Object-Oriented Option Framework for Robotics Manipulation in Clutter

Jing-Cheng Pang[1,3,*], Si-Hang Yang[1,3,*], Xiong-Hui Chen[1,3], Xinyu Yang[1,3], Yang Yu[1,3], Mas Ma[2], Ziqi Guo[2], Howard Yang[2] and Bill Huang[2]

***Abstract*— Domestic service robots are becoming increasingly popular due to their ability to help people with household tasks. These robots often encounter the challenge of manipulating objects in cluttered environments (MoC), which is difficult due to the complexity of effective planning and control. Previous solutions involved designing specific action primitives and planning paradigms. However, the pre-coded action primitives can limit the agility and task-solving scope of robots. In this paper, we propose a general approach for MoC called the Object-Oriented Option Framework (O3F), which uses the option framework (OF) to learn planning and control. The standard OF discovers options from scratch based on reinforcement learning, which can lead to collapsed options and hurt learning. To address this limitation, O3F introduces the concept of an object-oriented option space for OF, which focuses specifically on object movement and overcomes the challenges associated with collapsed options. Based on this, we train an object-oriented option planner to determine the option to execute and a universal object-oriented option executor to complete the option. Simulation experiments on the Ginger XR1 robot and robot arm show that O3F is generally applicable to various types of robot and manipulation tasks. Furthermore, O3F achieves success rates of 72.4% and 90% in grasping and object collecting tasks, respectively, significantly outperforming baseline methods.**

I. Introduction

Domestic service robots have drawn attention for their ability to assist people with reducing household chores such as cleaning [1], cooking [2], and organizing [3]. In household scenarios, these robots often encounter complex scenes where they must manipulate objects in cluttered environments (MoC). In these environments, objects are randomly placed and may have different orientations and configurations, as shown in Figure 1. To accomplish MoC tasks, the robot must identify objects, plan the sequence of manipulations, and execute the task without causing collisions or damage to the objects [4]. Although advances in computer vision have effectively addressed object identification [5, 6, 7], planning and control in MoC tasks remain a challenge.

Previous methods for manipulating objects typically involve designing action primitives and planning paradigms for specific MoC tasks. For instance, VMRD [8] was explicitly developed for grasping stacked objects, utilizing a tree structure to indicate the stacked objects and grasping them in order according to the tree. [4] proposes a collision-checking module to guide object planning and employs action primitives to move blocked objects. However, these specific-designed action primitives and planning paradigms can limit the agility of robotics, the scope of task-solving, and the generalization capabilities of these methods in different situations, despite their impressive performance in manipulation tasks.

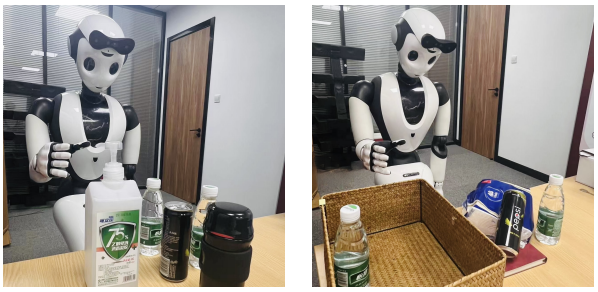


Fig. 1: Examples of Ginger XR1 robot [9] performing tasks in a domestic service scenario. On the left, it is shown grasping a blocked object. On the right, it is shown cleaning up the desktop.

On the other hand, learning-based techniques such as reinforcement learning (RL) have demonstrated generalization ability across many different domains [10]. However, standard RL can be inefficient when finding capable policies for tasks requiring long-horizon planning and sophisticated control [11], which is also the case for MoC. In this paper, we consider a scenario where the objects are randomly placed on the desk and the target object could be blocked, which is a common scenario in domestic service. We propose a solution to MoC based on the option framework (OF), which enhances RL through temporal abstraction actions called options [12, 13]. The standard OF discovers options from scratch based on RL, leading to collapsed options and hindering learning [14]. Our method, called **O**bject-**O**riented **O**ption **F**ramework (O3F), develops a specific option space for various object movements that are generalizable for MoC. Based on this, O3F solves MoC through two key components: (1) an option planner that generates an option from the option space, indicating the object movement, and (2) a universal option executor that controls the robot to complete the option. Unlike the standard OF that learns independent policies for each option and expects them to master valuable skills, O3F trains a universal option executor with a specific reward function to ensure that the executor can learn the skills useful for MoC.

[1]National Key Laboratory of Novel Software Technology, Nanjing University. [2]CloudMinds Robotics. [3]Polixir Technology.

*Equal contribution.

Yang Yu is the corresponding author. Email: yuy@nju.edu.cn.

Accepted by IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'23).

We conduct simulation experiments using two types of robots: the Ginger XR1 (shown in Figure 1) and a robot arm from Gym environment [15]. In the experiments with the Ginger XR1, O3F achieves a 91.3% success rate for grasping blocked objects, while the RL baseline struggled to learn a grasping policy. In the experiments with the robot arm, O3F achieves a 72.4% and 90% success rate on the grasping and collecting tasks, respectively, outperforming baseline learning-based approaches by 28.7%, which demonstrates the wide applicability of O3F. This research primarily focuses on developing a learning framework for solving MoC. O3F can be tested on a physical robot using existing sim-to-real techniques [16, 17]. We leave real robot adaptation of our method as part of future work.

The rest of this paper is organized as follows: Related works are discussed in Section II; Section IV presents the O3F method; Implementation details are elaborated in Section V; Experiment results showing the performance of our method are presented in Section VI; and finally, some conclusions and discussions are provided in Section VII.

