Deepmind control suite. arXiv preprint arXiv:1801.00690 (2018)
17. Todorov, E., Erez, T., Tassa, Y.: Mujoco: A physics engine for model-based control. In: 2012 IEEE/RSJ International Conference on Intelligent Robots and Systems. pp. 5026–5033. IEEE (2012)
18. Yang, X., Ji, Z., Wu, J., Lai, Y.K., Wei, C., Liu, G., Setchi, R.: Hierarchical reinforcement learning with universal policies for multistep robotic manipulation. IEEE Transactions on Neural Networks and Learning Systems (2021)