Reinforcement Learning with Spatial Reasoning for Dexterous Robotic Manipulation

Bowen Jiang

CMU-RI-TR-24-47

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Thesis Committee:
Prof. David Held (Chair)
Prof. Changliu Liu
Xianyi Cheng

Submitted in partial fulfillment of the requirements for the degree of Master of Science in Robotics.

Abstract

Robotic manipulation in unstructured environments requires adaptability and the ability to handle a wide variety of objects and tasks. This thesis presents novel approaches for learning robotic manipulation skills using reinforcement learning (RL) with spatially-grounded action spaces, addressing the challenges of high-dimensional, continuous action spaces and alleviating the need for extensive training data.

Our first contribution, HACMan (Hybrid Actor-Critic Maps for Manipulation), introduces a hybrid actor-critic model that maps discrete and continuous actions to 3D object point clouds, enabling complex non-prehensile interactions based on the spatial features of the object. Our second contribution, HACMan++ (Spatially-Grounded Motion Primitives for Manipulation), extends the framework to more generalized manipulation. It includes a diverse set of parameterized motion primitives, allowing the robot to perform a wide range of tasks by chaining these primitives together.

Through extensive experiments in simulation and on real robot platforms, we demonstrate the effectiveness of our proposed approaches in learning complex, long-horizon manipulation tasks with strong generalization to novel objects and environments. The thesis contributes to the state-of-the-art of robotic manipulation by providing novel RL approaches that leverage spatially-grounded action spaces and motion primitives, opening up new possibilities for more intelligent and capable robotic systems.

Acknowledgement

I would like to express my deepest gratitude to my parents and friends for their constant support, encouragement, and belief in me throughout my academic journey. I am truly grateful to my mentor, Wenxuan Zhou, for her invaluable guidance, inspirations, and advice that have been instrumental in shaping my research direction. My sincere thanks go to Yilin Wu for the fruitful collaborations and thought-provoking discussions that have greatly enriched my research experience. I am deeply indebted to my advisor, Prof. David Held, for his support and the opportunities he has provided, which have been essential in refining the quality of my work.

I would also like to acknowledge Zhanyi Sun, Yufei Wang, Ben Eisner, and other RPAD members for the inspiring discussions and valuable suggestions that have sparked new ideas and helped me approach my research from different perspectives.

For the HACMan project, I extend my thanks to Thomas Weng, Daniel Seita, Mrinal Kalakrishnan, Homanga Bharadhwaj, Patrick Lancaster, Vikash Kumar, Jay Vakil, and Priyam Parashar for their insightful feedback throughout the project. This work was supported by the Meta AI Mentorship Program.

For the HACMan++ project, I am grateful to Chialiang Kuo for his help in running experiments on the Adroit environment. This work is supported by National Institute of Standards and Technology under Grant No. 70NANB23H178.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Robotic manipulation, which involves interacting with and manipulating objects in the environment, is a crucial skill for robots to perform a wide range of tasks in various settings, from industrial assembly lines to household environments. Manipulation tasks often require a combination of prehensile (grasping) and non-prehensile (pushing, flipping, toppling, sliding) interactions [5, 23]. Learning complex manipulation skills with fine-grained control to handle object pose and shape variations remains a significant challenge due to the high-dimensional, continuous action spaces involved.

Recent advancements in Deep Reinforcement Learning (RL) have shown promise in learning manipulation skills directly from visual observations [9, 18, 31, 57]. However, most existing approaches learn low-level, raw actions, such as gripper or joint movements [28, 58, 53, 56, 57]. These action spaces pose significant challenges for learning complex, long-horizon tasks due to the difficulties in exploration, credit assignment, and training stability [9, 31]. As a result, the learned policies often struggle to generalize to novel object instances or categories.

To address these challenges, it is crucial to select the right level of abstraction for the action space when applying RL to manipulation tasks. The action space should provide sufficient expressiveness to solve the task while also enabling sample-efficient learning and generalization. In this thesis, we propose novel approaches to define action spaces that are spatially grounded on the observed point cloud of the environment, allowing the policy to reason about the spatial properties of the objects and the scene.

Our first contribution, presented in "HACMan: Learning Hybrid Actor-Critic Maps for 6D Non-Prehensile Manipulation," focuses on non-prehensile manipulation tasks. We introduce a hybrid actor-critic model that maps discrete and continuous actions to the 3D object point cloud, enabling the robot to perform complex non-prehensile interactions based on the spatial features of the object. This approach demonstrates strong performance and generalization to unseen objects in tasks such as 6D object pose alignment.