II. Related Works

A. Robotics Manipulation in Clutter

Previous research has shown significant interest in robotics manipulation in clutter [18, 19]. Notably, robotics grasping [20, 21] has emerged as a critical area of research, with methods such as [8, 4, 22] focusing on planning the manipulation order of objects. For example, [4] proposes a collision-checking module for selecting graspable objects, while [8] uses a tree structure to determine which object to grasp. Though these methods show promising performance, they suffer from poor generalization since they are task-specific. This limitation is relieved with the development of learning-based methods in recent years, which have demonstrated some extent of generalization ability. RL methods have exhibited their potential in addressing complex robotics manipulation problems, particularly in planning the grasping order. For instance, VPG [22] models the planning problem in dense clutter as an RL task and demonstrates that a policy can rapidly learn robotics grasping from scratch in dense clutter through a simple RL algorithm, such as DQN [23], by alternately executing action primitives. Additionally, [24] extends the set of action primitives in [22] and learns a generator and an evaluator to select over action primitives. There are also some graph-based RL algorithms that teach a robot to grasp [25, 26], which primarily focus on the area of visual perception. Besides, previous works also explore other scenes of manipulation in clutter, such as obstacle rearrangement [27, 28], object matching [29], and object stacking [30, 31]. However, their reliance on action primitives limits the robot's flexibility and hinders their applicability to new scenarios.

B. Option Framework for Reinforcement Learning

The Option Framework (OF) [12] offers a promising architecture for solving long-term, complex decision-making problems. OF learns both the option selection policy and multiple option policies simultaneously. An option can be a pre-coded behavior skill or a policy learned from scratch. Research in this area can be divided into option discovery and option learning. Option discovery focuses on identifying useful sub-goals or options within an environment, while option learning focuses on learning the policies associated with these options to effectively achieve the identified sub-goals. For option discovery, the option-critic architecture [13] shows that OF can train capable policies without the need to design the option. For option learning, previous works define a set of options representing various skills. With such a set, an inter-option policy can be trained for skill composition [32]. In this paper, our focus is on option learning. OF has demonstrated its effectiveness in solving robotics manipulation problems [31]. However, our work introduces the concept of object-oriented options that specifically focus on object movement.

III. Preliminary

A. Reinforcement Learning

An RL task can be formulated as a Markov decision process (MDP) [33, 34], which is described as a tuple $\mathcal{M} = (\mathcal{S}, \mathcal{A}, P, r, \gamma)$. Here, $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, P denotes the transition probability, and r denotes the reward function that reflects the quality of the agent's behavior. γ is the discount factor determining future rewards' weight. P is the transition function of the environment. A policy $\pi : \mathcal{S} \to \Delta(\mathcal{A})$ is a mapping function from the state space to the probability space over the action space. The RL agent interacts with the environment as follows: at timestep t, the agent observes a state s_t from the environment and executes an action $a_t \sim \pi(\cdot|s_t)$. Then, the agent receives a reward $r(s_t, a_t)$ and transitions to the next state $s_{t+1} \sim P(\cdot|s_t, a_t)$. RL aims to find a policy that maximizes the expected cumulative rewards.

B. Option Framework

An option [12] corresponds a triplet $o\{\mathcal{I}^o, \pi^o, \beta^o(s)\}$. Here, $\mathcal{I}^o \subseteq \mathcal{S}$ denotes the set of initialization states, π^o refers to the intra-policy of this option, which can be pre-trained with standard RL using a reward function specific to the subtask, and $\beta^o(s) \in \{0,1\}$ is the termination function that indicates whether an option is terminated at a given state. The Option Framework (OF) trains an inter-option policy $\Omega(o|s) : \mathcal{S} \to \mathcal{O}$ that selects execution options to maximize an expected discounted return. Here, $\mathcal{O}$ denotes the option space, typically a real number space. The inter-option policy plans over options to complete a task. In the rest of the paper, we refer to the intra- and inter-option policies as option executor and option planner, respectively.

IV. Method

This section introduces the definition of an object-oriented option, which is the foundation of O3F, followed by introducing four types of the processed state. We then present the MDP formulation of the option planner (OP) and the universal object-oriented option executor (OE). Finally, we describe the joint training procedure of OP and OE.

A. Object-Oriented Option Framework

The standard Option Framework (OF) discovers options from scratch based on RL, which can result in collapsed options and hinder learning [14]. This issue can be addressed if there is a predetermined option space where the objective of each option is clearly defined in advance. The effective MoC process can be decomposed into repeated executions of moving designated objects to a target location. Based on this idea, we developed the Object-Oriented Option Space for MoC, a multi-dimensional continuous space where each option depicts one object's movement, including the current location and target location of a specific object. We require that the intra-policy of each option completes the task of moving the designated object to the target location. This way, the MoC task can be completed by building an extra option planner to select appropriate options repeatedly.

Formally, each option o corresponds to a triplet $o\{\mathcal{I}^o, \pi_{\text{oe}}(\cdot|s,o), \beta(s,o)\}$:

- $\mathcal{I}^o$ is the initiation set. We assume that $\forall s \in \mathcal{S}, \forall o \in \mathcal{O} : s \in \mathcal{I}^o$ (i.e., all options are available everywhere), an assumption made in the majority of option-based algorithms [13]. Thus, the initiation set for any option is the state space: $\mathcal{I}^o = \mathcal{S}$.
- $\pi_{\text{oe}}(\cdot|s,o)$ makes decisions conditioned on state s and option o. Unlike traditional OF, which trains independent policies for each option, our method trains a universal intro-policy for all options, i.e., $\pi_{\text{oe}}(\cdot|s,o)$, which we call the universal object-oriented option executor (OE).
- $\beta(s,o)$ is the terminal function, which indicates whether option o terminates at state s. In our work, we assume it to be a known function that can be obtained by examining whether the object movement has been completed.

