Noname manuscript No.
(will be inserted by the editor)

Object Rearrangement in Clutter for Mobile Manipulator Using Hybrid Soft Actor-Critic Method

Jinhwi Lee · SeungHyun Kang · Changhwan Kim

Received: date / Accepted: date

Abstract This paper presents a reinforcement learning model to solve the object rearrangement task and motion planning (OR-TAMP) problem for a mobile manipulator in a cluttered environment. The objects that obstruct a target object are necessarily removed to grasp the target in such cluttered environments as tables, refrigerators, and shelves. In the environments, a robot needs to determine which object to move and where to relocate it. This problem can be formulated in a discrete space (task planning) and a continuous space (motion planning for a robot arm and a mobile). One of the reinforcement learning methods, *hybrid Soft Actor-Critic (hybrid SAC)*, is employed to deal with the problem in the two different spaces and expect good efficiency and convergence. We define the state, action, and reward function for the learning model of OR-TAMP problem. In the simulation results, the learned model runs in milliseconds and shows a success rate from 66% to 92% with the various numbers of objects. In addition, some of the results are examined in real robot experiments.

Keywords Object rearrangement · Task and motion planning · Mobile manipulation · Reinforcement learning

1 INTRODUCTION

Bringing an object to people is an essential service in robot manipulation. A robot is often requested to plan a task of obstacle rearrangement and execute it to grasp a target object when objects stand in narrow spaces, such as cupboards, refrigerators, and shelves. Figure 1 shows an example environment with objects on a shelf and a mobile manipulator. The robot system has a multibody system as a combination of a manipulator with multiple DoF, a mobile base, and a vision system. In such environment, objects could be grasped from the side or front because the boards or walls obstruct the manipulator from grasping from the top. Especially when objects stand around a target object in the environment, a robot needs to move them aside and grasp the target object. In this kind of dense environment, a task and motion planner should determine which objects to move and where to relocate them before grasping a target.

The obstacle rearrangement task and motion planning (OR-TAMP) problem has two distinct characteristics: determining which objects among the various objects to move and where to relocate them by picking and placing them. The former becomes a discrete problem, and the latter determines the relocation positions in the continuous workspace of a manipulator. Most algorithms separate these characteristics, and solve one problem and another sequentially. If the two sub-problems are individually considered, the

Jinhwi Lee
Korea Institute of Science and Technology, Seoul, Korea
Department of Mechanical Engineering, Hanyang University, Seoul, Korea
E-mail: jinhooi@kist.re.kr

SeungHyun Kang
Hyundai motor group, Uiwang, Korea
E-mail: 7328117@hyundai.com

Changhwan Kim
Korea Institute of Science and Technology, Seoul, Korea
E-mail: ckim@kist.re.kr

Fig. 1: An example of cluttered environment. Various objects are placed on the shelf. Some objects need to be removed to grasp a target object without collisions against the other objects and the boards.

OR-TAMP problem may become more complicated as the solutions of two sub-problems affect each other. Therefore, it could be of more use to solve the problems simultaneously.

The problem of determining which objects to move deals with only the objects that obstruct a target object, called obstacles. The objects that do not block the target may not affect the task of grasping the target. In other words, if an object rather than an obstacle is selected, it may take more time to obtain the solution due to the unnecessary actions. If the planner places an object rather than an obstacle at a wrong spot, it may increase the complexity of the entire task.

It is more challenging to determine where to relocate obstacles in the OR-TAMP problem. This problem aims to search not only empty positions in the workspace but accessible ones by a robot manipulator. The robot manipulator should be able to approach the positions for putting obstacles. For this, some algorithms use the limited numbers of sampled positions and verify their feasibility. In our work, a learning skill is used to obtain the feasible positions in the workspace instead of sampling.

In this work, we formulate the OR-TAMP problem and construct a single model for reinforcement learning, which simultaneously considers the two sub-problems of which obstacles to move and where to relocate them. We conduct virtual and real-world robot experiments to show the validity of the learned model.

2 RELATED WORK

Reinforcement learning has been used widely to learn pick-and-place skills but may have difficulties to deal with complex high-level planning problems such as the OR-TAMP problem. Some algorithms in [7, 11, 12] apply reinforcement learning to robotic manipulation problems focusing on the control of manipulator

movement. These algorithms may have limits to learn various and dynamic environments, since small changes in environments could make the model learn the changes additionally.

Several previous studies attempt to solve the OR-TAMP problems. Lee et al. [1] develop an algorithm to decide which obstacles to remove using the modified vector field histogram plus (modified VFH+). The modified VFH+ measures the angle range not to block an object and finds which objects to remove based on it. However, finding relocation positions is not dealt with in the algorithm. Nam et al. [2] propose a graph based algorithm. The algorithm formulates geometrical relations in objects into a graph and obtains the objects obstructing a target object using graph search. However, these two algorithms do not consider the relocation problem in a cluttered environment.