Building upon the success of spatially grounded action spaces, our second pa-

per, "HACMan++: Spatially-Grounded Motion Primitives for Manipulation," extends the framework to include a diverse set of parameterized motion primitives. By defining an action space that includes the type of primitive (e.g., grasp, push, place), the location in the environment where the action is applied, and the parameters of the motion, we enable the robot to perform a wide range of tasks by chaining together these primitives. This approach significantly improves generalization across different objects and the ability to apply to diverse tasks, outperforming existing methods in complex manipulation scenarios.

The main contributions of this thesis are: 1) proposing novel spatially-grounded action spaces for learning both prehensile and non-prehensile manipulation skills with RL, and 2) demonstrating their effectiveness in solving diverse manipulation tasks with strong generalization. This work takes a step towards equipping robots with the ability to autonomously learn and adapt manipulation strategies for various environments and tasks.

Chapter 2

HACMan: Learning Hybrid Actor-Critic Maps for 6D Non-Prehensile Manipulation

Manipulating objects without grasping them is an essential component of human dexterity, referred to as non-prehensile manipulation. Non-prehensile manipulation may enable more complex interactions with the objects, but also presents challenges in reasoning about gripper-object interactions. In this work, we introduce Hybrid Actor-Critic Maps for Manipulation (HACMan), a reinforcement learning approach for 6D non-prehensile manipulation of objects using point cloud observations. HACMan proposes a *temporally-abstracted* and *spatially-grounded* object-centric action representation that consists of selecting a contact location from the object point cloud and a set of motion parameters describing how the robot will move after making contact. We modify an existing off-policy RL algorithm to learn in this hybrid discrete-continuous action representation. We evaluate HACMan on a 6D object pose alignment task in both simulation and in the real world. On the hardest version of our task, with randomized initial poses, randomized 6D goals, and diverse object categories, our policy demonstrates strong generalization to unseen object categories without a performance drop, achieving an 89% success rate on unseen objects in simulation and 50% success rate with zero-shot transfer in the real world. Compared to alternative action representations, HACMan achieves a success rate more than three times higher than the best baseline. With zero-shot sim2real transfer, our policy can successfully manipulate unseen objects in the real world for challenging non-planar goals, using dynamic and contact-rich non-prehensile skills. Videos can be found on the project website: `https://hacman-2023.github.io`.

2.1 Introduction

The ability to manipulate objects in ways beyond grasping is a critical aspect of human dexterity. Non-prehensile manipulation, such as pushing, flipping, toppling, and sliding objects, is essential for a wide variety of tasks where objects are difficult to grasp or where workspaces are cluttered or confined. However, non-prehensile manipulation remains challenging for robots; previous work has only shown results with limited object generalization [5, 15] or limited motion complexity, such as planar pushing or manipulating articulated objects with limited degrees of freedom [25, 51, 52]. We propose a method that generalizes across object geometries while showing versatile interactions for complex non-prehensile manipulation tasks.

We present **H**ybrid **A**ctor-**C**ritic Maps for **Man**ipulation (HACMan), a reinforcement learning (RL) approach for non-prehensile manipulation from point cloud observations. The first technical contribution of HACMan is to propose an **object-centric action representation** that is **temporally-abstracted** and **spatially-grounded**. The agent selects a contact location and a set of motion parameters determining the trajectory it should take after making contact. The contact location is selected from the observed object point cloud which provides spatial grounding. At the same time, the robot decisions become more temporally-abstracted because we focus on only learning the contact-rich portions of the action.

The second technical contribution of HACMan is to incorporate the proposed action representation in an actor-critic RL framework. Since the contact location is defined over a discrete action space (selecting a contact point among the points in the object point cloud) and the motion parameters (defining the trajectory after contact) are defined over a continuous action space, our action representation is in a hybrid discrete-continuous action space. In HACMan, the actor network outputs *per-point* continuous motion parameters and the critic network predicts *per-point* Q-values over the object point cloud. Different from common continuous action space RL algorithms [8, 20], the per-point Q-values are used both to update the actor and also to compute the probability for selecting the contact location. We modify the update rule of an existing off-policy RL algorithm to incorporate such a hybrid action space.