Then, we can formulate an Option Planner (OP) as a policy that makes decisions in the option space: $\pi_{\text{op}}(\cdot|s) : \mathcal{S} \to \mathcal{O}$. OP is important for OF in solving complex problems, as it can generate temporally-extended plans for making long-term decisions. Our method connects OP and OE through joint training and execution procedures, as shown in the overall structure of our method presented in Figure 2.

B. Types of the Processed States

This subsection introduces four types of the processed states serving as the input of OP and OE:

Object encoding tensor: To handle uncertain numbers and shapes of objects, we discretize the workbench and represent it as a 3D object encoding tensor (OET). Each element in the tensor represents an item occupying the corresponding position. Possible item types include objects, air, and robotics joints. We will discuss the implementation of the encoding tensor later in Section V.

Task information: This contains information needed to complete the original manipulation task. For example, task information in an object stacking task includes stack number, the maximum height of all objects, etc.

Joint state: This includes joint angles and positions of the controllable joints, which are normalized to $[-1, 1]$ using min-max normalization. Higher-order information on joint movements, such as angular speed and acceleration, can also be involved.

Option information: This represents other information for completing the option, including the coordinate of the object center, the coordinate of the destination object location, and the distance between the gripper and the target object center. In a simulation environment, obtaining the coordinates directly is feasible. In real-world applications, the coordinates can potentially be obtained through the calculation of depth camera data.

C. The Option Planner Formulation

The OP π_{op} makes the decision at the option level for completing the MoC task. This subsection will describe the designs of the MDP for OP optimization.

Action space and decision-making. The OP outputs an option $o_t \in \mathbb{R}^6$, representing two coordinates in the working space denoted by $(x_{\text{obj}}, y_{\text{obj}}, z_{\text{obj}})$ and $(x_{\text{pos}}, y_{\text{pos}}, z_{\text{pos}})$. The first coordinate is used to select one object, and the second coordinate represents the target position of the selected object to move. After sampling o_t from π_{op}, the option information is constructed through a mapping function map. This function maps the two coordinates to the corresponding objects whose geometric center is closest to the generated coordinates. Such mapping from the coordinate to the object ensures that the OP can robustly select objects. The option information is then transmitted to the OE. In the next timestep, if the option is not terminated (i.e., not completing the object movement), the o_t sampling process is skipped, and the option information keeps to the OE. We wait for the OE to complete the option for $T_{\max}$ timesteps and overwrite the option by re-sampling o_t.

State space. The state of the OP s_t^{op} contains three parts: OET, joint states, and task information. Task information provides the OP with global and complete knowledge for accomplishing the task. The OET gives the global status of the workbench for the policy to make correct decisions. Joint states are helpful in some corner cases where the robot's movement is restricted.

Rewards. The reward function for training OP is defined as:

$$r^{\text{op}} = r^{\text{op}}_{\text{success}} + 0.1 \times r^{\text{op}}_{\text{time}} + 0.1 \times r^{\text{op}}_{\text{task}}.$$

The term $r^{\text{op}}_{\text{success}}$ indicates whether the overall MoC task has been successfully completed. At the terminal step, the agent receives a $r^{\text{op}}_{\text{success}}$ of +1 if it accomplishes the task and a $r^{\text{op}}_{\text{success}}$ of -1 if it fails. $r^{\text{op}}_{\text{success}}$ is set to be -1 if the agent does not complete the task after the maximized timestep $T^{\text{op}}_{\max}$. $r^{\text{op}}_{\text{time}}$ is a time penalty term, which is -1 at each step. It encourages the agent to complete the task using as few motions as possible. $r^{\text{op}}_{\text{task}}$ is a task-related reward, indicating whether the task assigned by option planner has been completed. This reward can be easily designed to adapt to diverse tasks: in the object collecting task, $r^{\text{op}}_{\text{task}}$ is an indicator of whether objects stay in the valid area, and in the blocked object grasping task, $r^{\text{op}}_{\text{task}}$ is an indicator of

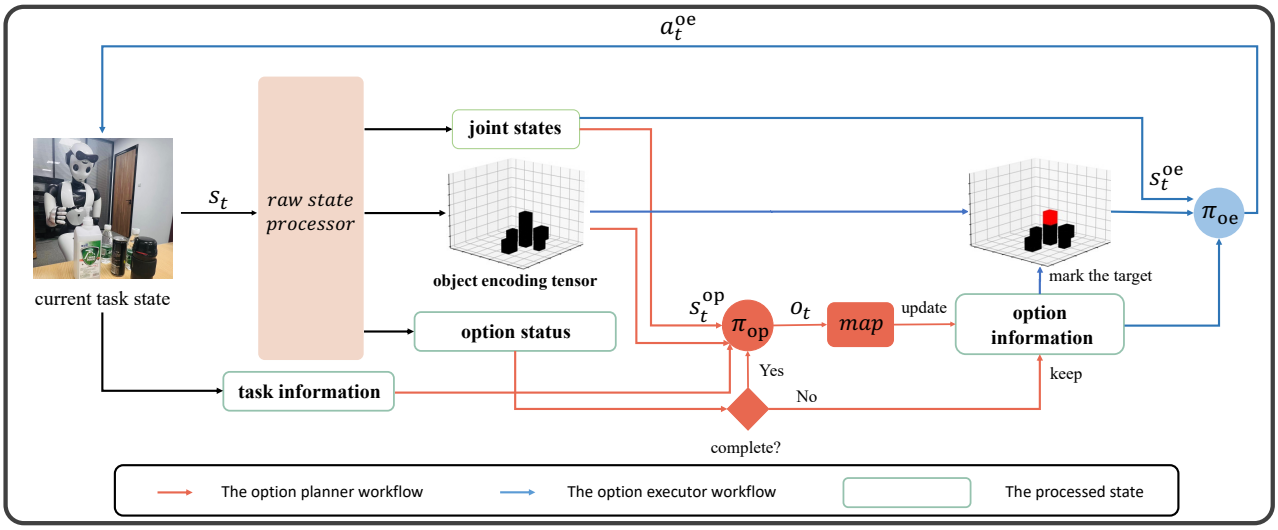


Fig. 2: Illustration of the structure and the deployment workflow of O3F. The framework comprises two modules: the option planner π_{op} that determines the option for object movement, and the option executor π_{oe} manipulates the objects to accomplish the option task.