Cheong et al. [3] solve the relocation problem when the order of removing obstacles is known. The algorithm samples empty spaces in an environment and checks whether the relocation position is valid in the aspect of kinematics of a robotic manipulator and blocks other objects. Lee et al. [4] and Ahn et al. [5] consider the two problems of what to move and where to relocate it. Lee et al. extend their previous graph search algorithm in [1] to obtain a global solution for the two problems. The algorithm includes non-prehensile manipulation skills to search for more possible objects to move in the sense of kinematics. Ahn et al. [5] also extend their previous graph search algorithm [2] to solve the relocation problem of obstacles. Their algorithms generate the relocation positions and add them to the geometrical graph of objects to obtain a local solution for what to move and where to relocate it.

Cheong et al. [6] propose the DORE algorithm based on a Deep Q Network (DQN) to solve the OR-TAMP problem. The DORE algorithm uses the sequentially connected two DQN models to solve the two sub-problems. The agent in DQN may have a difficulty in considering continuous action spaces to find relocation positions in an environment (i.e. table, shelf) so that the DORE algorithm transforms a continuous action and state space into a discrete grid map. The transformation could cause difficulties in applying the DORE algorithm to a real-world environment because of scalability and kinematical feasibility.

Most of the search based algorithms mentioned above consider the two problems of what obstacles to move and where to relocate them separately. This kind of approach has mutual effects on the solution of each sub-problem so that it generates more search configurations, which could cause more planning time. We apply reinforcement learning to consider the two sub-problems in the OR-TAMP problem. Unlike the DORE algorithm above, our proposed learning model considers discrete and continuous spaces in a single model to reduce learning time and training episodes.

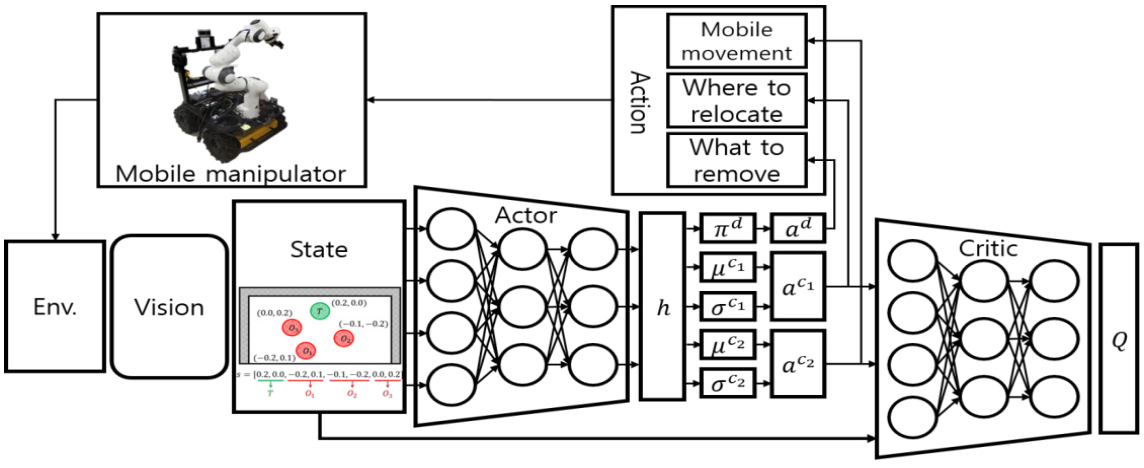


Fig. 2: The structure of proposed method. The actor shares the hidden layers h to return one discrete distribution π^d, the means $\mu^{c_{1,2}}$ and the standard deviations $\sigma^{c_{1,2}}$ of the continuous component. The discrete action a^d (which obstacles to move in the OR-TAMP problem) is sampled from π^d. The continuous actions, a^{c_1} (where to relocate the obstalces) and a^{c_2} (mobile movements), are computed from their mean and standard deviation vectors by using standard normal noise and a tanh non-linearity. The critic takes continuous action results from the actor and the state representation. More details on hybrid SAC are referred into [8].

3 PROBLEM DESCRIPTION

Multiple objects, including a target object, make an environment cluttered so that some objects may become the obstacles for grasping the target. The task , requested by a user, of carrying the target object considers how to grasp the target with obstacle rearrangement. This OR-TAMP problem requires determining which obstacles should be moved and where they are relocated in the workspace. The problem also considers the motion planning of a mobile manipulator for kinematical feasibility. In order to solve the OR-TAMP problem with motion planning, this section describes a reinforcement learning based algorithm and its formulation.

3.1 Reinforcement learning based algorithm: hybrid SAC

We employ one of the actor-critic methods, *hybrid Soft Actor-Critic (hybrid SAC)* in [8], as it deals with a continuous action space by learning the policy itself, not calculating the state or $Q-$ value function. On the other hand, such a value-based method as DQN evaluates the state or $Q-$ value function using the Bellman equation, which is hard to calculate in the continuous action space.