We apply HACMan to a 6D object pose alignment task with randomized initial object poses, randomized 6D goal poses, and diverse object geometries (Fig. 2.1). In simulation, our policy generalizes to unseen objects without a performance drop, obtaining an 89% success rate on unseen objects. In addition, HACMan achieves a training success rate more than three times higher than the best baseline with an alternative action representation. We also perform real robot experiments with zero-shot sim2real transfer, in which the learned policy performs dynamic object interactions over unseen objects of diverse shapes with non-planar goals. Our contributions include:

- We propose a novel object-centric action representation based on 3D spatial action maps to learn complex non-prehensile interactions. We also

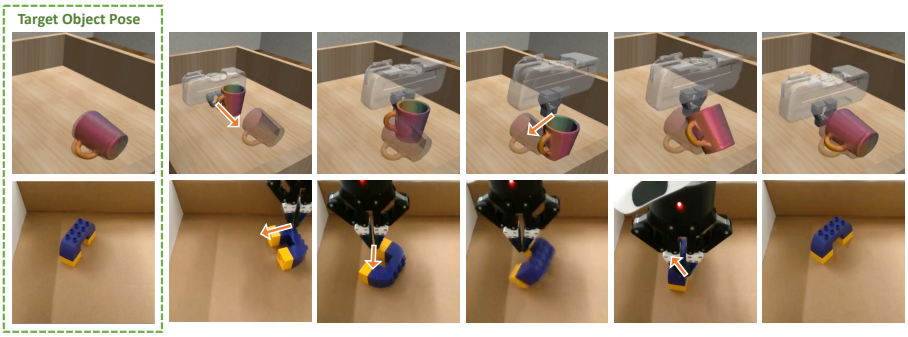


Figure 2.1: We propose HACMan (**H**ybrid **A**ctor-**C**ritic Maps for **Man**ipulation), which allows non-prehensile manipulation of unseen objects into arbitrary stable poses. With HACMan, the robot learns to push, tilt, and flip the object to reach the target pose, which is shown in the first column and in the top row with transparency. The policy allows for dynamic object motions with complex contact events in both simulation (top) and in the real world (bottom). The performance of the policy is best understood from the videos on the website: `https://hacman-2023.github.io`.

modify an existing off-policy RL algorithm to incorporate such a hybrid discrete-continuous action space.

- The proposed action representation demonstrates substantive improvements of performance over the baselines and shows strong generalization to unseen objects.
- The learned policy showcases complex contact-rich and dynamic manipulation skills, including pushing, tilting, and flipping, shown both in simulation and with a real robot.

2.2 Related Work

Non-prehensile manipulation. Non-prehensile manipulation is defined as manipulating objects without grasping them [23]. Many non-prehensile manipulation tasks involve complex contact events among the robot, the object, and the environment, which lead to significant challenges in state-estimation, planning and control [5, 15, 26, 52]. Recent work has applied learning-based methods in non-prehensile manipulation, but they are limited in terms of either skill complexity [48, 25, 51, 19] or object generalization [56, 19]. In contrast, our work shows 6D object manipulation involving more complex object interactions while also generalizing to a large variety of unseen object geometries.

Visual Reinforcement Learning with Point Clouds. Recent research has explored various ways of incorporating point clouds into RL [35, 21]. To overcome the optimization difficulties, previous work has tried pre-training the feature extractor with an auxiliary loss [16], initializing the RL policy with be-

havior cloning [47], or using student-teacher training [3, 4] (see detailed discussion in Appendix A.6). Our method does not require these additional training procedures due to the benefits of the proposed action representation. In the experiments, we show that the baselines following the most relevant previous work [35, 3, 4] struggles when the task becomes more complex.

Spatial action maps. Similar to our method, recent work has explored spatial action maps that are densely coupled with visual input instead of compressing it into a global embedding, based on images [55, 51, 7], point clouds [36, 25, 44], or voxels [39]. Unlike previous works with spatial action maps that consider one-shot decision making (similar to a bandit problem) [25, 51, 48] or rely on expert demonstrations with ground truth actions [55, 36, 39], our method reasons over multi-step sequences with no expert demonstrations. For example, Xu et al. [51] only chooses a single contact location followed by an action sequence, rather than a sequence of contact interactions. Unlike previous work in spatial action maps that uses DQN with discrete actions [54, 17, 48, 7], our hybrid discrete-continuous action space allows the robot to perform actions without discretization. Furthermore, we demonstrate the benefit of spatial action representations when applied to a 6D non-prehensile manipulation task, which is more challenging than the pick-and-place and articulated object manipulation tasks in previous work.