whether the target object is grasped. There have been various practical techniques for determining the coefficients of each reward term [35]. In general, the weight of $r^{\text{op}}_{\text{success}}$ should be significantly higher than that of other terms, as it directly impacts the overall task success.

D. The Option Executor Formulation

The OE performs motion control to complete the option generated by the OP. We create an MDP to train the OE as the following:

Action space. The OE's action is the target torque of all controllable joints. In our experiments, we use the joint angle of all controllable joints due to the limitation of the selected simulation. As O3F optimizes OE with RL, it can be applied to other robot control modes.

State space. The OE's observation consists of the OET, joint state, and option information. The state space shares the same design as the OP, but the OET is extra marked with the target object to be moved.

Rewards. At each timestep, the reward for training π_{oe} is formulated as follows:

$$r^{\text{oe}}_t = r^{\text{oe}}_{\text{success}} + 0.001 \times r^{\text{oe}}_{\text{time}} + 0.1 \times r^{\text{oe}}_{\text{handnear}} + 0.1 \times r^{\text{oe}}_{\text{objnear}}.$$

$r^{\text{oe}}_{\text{success}}$ is the reward for option accomplishment. It equals $+1$ when the target object is successfully manipulated to the target location and -1 if it fails. $r^{\text{oe}}_{\text{time}}$ is similar to the OP, encouraging quick task completion. $r^{\text{oe}}_{\text{handnear}}$ is a reward to guide the gripper to get close to the target object. It is formulated as $r^{\text{oe}}_{\text{handnear}} = d_{t-1} - d_t$, where d_t denotes the distance between the gripper and the target object at timestep t. Like $r^{\text{oe}}_{\text{handnear}}$, $r^{\text{oe}}_{\text{objnear}} = d^o_{t-1} - d^o_t$ aims at guiding the target object is getting close to the target location, where d^o_t denotes the distance between the object's location and the target location.

E. Training Procedure

In the previous section, we introduced how to create MDPs to formulate OP and OE. This section presents the joint training procedure that connects OP and OE. Our method trains OE and OP simultaneously using the proximal policy optimization (PPO) [36] algorithm.

During sampling in the environments, the OP selects an option based on the OP state. Then, OE selects an action to control the robot and completes the option task until the option is completed or the maximum steps are reached. The samples are collected and stored in the buffer each time π_{op} and π_{oe} execute. Finally, OP and OE are optimized with the collected samples. In practical implementation, the O3F allows for the separate training of the OP and the OE. This means that the OE can be trained initially to handle object movement, and subsequently, the OP can be trained to plan and make decisions based on the learned OE. This sequential training approach enables the system to effectively learn and utilize the object-oriented options for planning and control.

V. Implementation Details

We use the proximal policy optimization RL algorithm (PPO) [36] to train the policy networks (π_{op} and π_{oe}). Figure 3 shows the network architecture used in our experiments. The policy networks are trained with a learning rate of 3×10^{-4} using the Adam optimizer [37]. All other hyper-parameters are the same as in the standard PPO implementation [38]. Our hardware configuration includes NVIDIA GeForce 2080ti GPUs and Intel(R) Xeon(R) Silver 4110 CPUs.

To construct the object encoding tensor (OET), we discretize the space above the workbench into 1 cm intervals and initialize a 3-D tensor with all zero values. The elements in the tensor are assigned different values to indicate which object is at the corresponding position. Through this process, the OET can represent various object layouts using tensors of the same size. In our implementation, we are accessible to the bounding box to the objects and use them to construct OET. In practice, OET can be obtained by visual processing. Previ-

ous works have studied object recognition [39], estimation of object position [40, 41], pose [42, 43], and shape [44, 45] in robotics. Through these well-verified techniques, the OET can be reconstructed. However, in this work, we neglect the visual processing module, as we are more interested in designing a general algorithm for MoC.

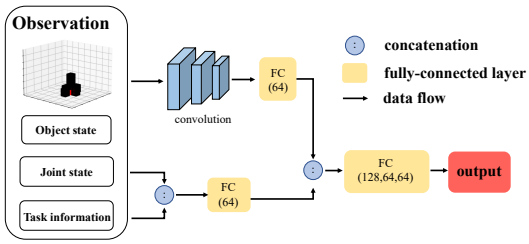


Fig. 3: Policy/value neural networks are constructed using convolutional layers followed by fully-connected layers to process object states, while fully-connected layers directly process joint states and task information.

VI. Experiments

This section will verify the effectiveness of O3F by conducting simulation experiments. Specifically, we aim to answer the following questions: (1) Can O3F generalize to different robots and manipulation tasks? (2) How does O3F compare to learning-based baseline methods? (3) How does the learned policy behave?