In the present problem, we apply the hyper SAC to simultaneously consider the two characteristics of the OR-TAMP problem. As shown in Fig. 2, the actor returns the mean and the standard deviation from a state s identical to the standard SAC algorithm in [10]. The actor receives the state s as input through a vision process and returns the continuous values that determine where to relocate obstacles. The actor also returns a discrete value by sharing the hidden layer, which is different from the actor in the standard SAC. The discrete value of policy ($\pi(a^d|s)$) denotes which obstacle needs to be relocated. The critic takes only the continuous values to calculate the Q-values, besides the discrete value is optimized by minimizing the KL divergence [8]. Choosing which obstacle to move is independent of determining where to relocate them. The entropy bonus used in the target Q-value for the critic network Q is computed as Eq. 1, which implies the joint entropy from the discrete part ($\pi(a^d|s)$) and continuous part ($\pi(a^c|s, a^d)$) of the policy.

$$\alpha^d H(\pi(a^d|s)) + \alpha^c \sum_{a^d} \pi(a^d|s)\mathcal{H}(\pi(a^c|s, a^d)) \tag{1}$$

, where α^d and α^c are the hyperparameters to encourage exploration for discrete and continuous actions respectively. It is noticed that the continuous action a^c is composed of the two sub-actions, a^{c_1} (mobile movement) and a^{c_2} (where to relocate the obstacle).

3.2 Problem formulation

The agent of hybrid SAC takes the actions in various episodes and gets the rewards to learn how to grasp a target object. We suggest the state, action, and reward function for the hybrid SAC.

3.2.1 State & numbering of objects

The agent uses objects' positions as normalized values as a state s in the OR-TAMP problem. In this work, we assume that there is only one target object and all the objects are identical in shape and size. The object number is given from the closest object to a robot to the outermost, which is called numbering herein. The state s is composed of the position of target object and those of the other objects (except the target) in order. The number of objects N affects the state and input shape. Figure 3 shows an example of the state with one target object and three objects.

Once the agent conducts an action, i.e. relocating an obstacle, the object configuration is changed. This causes a change in the order of objects in the aspect of the shortest distance to the robot as well. Based on this rule, the object number needs to be updated. This numbering rule is able to keep the same values of state to represent the identical object configurations. Figure 4 shows the effect of re-numbering the objects. The first column represents an example of the learned object configurations. In the first and second steps, the agent plans to grasp the target by relocating the obstacles in the order of O_1 and O_2. Unlike the first column, the second column shows the same object configuration without following the proposed numbering rule. In the second column, the agent attempts to relocate O_1 first and then O_2 as the agent learned the same configuration before in the training. However, the obstacles of O_1 and O_2 are not the same as those in

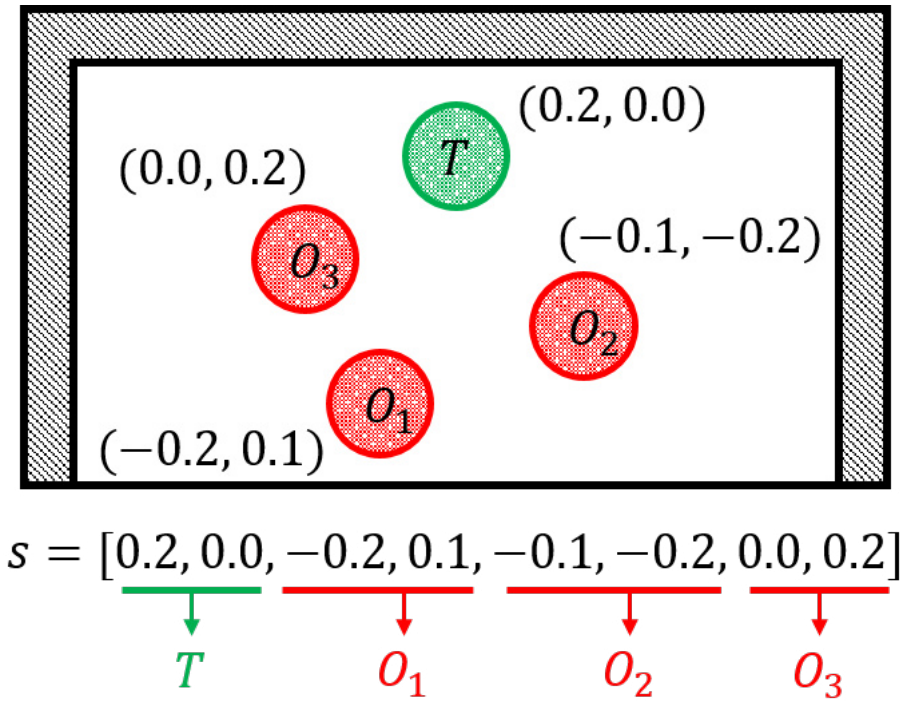


Fig. 3: An example of the state. Objects in red stand around a target object in green. The object number is given from the closest object to a robot (on the bottom) to the outermost as O_1, O_2, and O_3. The state s is composed of the position of target object and those of the other objects in order.

the first column, which yields a different configuration from the second step in the first column. The agent at the first step of the second column cannot relocate the obstacle O_2, because it is blocked by the obstacle O_3. The numbering of objects needs to be given with a certain fixed rule, which is herein the order of the shortest distance to a robot.

3.2.2 Action

The agent deals with a combined action space to consider which obstacle to move, where to relocate it, and where to move the mobile base simultaneously.

$$A_t = \{a_t | a_t = [a^d, a^{c_1}, a^{c_2}]\} \tag{2}$$

, where $a^d = 0, ..., (N-1)$. $a^d = 0$ means the action of grasping the target object and $a^d = 1, ..., (N-1)$ means an obstacle number among the $N-1$ objects except the target. a^{c_1} denotes the mobile movement and a^{c_2} denotes the relocation position for an obstacle. a^d has a discrete value to denote which obstacle to move. The second component of the action a^{c_1} has two continuous values, which denotes the mobile base movement. We assume the mobile base can move forward and backward in this work. The first value of the a^{c_1} means the movement of the mobile base to approach the shelf for grasping. The second means the movement of the mobile base to approach the shelf for relocating. The a^{c_2} also has two continuous values, which denote the $x-, y-$ coordinates for the relocation in the manipulator's workspace.