RL with hybrid discrete-continuous action spaces. Most RL algorithms focus on either a discrete action space [24] or a continuous action space [20, 8, 12]. However, certain applications are defined over a hybrid action space where the agent selects a discrete action and a continuous parameter for the action [30, 13, 50]. Unlike previous work, our hybrid action space uses a spatial action representation in which the discrete actions are defined over a map of visual inputs. The closest to our work is Feldman et al. [7] in terms of applying RL to spatial action maps, but they only consider a finite horizon of 2. We include formal definitions of the policies over the hybrid action space and modify the loss functions and exploration accordingly.

2.3 Preliminaries

A Markov Decision Process (MDP) models a sequential stochastic process. An MDP can be represented as a tuple (S, A, P, r), where S is the state space; A is the action space; $P(s_{t+1}|s_t, a_t)$ is the transition probability function, which models the probability of reaching state s_{t+1} from state s_t by taking action a_t; $r(s_t, a_t, s_{t+1})$ is the immediate reward at time t. The objective is to maximize the return R_t, defined as the cumulative discounted reward $R_t = \sum_{i=0}^{\infty} \gamma^i r_{t+i}$. Given a policy π, the Q-function is defined as $Q^\pi(s, a) = \mathbb{E}_\pi[R_t|s_t = s, a_t = a]$.

HACMan is built on top of Q-learning-based off-policy algorithms with a continuous action space [8, 12, 13]. In these algorithms, we define a deterministic policy π_θ parameterized by θ and a Q-function Q_ϕ parameterized by ϕ. Note that since the policy is deterministic, we use epsilon-greedy during exploration. Given a dataset D with transitions (s_t, a_t, s_{t+1}), the Q-function loss is defined

according to the Bellman residual:

$$L(\phi) = \mathbb{E}_{s_t, a_t, s_{t+1} \sim D}[(Q_\phi(s_t, a_t) - y_t)^2], \tag{2.1}$$

where y_t is defined as:

$$y_t = r_t + \gamma Q_\phi(s_{t+1}, \pi_\theta(s_{t+1})). \tag{2.2}$$

The policy loss for π_θ is defined to maximize the Q-function:

$$J(\theta) = -\mathbb{E}_{s_t \sim D}[Q^{\pi_\theta}(s_t, a_t)|_{a_t = \pi_\theta(s_t)}]. \tag{2.3}$$

2.4 Problem Statement and Assumptions

We focus on the task of 6D object pose alignment with non-prehensile manipulation. The objective of the robot is to perform a sequence of non-prehensile actions (i.e. pushing, flipping) to move an object on the table into a target goal pose. We assume that the goals are stable object poses on the table. The robot policy observes the point cloud of the scene from depth cameras, denoted as $\mathcal{X}$. We assume that the point cloud observation is segmented between the background and the object to be manipulated. Thus, the full point cloud $\mathcal{X}$ consists of the object point cloud $\mathcal{X}^{obj}$ and the background point cloud $\mathcal{X}^b$. The feature for each point is a 4-dimensional vector, including a 1-dimensional segmentation mask and a 3-dimensional goal flow vector (will be defined in Section 2.5.3).

2.5 Method

2.5.1 Action Representation

We propose an object-centric action space that consists of two parts: a **contact location** a^{loc} on an object and **motion parameters** a^m which define how the robot moves to interact with the object after contact. As shown in Fig. 2.2, to execute an action, the end-effector will first move to a location in free space near location a^{loc}, after which it will interact with the object using the motion parameters a^m. After the interaction, the end-effector will move away from the object, a new observation is obtained, and the next action can be taken.

Specifically, given the object point cloud $\mathcal{X}^{obj} = \{x_i \mid i = 1 \dots N\}$, where $x_i \in \mathbb{R}^3$ are the point locations, the contact location a^{loc} is chosen from among the points in $\mathcal{X}^{obj}$. Thus, a^{loc} is defined over a discrete action space of dimension N. We assume a collision-free motion planner to move the gripper to the contact location a^{loc} (see Appendix A.1.6 for details). In contrast, the motion parameters a^m, which define how the gripper interacts with the object after contact, are defined in a continuous action space. Furthermore, we define a^m as the end-effector delta position movement from the contact position, hence $a^m \in \mathbb{R}^3$. Our experiments show that translation-only movements are sufficient to enable complex 6D object manipulation in our task. We also include

Figure 2.2: Illustration of our action space.

additional experiments on extending motion parameters to enable rotations in Appendix A.3.4.