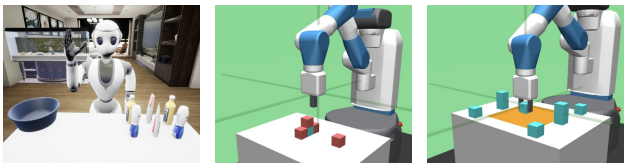
A. Descriptions About Robots and Manipulation Tasks

We first introduce two robots (Ginger XR1 and robot arm) and manipulation tasks (grasping and collecting) used in our experiments.

Ginger XR1 robot (Figure 4(a)) is a humanoid robot that is based on UE4 [46]. In our experiments, the agent controls the 7-DoF right arm, the chassis, and the open/close function of the right hand. The evaluation task is to grasp a container of orange juice that is blocked by several objects. The robot moves the obstacle to a basket to make room for grasping the target. The task fails if (1) a collision occurs (e.g., collisions between a joint and the desk), (2) the object falls, or (3) the maximum decision steps are reached (8 for the option planner and 150 for the option executor). During the training phase, the number and position of objects are randomly sampled. The test cases are shown in Figure 5 (row 1). We design these specific test cases to ensure that the target object is blocked. It is important to note that in practice, the object positions are randomly sampled across the workbench, and each object position may have an offset, ensuring the diversity of test cases.

Robot arm is from the MuJoCo platform [47]. The workspace is a 50 cm $\times$ 70 cm area. The task fails when (1) any object drops off the desk, (2) a collision occurs, or (3) reaches maximum decision steps (16 for the option planner and 100 for the option executor). The details are as follows:

- Blocked object grasping (Figure 4(b)): Objects are randomly placed on a table where obstacles can block the target object (cyan). The number of objects is uncertain. The agent must grasp the target object without colliding with or damaging other objects. Test cases are shown in Figure 5 (rows 2-3).
- Object collecting (Figure 4(c)): The agent collects all objects scattered over the desk and brings them to a specified area (yellow area in the figure). The number of objects ranges from 1 to 6. In our experiments, we reuse and fix the parameters of the option executor trained in the grasping task and only train the option planner.



(a) Grasp (Ginger XR1) (b) Grasp (c) Collect

Fig. 4: An visualization of tasks in our experiments. **(a):** Grasp the blocked orange juice with Ginger XR1. **(b):** Grasp the blocked object (cyan block) with robot arm. **(c):** Collect all objects together.

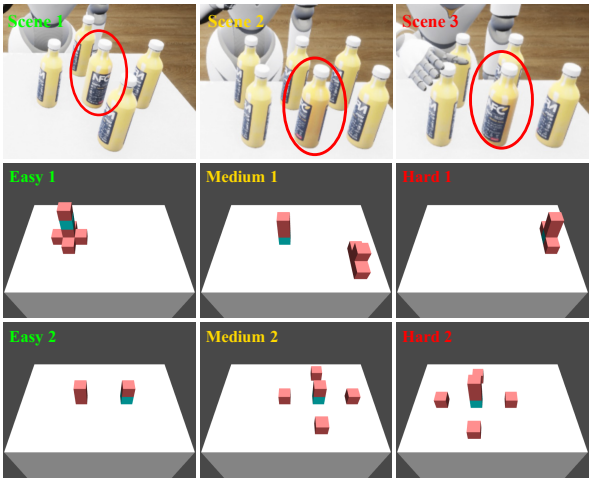


Fig. 5: Various test cases of the robotics grasping task. The target object is marked by red circle or cyan block in this figure.

All reported results are averaged over five random runs, each interacting with the environment for 30 million environment timesteps.

B. Results on Ginger XR1

Table I presents the grasping success rate of O3F on the Ginger robot. The results show that O3F successfully learns a policy that grasps blocked objects in cluttered scenes, achieving an average success rate of 91.3%. For comparison, we train a PPO policy to directly control the robot for 30M steps with a reward function equivalent to that used for training the option planner in O3F. As a result, PPO fails to learn a capable policy for grasping blocked objects; the success rate is lower than 5% for all scenes. This result can be attributed to the difficulty of exploring successful examples in such a long-term and complex task. Figure 6 shows the execution process of O3F policy, which clears obstacles one by one and finally grasps the target.