3.2.3 Reward function & termination conditions

We define a suitable reward function for the OR-TAMP problem. The agent gets a positive reward when the robotic manipulator grasps the target object. This means that the agent confirms the motion planning results for grasping the target object, and then the episode ends. The agent gets a negative reward when it exceeds the maximum action count M_a, which plays as another termination condition. M_a denotes the maximum number of actions allowed within a single episode. We bound M_a; otherwise, the agent may attempt to rearrange the same obstacles, repeatedly. In this work, we set M_a same as the number of

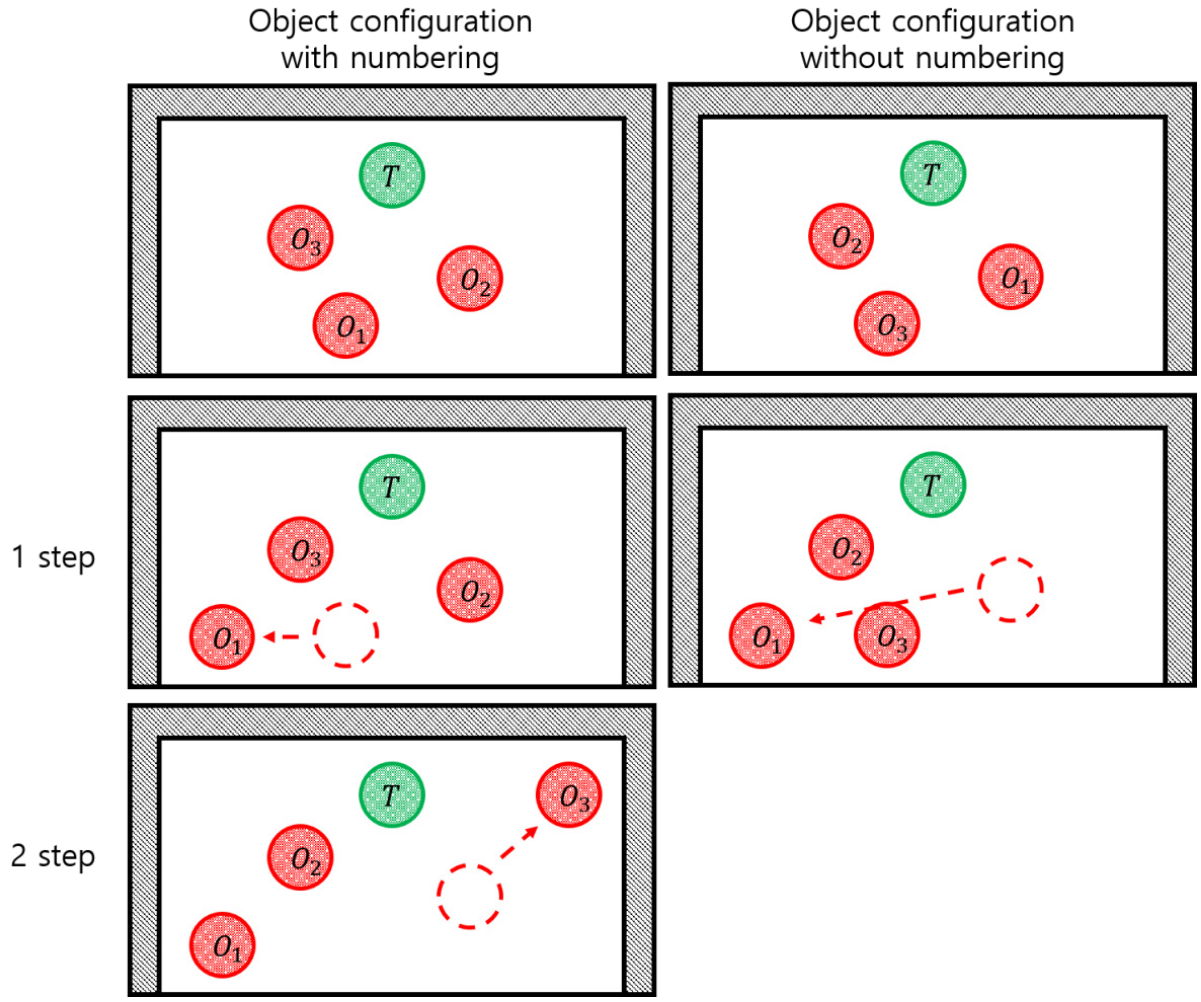


Fig. 4: Comparison of results with and without numbering. The first column represents an example of the object configuration with the proposed numbering rule and the second column shows the same object configuration without using the numbering rule.

Table 1: Comparison of the simulation results between the proposed method and the baseline method for the OR-TAMP problem with N=5, N=10, and N=15. Each of the three cases is simulated 50 times with 50 different object configurations. All the numbers except the success rate show the mean and standard deviation (in parentheses).