The proposed action representation has two benefits compared to previous work. First, it is **temporally-abstracted**. We "abstract" a sequence of lower-level gripper movements of approaching the contact and executing the motion parameters into one action decision step in the RL problem definition. Compared to the common action space of end-effector delta movements [28, 53, 56, 33], the agent with our action space can avoid wasting time learning how to move in free space and instead focus on learning contact-rich interactions. Second, it is **spatially-grounded** since the agent selects a contact location from the observed object point cloud.

2.5.2 Hybrid RL Algorithm

The proposed action space is a hybrid discrete-continuous action space: the contact location a^{loc} is discrete while the motion parameters a^m are continuous. We propose a way to adapt existing off-policy algorithms designed for continuous action spaces [8, 12] to this hybrid action space. First, consider the simpler case of an action space that only has the continuous motion parameters a^m. In this case, we can directly apply existing off-policy algorithms as described in Section 2.3. Given the observation s (which is the point cloud $\mathcal{X}$ in our task), we can train an actor to output the motion parameters $\pi_\theta(s) = a^m$. Similarly, the critic $Q_\phi(s, a^m)$ outputs the Q-value given the observation s and the motion parameters a^m; the Q-value can be used to update the actor according to Eqn. 2.3.

To additionally predict the contact location a^{loc}, we also need the policy to select a point among the discrete set of points in the object point cloud $\mathcal{X}^{obj}$. Our insight is that we can embed such a discrete component of the action space into the critic by training the critic to output a per-point Q-value Q_i for each point x_i over the entire point cloud. The Q-value at each point on the object represents the estimated return after selecting this point as the contact location. These Q-values can thus be used not only to update the actor, but also to select the contact location as the point with the highest Q-value. Additionally, we train

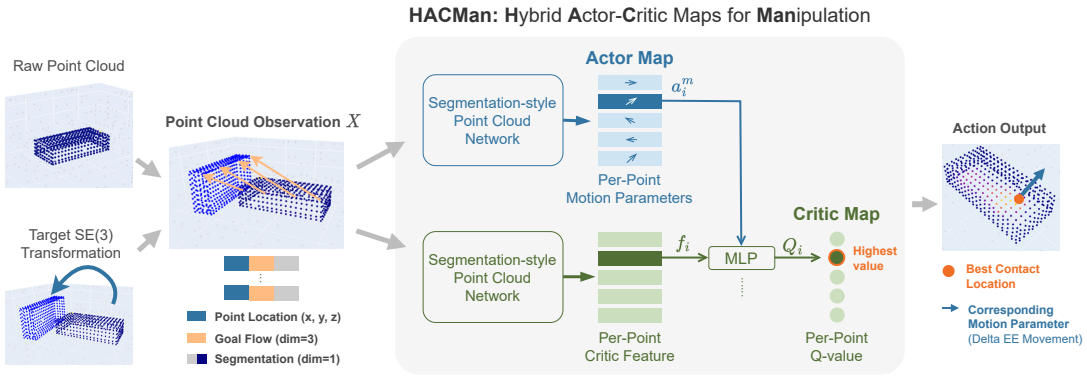


Figure 2.3: An overview of the proposed method. The point cloud observation includes the location of the points and point features. The goal is represented as per-point flow of the object points. The actor takes the observation as input and outputs an **Actor Map** of per-point motion parameters. The Actor Map is concatenated with the per-point critic features to generate the **Critic Map** of per-point Q-values. Finally, we choose the best contact location according to the highest value in the Critic Map and find the corresponding motion parameters in the Actor Map.

the actor to also output per-point motion parameters a_i^m for each point x_i. If point x_i is selected as the contact location a^{loc}, then the motion parameters at this point a_i^m will be used as the gripper motion after contact.

The overall architecture is shown in Fig. 2.3. The actor π_θ receives as input the point cloud observation and outputs per-point motion parameters $\pi_\theta(\mathcal{X}) = \{a_i^m = \pi_{\theta,i}(\mathcal{X}) \mid i = 1 \dots N\}$. We call this per-point output an **"Actor Map"**. The critic also receives as input the point cloud observation. It first calculates per-point features $f(\mathcal{X}) = \{f_i \mid i = 1 \dots N\}$. The critic then concatenates each per-point feature f_i (Section 2.4) with the corresponding per-point motion parameter a_i^m and inputs the concatenated vector to an MLP. The output of the MLP is a per-point Q-value: $Q_i = Q_\phi(f_i, a_i^m)$, which scores the action of moving the gripper to location x_i and executing motion parameters a_i^m. We call this per-point output a **"Critic Map"**. In this way, the critic is able to reason jointly about the contact location (via the feature f_i) as well as the motion parameters a_i^m. In our implementation, both the actor and the critic use segmentation-style PointNet++ architecture [34] (Appendix A.2).