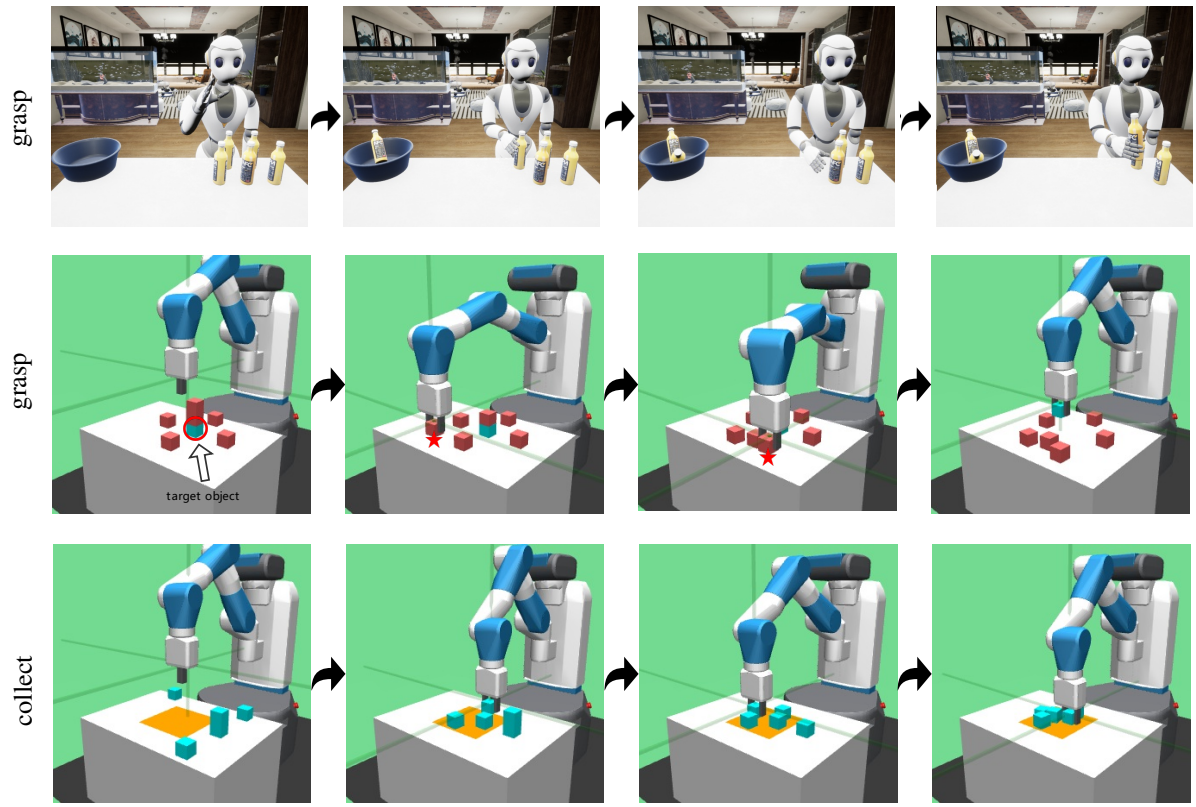


Fig. 6: Deployment examples of the policy learned by O3F on various MoC tasks. **Row 1:** On a Ginger XR1 robot, O3F learns to grasp the blocked orange juice. **Row 2:** In a grasping task on a robot arm, the cyan block is the target. O3F learns to move obstacles to the designated space area first and then grasp the target object. The ★ in the figure represents the target location of the generated option. **Row 3:** O3F learns to collect all objects and place them in the specified area.

TABLE I: Success rate (%) of grasping task on the Ginger XR1 Robot.

Method	Scene 1	Scene 2	Scene 3	Average
O3F (Ours)	**88.0**	**95.0**	**91.0**	**91.3**
PPO [36]	2.0	4.0	3.0	3.0

C. Results on Robot Arm

Comparison with baselines on grasping tasks. In this section, we conduct experiments on a robot arm and compare our method with two learning-based baseline methods: **VPG** and **PPO**. Specifically, we test the performance of these algorithms on a blocked object-grasping task. VPG uses DQN to train a selector that chooses between two action primitives (pushing and grasping) and is effective in grasping blocked objects. PPO is a reinforcement learning algorithm that trains a policy to control the robot's joints. We present the comparison results in Table II. O3F outperforms VPG on all three levels of tasks. This can be attributed that VPG solve the problem via planning based on the fixed action primitives, which are limited in its action flexibility. On the other hand, the vanilla learning-based algorithm, PPO, fails to learn capable policy, indicating that the difficulty of the long-term planning and control tasks. In contrast, O3F significantly outperforms the baselines in the more challenging tasks, e.g., O3F achieves a success rate of 60.6% on the Hard level, while PPO only has 8.33%. This result demonstrates O3F's capacity to solve these complex tasks.

TABLE II: Success rate (%) of blocked object grasping in varying scenes of tasks. The number following ± represents the standard deviation over five random runs.

Method	Easy	Medium	Hard	Average
O3F (Ours)	**87.8± 1.57**	**68.9± 12.2**	**60.6± 4.78**	**72.4**
VPG [24]	45.0± 5.93	45.6± 2.08	40.6± 11.9	43.7
PPO [36]	42.8± 3.93	10.6± 3.42	8.33± 7.58	20.6

Applicability to different manipulation tasks. In addition to the grasping task, we conduct experiments on the object-collecting task. We summarize the success rates of the optimized policies for various tasks in Table III. Our method achieves success rates of 72.4% and 90.0% in the two tasks, respectively. These results demonstrate that O3F generally learns to solve various manipulation tasks in clutter. Furthermore, our method learns the policy more efficiently than the baseline RL method, PPO. We also observe that the option executor is reusable across different tasks, indicating its potential as an effective sub-policy for complex problem-solving.

Execution examples of O3F policy. We demonstrate the performance of policies learned by O3F by visualizing their behaviors on different tasks. Figure 6 illustrates the planning workflows. For instance, in the blocked object grasping task,

TABLE III: Success rate (%) of O3F and PPO in grasping and collecting tasks with the robot arm.

Method	grasping	collecting
O3F (Ours)	**72.4**	**90.0**
PPO [24]	20.6	3.33

the agent selects obstacles that directly block the target object and relocates them to a designated space (marked by the star symbol in the figure). Additionally, it effectively completes the object-collecting task by correctly selecting the blocks and the target locations.

VII. Conclusion and Future Work

We propose an option-based framework, called O3F, that includes an option planner that determines how an object moves and a universal option executor that completes the object movement. Empirical results demonstrate that O3F can generally learn capable policies in various manipulation tasks in cluttered environments. Our results suggest that O3F has the potential to master various real-world robot manipulations in clutter. However, there are still limitations and future works to be discussed.

State representation. In the raw state processor, we neglect the visual processing module and assume that the object encoding tensor can be obtained through third-party components since this paper focuses on designing an efficient method for MoC. Fortunately, related works have shown powerful abilities in object recognition [48, 49], pose estimation [43, 42], occluded object recovery [50], and shape approximation [44, 45]. As mentioned in section V, it is possible to reconstruct an object encoding tensor or equivalent representation by combining these techniques. Even if reconstruction errors exist, promising techniques in Sim2Real [51, 52, 53] can alleviate such errors.