Measure	N=5		N=10		N=15	
	Proposed	Baseline	Proposed	Baseline	Proposed	Baseline
Success rate (%)	92	66	86	36	66	4
Planning time (ms)	1.35 (0.62)	14.44 (36.34)	2.02 (1.48)	127.15 (429.12)	2.83 (1.99)	69.97 (82.96)
Number of action attempts	1.52 (1.15)	2.36 (1.91)	2.86 (3.08)	6.76 (4.36)	6.78 (6.04)	14.44 (2.77)

objects N. The agent gets another negative reward for the action of rearranging obstacles. This negative reward leads the agent to take the minimum actions to end the episode early. In this second reward, the different negative values are assigned to the four types of failures as follows: (i) failure in mobile movement for grasping, (ii) failure in grasp an obstacle, (iii) failure in the mobile movement for relocating, and (iv) failure in relocating the obstacle. These actions are sequentially connected. For example, the failure of action (i) occurs, and the remaining actions for grasping and relocating are no longer needed. Due to this, the larger magnitudes of negative values are given to the failure cases in the order of (i),(ii), (iii), and (iv). For the total reward function, the agent combines each of the positive and negative rewards and learns the episodes to increase the total return.

Fig. 5: The results of learning for the OR-TAMP problem. The x-axis of each graph represents a relative step. The models with N=5, N=10 and N=15 stop learning at 40K, 60K and 100K steps, respectively.

4 EXPERIMENTS

First, this section shows the reinforcement learning results according to the several numbers of objects N. Secondly, we compare the simulation results of the proposed method with those of the method suggested in [1] as a baseline. Since the baseline method does not consider the mobile movement, we assume the mobile base is located in front of the shelves for comparison. We compare both methods in terms of success rate, planning time, and the number of action attempts for the OR-TAMP problem. We make the learning and simulation environments by using OpenAI Gym [13]. Finally, we implement the proposed method in a real robot environment for validation to solve an example of the OR-TAMP problem. We test simulation and real robot experiments with the 7-DOF manipulator (Franka Emika Panda) with the two-finger gripper (Robotiq). We use MoveIt [14] for motion planning. The reinforcement learning and simulations are performed using Intel Core i9-9900K 3.6GHz with 32G RAM and GeForce RTX 2080 super.

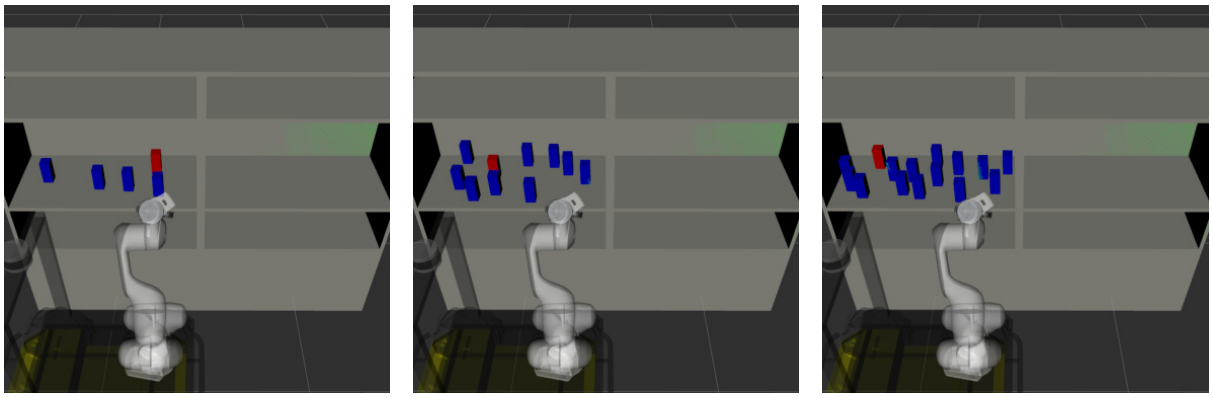
4.1 Learning results

We set three different environments with $N = 5, 10, 15$ to learn the model. We use the same hyperparameters of the model in all three different environments. The only difference is the input shape, which is the number of objects N. In all the environments, only one target object exists, and the remaining objects could be obstacles to blocking the target or not. The positions of objects are randomly generated in the workspace. The objects are not overlapped with each other physically.

As shown in the first of Fig. 5, the episode lengths converge stably with the reasonable numbers for the cases of $N = 5, 10, 15$. These episode lengths are observed under the maximum action count M_a. The value of M_a is given the same as the number of objects in the experiments. For example, the episode length of the case of $N = 5$ should be less than five, as seen in the figure (See Sec. III-B-3). The converged values may not guarantee the globally minimum number of action attempts of the OR-TAMP problem but could generate the sub-optimal number of action attempts. In other words, unnecessary and meaningless actions might be generated to grasp the target. As shown in the second of Fig.5, the episode rewards also converge with large total returns, and the agent could learn more successful actions to rearrange obstacles (See Section III-B-3). As shown in the third of Fig. 5, the success rates also converge at high values. The success rate is computed only when the target object is successfully grasped for the last ten episodes. When the number of objects increases in a fixed workspace, more actions for rearrangement are attempted, which may increase the possibility of action failure so that less rewards and success rates are obtained, as seen in the second and third of the figure.