At inference time, we can select the point x_i within the object points $\mathcal{X}^{obj}$ with the highest Q-value Q_i as the contact location and use the corresponding motion parameters a_i^m. For exploration during training, we define a policy π^{loc} which selects the contact location based on a softmax of the Q-values over all of the object points. The probability of a point x_i being selected as the contact location is thus given as:

$$\pi^{loc}(x_i \mid s) = \pi^{loc}(x_i \mid \mathcal{X}) = \frac{\exp(Q_i/\beta)}{\sum_{k=1,\dots,N} \exp(Q_k/\beta)}. \tag{2.4}$$

β is the temperature of the softmax which controls the exploration of the contact location. Note that the background points $\mathcal{X}^b$ are included in the observation $s = \mathcal{X} = \{\mathcal{X}^b, \mathcal{X}^{obj}\}$, but are excluded when choosing the contact location. We modify the update rules of the off-policy algorithm for this hybrid policy. Given $s = \mathcal{X}$, we first define the per-point loss for updating the actor $\pi_{\theta,i}(s)$ at location x_i according to Eqn. 2.3:

$$J_i(\theta) = -Q_\phi(f_i, a_i^m) = -Q_\phi(f_i, \pi_{\theta,i}(s)). \tag{2.5}$$

f_i is the feature corresponding to point x_i. The total objective of the actor is then computed as an expectation over contact locations:

$$J(\theta) = \mathbb{E}_{x_i \sim \pi^{loc}}[J_i(\theta)] = \sum_i \pi^{loc}(x_i \mid s) \cdot J_i(\theta). \tag{2.6}$$

$\pi^{loc}(x_i \mid s)$ is the probability of sampling contact location x_i, defined in Eq. 3.4. The difference between Eqn. 2.6 and the regular actor loss in Eqn. 2.3 is that we use the probability of the discrete action to weight the loss for the continuous action. To take into account π_{loc} during the critic update, the Q-target y_t from Eqn. 2.2 is modified to be:

$$y = r_t + \gamma \mathbb{E}_{x_i \sim \pi^{loc}}[Q_\phi(f_i(s_{t+1}), \pi_{\theta,i}(s_{t+1}))]. \tag{2.7}$$

2.5.3 Representing the Goal as Per-Point Goal Flow

As described in Section 2.4, the objective of our task is to move an object to a given goal pose. Instead of concatenating the goal point cloud to the observed point cloud [3, 4], we represent the goal as per-point "goal flow": Suppose that point x_i in the initial point cloud corresponds to point x_i' in the goal point cloud; then the goal flow is given by $\Delta x_i = x_i' - x_i$. The goal flow Δx_i is a 3D vector which is included as the feature of the point cloud observation (concatenated with a segmentation label, resulting in a 4-dimensional feature vector). This flow representation of goal is also used to calculate the reward and success rate for the pose alignment task (Appendix A.1.3). In the real robot experiments, we estimate the goal flow using point cloud registration (Appendix A.4). Our ablation experiments suggest that utilizing the flow representation of the goal drastically enhances performance compared to directly concatenating the goal point cloud (Appendix A.3.1).

2.6 Experiment Setup

We evaluate our method on the 6D object pose alignment task as described in Section 2.4. The objective is to perform a sequence of non-prehensile actions (i.e. pushing, flipping) to move the object to a given goal pose. In this section, we describe the task setup in simulation, used for training and simulation evaluation (see Section 2.8 for real robot experiments).

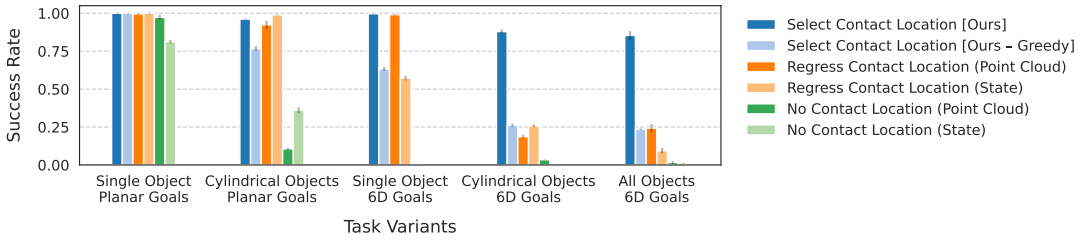


Figure 2.4: Baselines and Ablations. Our approach outperforms the baselines and the ablations, with a larger margin for more challenging tasks on the right. Success rates for simple tasks - pushing a single object to an in-plane goal - are high for all methods, but only HACMan achieves high success rates for 6D alignment of diverse objects.