Real world deployment. Our experiments were conducted in a simulation environment, but they can be supplemented with real-world experiments to validate the effectiveness of our approach in practical scenarios. Furthermore, it would be interesting to consider more cluttered environments, such as a storage basket, for real-world deployment. This would provide a more challenging and realistic scenario for the robots to operate in, and it would be valuable to evaluate the performance of our method in such environments. Additionally, in this study, the robots only consider the position of objects. However, it is important to acknowledge that real-world manipulation tasks often involve additional complex factors, such as slipperiness, grasping orientation, and specific points of grasping. To further enhance our approach, it would be valuable to explore and incorporate these factors into our methodology. This would provide a more comprehensive and realistic evaluation of the proposed method, and it would better align with the complexities of real-world manipulation tasks.

VIII. Acknowledgement

This work is partly supported by the National Key Research and Development Program of China (2021ZD0140409), the National Science Foundation of China (61921006, 61876119, 62276126), and the Natural Science Foundation of Jiangsu (BK20221442). We would like to thank Chao Zhang and Fan-Ming Luo for their valuable discussions on the design of our method. We would also like to acknowledge the support and assistance provided by Yujia Xiao, Qiang Luo, and Wenhao Cui in utilizing the Ginger simulation environment. Additionally, we extend our appreciation to the anonymous reviewers for their comprehensive suggestions on improving this paper.

References

[1] Yichen Lu and Zheng Liao. Towards happy housework: Scenario-based experience design for a household cleaning robotic system. *EAI Endorsed Transactions on Scalable Information Systems*, 10(3):12, 2023.

[2] Grzegorz Sochacki, Arsen Abdulali, and Fumiya Iida. Mastication-enhanced taste-based classification of multi-ingredient dishes for robotic cooking. *Frontiers Robotics AI*, 9:886074, 2022.

[3] Irene Garcia-Camacho, Júlia Borràs Sol, Berk Çalli, Adam Norton, and Guillem Alenyà. Household cloth object set: Fostering benchmarking in deformable object manipulation. *IEEE Robotics and Automation Letters*, 7(3):5866–5873, 2022.

[4] Adithyavairavan Murali, Arsalan Mousavian, Clemens Eppner, Chris Paxton, and Dieter Fox. 6-dof grasping for target-driven object manipulation in clutter. In *ICRA*, 2020.

[5] Andy Zeng, Shuran Song, Kuan-Ting Yu, Elliott Donlon, Francois R Hogan, Maria Bauza, Daolin Ma, Orion Taylor, Melody Liu, Eudald Romo, et al. Robotic pick-and-place of novel objects in clutter with multi-affordance grasping and cross-domain image matching. *The International Journal of Robotics Research*, 41(7):690–705, 2022.

[6] Vo Duy Cong and Le Duc Hanh. A new decoupled control law for image-based visual servoing control of robot manipulators. *International Journal of Intelligent Robotics and Applications*, 6(3):576–585, 2022.

[7] Muhammad Zubair Irshad, Chih-Yao Ma, and Zsolt Kira. Hierarchical cross-modal agent for robotics vision-and-language navigation. In *ICRA*, 2021.

[8] Hanbo Zhang, Xuguang Lan, Site Bai, Lipeng Wan, Chenjie Yang, and Nanning Zheng. A multi-task convolutional neural network for autonomous robotic grasping in object stacking scenes. In *IROS*, 2019.

[9] CloundMinds Robotics. Cloud Ginger XR-1 robot.

[10] Robert Kirk, Amy Zhang, Edward Grefenstette, and Tim Rocktäschel. A survey of zero-shot generalisation in deep reinforcement learning. *Journal of Artificial Intelligence Research*, 76:201–264, 2023.

[11] Abhishek Gupta, Vikash Kumar, Corey Lynch, Sergey Levine, and Karol Hausman. Relay policy learning: Solving long-horizon tasks via imitation and reinforcement learning. In *CoRL*, 2019.

[12] Richard S. Sutton, Doina Precup, and Satinder Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artif. Intell.*, 112(1-2):181–211, 1999.

[13] Pierre-Luc Bacon, Jean Harb, and Doina Precup. The option-critic architecture. In *AAAI*, 2017.

[14] Akhil Bagaria, Jason K. Senthil, Matthew Slivinski, and George Konidaris. Robustly learning composable options in deep reinforcement learning. In Zhi-Hua Zhou, editor, *IJCAI*, pages 2161–2169, Virtual Event, 2021.

[15] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016.

[16] Josh Tobin, Lukas Biewald, Rocky Duan, Marcin Andrychowicz, Ankur Handa, Vikash Kumar, Bob McGrew, Alex Ray, Jonas Schneider, Peter Welinder, Wojciech Zaremba, and Pieter Abbeel. Domain randomization and generative models for robotic grasping. In *IROS*, 2018.

[17] Manuel Kaspar, Juan David Muñoz Osorio, and Jürgen Bock. Sim2real transfer for reinforcement learning without dynamics randomization. In *IROS*, 2020.

[18] Qingkai Lu, Kautilya Chenna, Balakumar Sundaralingam, and Tucker Hermans. Planning multi-fingered grasps as probabilistic inference in a learned deep network, 2018.

[19] Ian Lenz, Honglak Lee, and Ashutosh Saxena. Deep learning for detecting robotic grasps. In *IJRR*, 2015.

[20] Kilian Kleeberger, Richard Bormann, Werner Kraus, and Marco F Huber. A survey on learning-based robotic grasping. *Current Robotics Reports*, 1(4):239–249, 2020.

[21] Jinda Cui and Jeff Trinkle. Toward next-generation learned robot manipulation. *Science Robotics*, 6(54):9461, 2021.