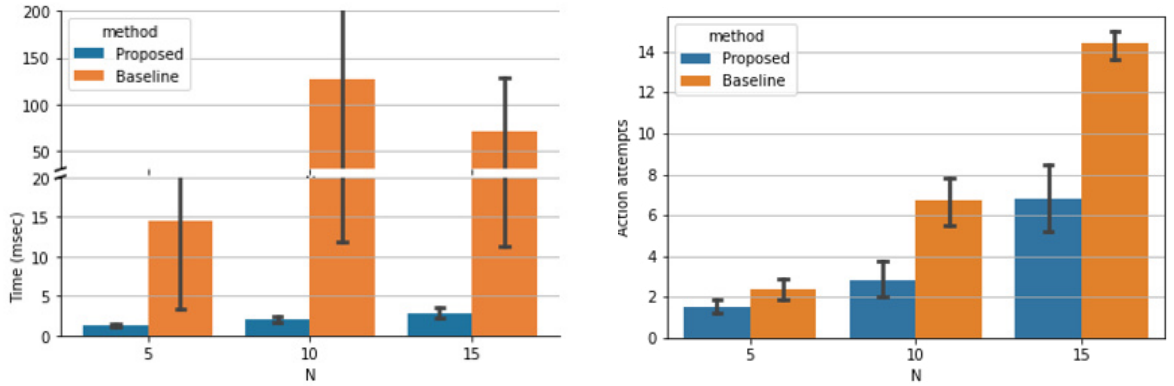
Table 2: The number of obstacles relocated by the proposed method only in the successful episodes from the simulation. All the numbers show the mean and standard deviation (in parentheses).

Measure	N=5	N=10	N=15
# of relocated obstacles	1.22 (0.51)	1.70 (1.08)	2.55 (1.20)



(a) N=5 (b) N=10 (c) N=15

Fig. 6: The simulation environments. MoveIt is used for motion planning and kinematics. A target object (red) and the objects (blue).



(a) Planning time (msec) when the episode is successful. (b) The Number of action attempts until the episode ends.

Fig. 7: Simulation results of planning time and action attempts.

4.2 Simulation results

A gym environment used in a simulation experiment is the same as one used in learning. We compare the proposed method with the baseline method. For comparing the two methods, the same object configurations are used, and the outputs from both methods are the actions of rearrangement (i.e. which obstacles to move and where to relocate them). We perform the experiments with the 50 random instances for each of $N = 5, 10, 15$. The results are shown in Tables 1 and 2 and Fig. 7. As shown in Fig. 7a, the proposed method returns a plan in a few milliseconds, which is much faster than the baseline method. Even if the number of objects increases, the proposed method takes less time to return a solution. The total planning time slightly increases when the number of objects N increases. This is due to more obstacles that the agent needs to rearrange.

Furthermore, as shown in Tables 1 and 2, the proposed method relocates fewer obstacles and shows higher success rates than the baseline method. Especially for a very dense environment due to many objects in a fixed workspace, the baseline method may often fail to find feasible relocation positions and it couldn't grasp the target object finally. This is because the sampling method in the baseline method shows low feasibility to generate relocation positions in dense environment.

4.3 Real robot experiments

We set a real robot environment as shown in Fig. 9. The mobile manipulator, which consists of a non-holonomic mobile robot (Husky UGV) and a 7-DOF manipulator (Franka Emika Panda) mounted on it,