Task Setup. The simulation environment is built on top of Robosuite [58] with MuJoCo [46]. We include 44 objects with diverse geometries from Liu et al. [22]. Details and visualizations of the object models are included in Appendix A.1.1. The object dataset is split into three mutually exclusive sets: training set (32 objects), unseen instances (7 objects) and unseen categories (5 objects). The 7 unseen instances consist of objects from categories included in the training set, whereas the 5 objects in "unseen categories" consist of novel object categories. An episode is considered a success if the average distance between the corresponding points of the object and the goal is less than 3 cm. More details on our simulation environment setup can be found in Appendix A.1.

Task Variants. To analyze the limitations of different methods, we design the object pose alignment task with varying levels of difficulty. We consider three types of object datasets: An **All Objects** dataset that includes the full object dataset, a **Cylindrical Objects** dataset consisting of only cylindrical objects, and a **Single Object** dataset consists of just a single cube. We also try different task configurations: In the **Planar goals** experiments, the object starts from a *fixed* initial pose at the center of the bin, and the goal pose is a randomized *planar translation* of the starting pose. In the **6D goals** experiments, both the object initial pose and goal pose are *randomized SE(3)* stable poses, not limited to planar transformations. This task requires SE(3) object movement to achieve the goal which imposes challenges in spatial reasoning. These different task variations are used to show at what level of difficulty each of the baseline methods stop being able to complete the task.

2.7 Simulation Results

In this section, we demonstrate the effectiveness of HACMan compared to the baselines and ablations. Fig. 2.4 summarizes the performance of each method after being trained with the same number of environment interactions. The training curves, tables, and additional results can be found in Appendix A.3. Implementation details of all methods are included in Appendix A.2.

	Spatially Grounded?	Temporally Abstracted?
Select Contact Location [Ours]	✓	✓
Regress Contact Location	×	✓
No Contact Location [3, 4, 56, 35, 53, 28]	×	×

Table 2.1: Features of the proposed action representation compared to the baselines.

Effect of action representations. We compare our method with two alternative action representations, summarized in Table 2.1. In **Regress Contact Location**, the policy directly regresses to a contact location and motion parameters, instead of choosing a contact point from the point cloud as in HACMan. The **No Contact Location** baseline directly regresses to a delta end-effector movement at each timestep. For every action, the robot continues from its position from the previous action, instead of first moving the gripper to a selected contact location. This is the most common action space in manipulation [3, 4, 56, 35, 53, 28]. As input for these two baselines, we use either point cloud observations or ground-truth state, establishing four baselines in total. The baseline that regresses motion parameters from point cloud observations is a common action representation used in prior work in RL from point clouds such as Qin et al. [35]. The baseline that regresses motion parameters from ground-truth state is the most common approach in prior work, such as in Zhou and Held [56] as well as the teacher policies in Chen et al. [3, 4] (see Appendix A.6 for more discussion).

As shown in Fig. 2.4, these baseline action representations struggle with the more complex task variants. For the most challenging task variant "All Objects + 6D goals," our method achieves a success rate 61% better than the best baseline (see Table A.3 for numbers). As mentioned in Section 2.5, the proposed action representation benefits from being *spatially-grounded* and *temporally-abstracted*. The comparison against the baselines demonstrates the importance of each of these two features (Table 2.1). The "Regress Contact Location" baseline still benefits from being temporally-abstracted because the gripper starts from a location chosen by the policy at each timestep; however, this action representation is not spatially-grounded because it regresses to a contact location which might not be on the object surface, unlike our approach which selects the contact location among the points in the point cloud observation. Thus, the "Regress Contact Location" baseline suffers from training difficulties with more diverse objects (last two variants in Fig. 2.4). The "No Contact Location" baseline is neither spatially-grounded nor temporally-abstracted; it follows the usual approach from prior work [3, 4, 56, 35, 53, 28] of regressing an end-effector delta motion at each timestep. While this is the most common action space in prior work, it has close to zero performance with 6D goals.