[22] Andy Zeng, Shuran Song, Stefan Welker, Johnny Lee, Alberto Rodriguez, and Thomas A. Funkhouser. Learning synergies between pushing and grasping with self-supervised deep reinforcement learning. In *IROS*, 2018.

[23] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin A. Riedmiller, Andreas Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, pages 529–533, 2015.

[24] Kechun Xu, Hongxiang Yu, Qianen Lai, Yue Wang, and Rong Xiong. Efficient learning of goal-oriented push-grasping synergy in clutter. *IEEE Robotics and Automation Letters*, 6(4):6337–6344, 2021.

[25] Yixin Lin, Austin S. Wang, Eric Undersander, and Akshara Rai. Efficient and interpretable robot manipulation with graph neural networks. *IEEE Robotics and Automation Letters*, 7(2):2740–2747, 2022.

[26] Niklas Funk, Georgia Chalvatzaki, Boris Belousov, and Jan Peters. Learn2assemble with structured representations and search for robotic architectural construction. In *CoRL*, 2021.

[27] SangHun Cheong, Brian Y. Cho, Jinhwi Lee, Jeongho Lee, Dong Hwan Kim, Changjoo Nam, ChangHwan Kim, and Sung-Kee Park. Obstacle rearrangement for robotic manipulation in clutter using a deep q-network. *Intelligent Service Robotics*, 14(4):549–561, 2021.

[28] Daniel Strömbom and Andrew J. King. Robot collection and transport of objects: A biomimetic process. *Frontiers Robotics AI*, 5:48, 2018.

[29] Walter Goodwin, Sagar Vaze, Ioannis Havoutis, and Ingmar Posner. Semantically grounded object matching for robust robotic scene rearrangement. In *ICRA*, 2022.

[30] Ryosuke Iinuma, Yusuke Hori, Hiroyuki Onoyama, Yukihiro Kubo, and Takanori Fukao. Robotic forklift for stacking multiple pallets with RGB-D cameras. *Journal of Robotics and Mechatronics*, 33(6):1265–1273, 2021.

[31] Xintong Yang, Ze Ji, Jing Wu, Yu-Kun Lai, Changyun Wei, Guoliang Liu, and Rossitza Setchi. Hierarchical reinforcement learning with universal policies for multistep robotic manipulation. *IEEE Transactions on Neural Networks and Learning Systems*, 2021.

[32] Xue Bin Peng, Michael Chang, Grace Zhang, Pieter Abbeel, and Sergey Levine. MCP: learning composable hierarchical control with multiplicative compositional policies. In *NeurIPS*, 2019.

[33] Martin L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley Series in Probability and Statistics. 1994.

[34] Richard S. Sutton and Andrew G. Barto. Reinforcement learning: An introduction. *IEEE Transaction on Neural Networks*, page 1054, 1998.

[35] Athanasia Karalakou, Dimitrios Troullinos, Georgios Chalkiadakis, and Markos Papageorgiou. Deep reinforcement learning reward function design for autonomous driving in lane-free traffic. *System*, 11(3):134, 2023.

[36] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017.

[37] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *ICLR*, 2015.

[38] Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann. Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 22(268):1–8, 2021.

[39] Shimon Ullman. Visual object recognition. In *ISTCS*, 1996.

[40] Ashley W. Stroupe, Martin C. Martin, and Tucker R. Balch. Distributed sensor fusion for object position estimation by multi-robot systems. In *ICRA*, 2001.

[41] Guoguang Du, Kai Wang, Shiguo Lian, and Kaiyong Zhao. Vision-based robotic grasping from object localization, object pose estimation to grasp estimation for parallel grippers: a review. *Artificial Intelligence Review*, 54(3):1677–1734, 2021.

[42] Vassileios Balntas, Andreas Doumanoglou, Caner Sahin, Juil Sock, Rigas Kouskouridas, and Tae-Kyun Kim. Pose guided RGBD feature learning for 3d object pose estimation. In *ICCV*, 2017.

[43] Umar Asif, Jianbin Tang, and Stefan Harrer. Graspnet: An efficient convolutional neural network for real-time grasp detection for low-powered devices. In *IJCAI*, 2018.

[44] Kai Huebner, Steffen Ruthotto, and Danica Kragic. Minimum volume bounding box decomposition for shape approximation in robot grasping. In *ICRA*, 2008.

[45] Jiajun Wu, Chengkai Zhang, Tianfan Xue, Bill Freeman, and Josh Tenenbaum. Learning a probabilistic latent space of object shapes via 3d generative-adversarial modeling. In *NeurIPS*, 2016.

[46] Epic Games. Unreal engine 4, 2019.

[47] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *IROS*, 2012.

[48] Daniel Maturana and Sebastian A. Scherer. Voxnet: A 3d convolutional neural network for real-time object recognition. In *IROS*, 2015.

[49] Theo Gevers and Arnold W. M. Smeulders. Color based object recognition. In *ICIAP*, 1997.

[50] Seunghyeok Back, Joosoon Lee, Taewon Kim, Sangjun Noh, Raeyoung Kang, Seongho Bak, and Kyoobin Lee. Unseen object amodal instance segmentation via hierarchical occlusion modeling. In *ICRA*, 2022.

[51] Josh Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. In *IROS*, 2017.

[52] Yevgen Chebotar, Ankur Handa, Viktor Makoviychuk, Miles Macklin, Jan Issac, Nathan D. Ratliff, and Dieter Fox. Closing the sim-to-real loop: Adapting simulation randomization with real world experience. In *ICRA*, 2019.

[53] Jan Matas, Stephen James, and Andrew J. Davison. Sim-to-real reinforcement learning for deformable object manipulation. In *CoRL*, 2018.