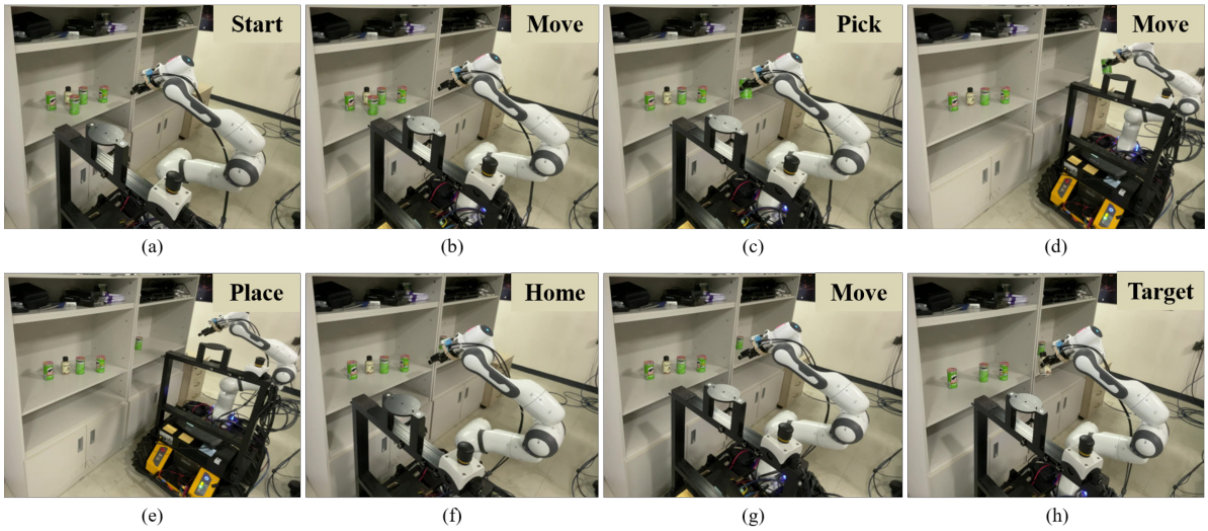
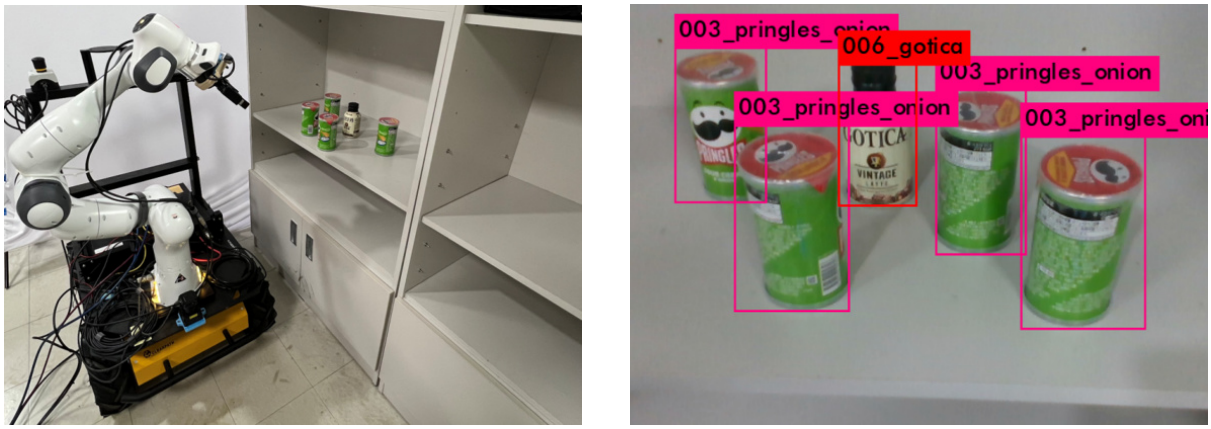


Fig. 8: The action sequence from the proposed method in a real robot experiment. The robot starts in front of the shelf as (a). The robot follows the action sequence as (b) mobile movement for grasp the obstacle, (c) grasping the obstacle, (d) mobile movement for relocating the obstacle, (e) relocating the obstacle, (f) mobile movement to home, (g) mobile movement for grasping the target, and (h) grasping the target.

is used in the experiments. A hand-eye calibration depth camera (Realsense D435) is attached on the wrist of the manipulator. We use YOLO-v3 [9] to detect objects and compute the grasping points. The mobile manipulator initially stands in front of the shelves to detect objects. We assume that the mobile manipulator can move parallel to the shelves. It can move forward and backward for grasping and placing obstacles.

In the experiment, as seen in Fig. 9b, the real robot recognizes four objects (cans of pringles in green) and one target (a bottle of coffee in white), generates the object configuration, and makes the state. The agent puts the state as input to the learned model of hybrid SAC and selects the action for rearrangement. The robot checks the feasibility of motion planning results corresponding to the action from the learned model and executes the motions if there are no troubles. Several snapshots of the experiment are shown in Fig. 8. It takes 195 seconds to grasp the target object after relocating one obstacle, which includes the time for mobile movements. The agent takes 1.27 milliseconds to plan each action of rearranging one obstacle and grasping the target and uses the remaining time for generating feasible motions and executing them in the real robot.



(a) An example of real robot experimental configurations.

(b) Object recognition results in the real robot experiment.

Fig. 9: Object recognition uses a hand-eye calibration camera attached on the robot's wrist. The object recognition result is used as the input of the proposed model.

5 CONCLUSIONS

We propose a reinforcement learning model based on hybrid SAC to deal with the two characteristics of the OR-TAMP problem: One is to determine which obstacles to move among multiple objects, which is formulated in a discrete space, and another is to obtain where to relocate the obstacles in the continuous workspace. We define the state and combine the discrete and continuous actions in a single model. Several types of failure cases are defined to specify the sequentially connected actions for rearrangement. Different reward values are assigned to each failure case by considering the action sequence. From the simulations, the proposed method converges reasonably with various object configurations. The method works faster than the baseline and plans more feasible actions. We observe that the method attempts more actions for rearrangement in denser object configurations as expected. The actions planned by the method are executed using a real robot within acceptable runtime. Further improvements may be achieved by considering the full mobility of the mobile base with non-holonomic or holonomic systems. The flexibility of the number of objects should be investigated in detail.

Acknowledgements This work was supported by the Technology Innovation Program and Industrial Strategic Technology Development Program (20018256, Development of service robot technologies for cleaning a table)

References

1. Lee, Jinhwi and Cho, Younggil and Nam, Changjoo and Park, Jonghyeon and Kim, Changhwan, Efficient obstacle rearrangement for object manipulation tasks in cluttered environments, International Conference on Robotics and Automation (ICRA), 183–189, 2019.
2. Nam, Changjoo and Lee, Jinhwi and Cheong, Sang Hun and Cho, Brian Y and Kim, ChangHwan, Fast and resilient manipulation planning for target retrieval in clutter, IEEE International Conference on Robotics and Automation (ICRA), 3777–3783, 2020.
3. Cheong, Sang Hun and Cho, Brian Y and Lee, Jinhwi and Kim, ChangHwan and Nam, Changjoo, Where to relocate?: Object rearrangement inside cluttered and confined environments for robotic manipulation, IEEE International Conference on Robotics and Automation (ICRA), 7791–7797, 2020.
4. Lee, Jinhwi and Nam, Changjoo and Park, Jong Hyeon and Kim, ChangHwan, Tree Search-based Task and Motion Planning with Prehensile and Non-prehensile Manipulation for Obstacle Rearrangement in Clutter, IEEE International Conference on Robotics and Automation (ICRA), 8516–8522, 2021.
5. Ahn, Jeeho and Lee, Jaeho and Cheong, Sang Hun and Kim, ChangHwan and Nam, Changjoo, An integrated approach for determining objects to be relocated and their goal positions inside clutter for object retrieval, IEEE International Conference on Robotics and Automation (ICRA), 6408–6414, 2021.
6. Cheong, Sanghun and Cho, Brian Y and Lee, Jinhwi and Lee, Jeongho and Kim, Dong Hwan and Nam, Changjoo and Kim, Chang-hwan and Park, Sung-kee, Obstacle rearrangement for robotic mafnipulation in clutter using a deep Q-network, Intelligent Service Robotics, 14, 4, 549–561, 2021.
7. Yuan, Weihao and Hang, Kaiyu and Kragic, Danica and Wang, Michael Y and Stork, Johannes A, End-to-end non-prehensile rearrangement with deep reinforcement learning and simulation-to-reality transfer, Robotics and Autonomous Systems, 119, 119–134, 2019.
8. Delalleau, Olivier and Peter, Maxim and Alonso, Eloi and Logut, Adrien, Discrete and continuous action representation for practical rl in video games, arXiv preprint arXiv:1912.11077, 2019.
9. Redmon, Joseph and Farhadi, Ali, Yolov3: An incremental improvement, arXiv preprint arXiv:1804.02767, 2018.
10. Haarnoja, Tuomas and Zhou, Aurick and Abbeel, Pieter and Levine, Sergey, Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor, International conference on machine learning, 1861–1870, 2018.
11. Andrychowicz, Marcin and Wolski, Filip and Ray, Alex and Schneider, Jonas and Fong, Rachel and Welinder, Peter and McGrew, Bob and Tobin, Josh and Pieter Abbeel, OpenAI and Zaremba, Wojciech, Hindsight experience replay, Advances in neural information processing systems, 30, 2017.
12. Schulman, John and Levine, Sergey and Abbeel, Pieter and Jordan, Michael and Moritz, Philipp, Trust region policy optimization, International conference on machine learning, 1889–1897, 2015.
13. Brockman, Greg and Cheung, Vicki and Pettersson, Ludwig and Schneider, Jonas and Schulman, John and Tang, Jie and Zaremba, Wojciech, Openai gym, arXiv preprint arXiv:1606.01540, 2016.
14. Coleman, David and Sucan, Ioan and Chitta, Sachin and Correll, Nikolaus, Reducing the barrier to entry of complex robotic software: a moveit! case study, arXiv preprint arXiv:1404.3785, 2014.
15. Sucan, Ioan A and Moll, Mark and Kavraki, Lydia E, The open motion planning library, IEEE Robotics & Automation Magazine, 19, 4, 72–82, 2012.
16. Beeson, Patrick and Ames, Barrett, TRAC-IK: An open-source library for improved solving of generic inverse kinematics, IEEE-RAS 15th International Conference on Humanoid Robots (Humanoids), 928–935, 2015.
17. Kuffner, James J and LaValle, Steven M, RRT-connect: An efficient approach to single-query path planning, IEEE International Conference on Robotics and Automation (ICRA), 995–1001, 2000.
18. Lobbezoo, Andrew and Qian, Yanjun and Kwon, Hyock-Ju, Reinforcement Learning for Pick and Place Operations in Robotics: A Survey, Robotics, 10, 3, 105, 2021.
19. Marzari, Luca and Pore, Ameya and Dall'Alba, Diego and Aragon-Camarasa, Gerardo and Farinelli, Alessandro and Fiorini, Paolo, Towards Hierarchical Task Decomposition using Deep Reinforcement Learning for Pick and Place Subtasks, 20th International Conference on Advanced Robotics (ICAR), 640–645, 2021.

20. Gualtieri, Marcus and ten Pas, Andreas and Platt, Robert, Pick and place without geometric object models, IEEE international conference on robotics and automation (ICRA), 7433–7440, 2018.
21. Sehgal, Adarsh and La, Hung and Louis, Sushil and Nguyen, Hai, Deep reinforcement learning using genetic algorithm for parameter optimization, IEEE International Conference on Robotic Computing (IRC), 596–601, 2019.
22. Deng, Yuhong and Guo, Xiaofeng and Wei, Yixuan and Lu, Kai and Fang, Bin and Guo, Di and Liu, Huaping and Sun, Fuchun, Deep reinforcement learning for robotic pushing and picking in cluttered environment, IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 619–626, 2019.