Effect of Multi-step Reasoning. To test the necessity of multi-step reasoning for the pose alignment task, we experiment with a **"Greedy"** version of HACMan by setting the discount factor γ in the RL algorithm to $\gamma = 0$. This

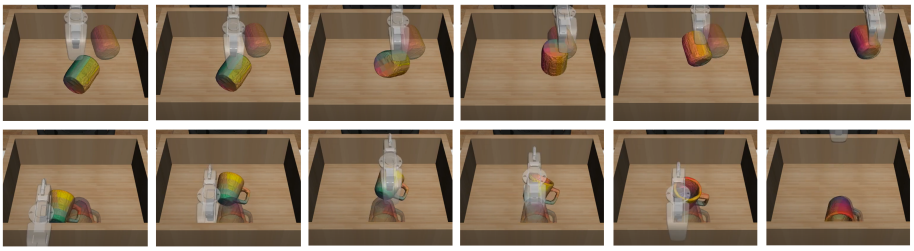


Figure 2.5: Qualitative results for the object pose alignment task. HACMan shows complex non-prehensile behaviors that move the object to the goal pose (shown as the transparent object).

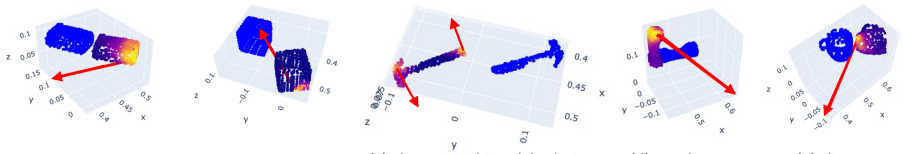


Figure 2.6: Goal-conditioned Critic Maps. Blue: goal point cloud. Color map: observed object point cloud. Lighter colors indicate higher Critic Map scores. Red arrows: motion parameters at a selected location. The policy uses different contact locations based on object geometries and goals.

forces the algorithm to optimize for greedy actions for each step. Using RL for multi-step reasoning is one of the important differences between our method and previous work such as Where2Act [25] and UMPNet [51] which optimize for one-step contact locations. Fig. 2.4 indicates that greedy actions might work for planar goals, but suffer from poor performance for 6D goals that requires multi-step non-greedy interactions. For example, the last row in Fig. 2.5 shows an example of our method pushing the object away from the goal position to prepare for flipping it to the correct orientation, demonstrating non-greedy behavior. In contrast, we find that the greedy ablation often results in local optima of only trying to match the object position but not its orientation.

Generalization to unseen objects. The evaluation of our method over unseen objects with 6D goals is summarized in Table 2.2. Our method generalizes well to unseen object instances and unseen categories without a performance

Object Set Split	Success Rate	# of Objects
Train	0.833 ± .018	32
Train (Common Categories)	0.887 ± .024	13
Unseen Instance (Common Categories)	0.891 ± .033	7
Unseen Category	0.827 ± .047	5

Table 2.2: Generalization to unseen objects.

drop. When we increase the maximum episode length from 10 steps to 30 steps, our method achieves 95.1% success on unseen categories (Appendix A.3.7). Table 2.2 shows that, comparing the same set of object categories ("common categories"), the success rates of the training instances are similar to the unseen instances. The differences in geometry comparing the training objects and the unseen objects are visualized in Appendix A.1.1.

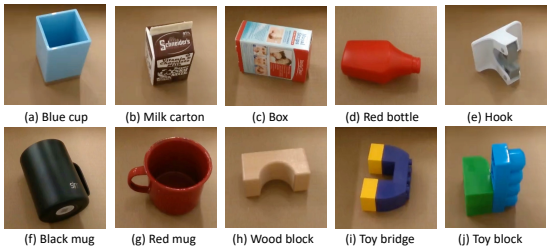
Goal-conditioned Object Affordance and Multimodality. We visualize the Critic Map, which computes the score of each contact location of the object (Fig. 2.6). The Critic Maps capture goal-conditioned object affordances which describe how the object can be moved to achieve the goal. Fig. 2.6(a) and (b) are two scenarios of performing translation object motions for different object heights: For a thin object, the Critic Map highlights the region of the top of the object (dragging) or from the back side of the object (pushing). For a thick object, it prefers to push from the bottom to avoid the object falling over. In Fig. 2.6(c), the hammer needs to be rotated by 180 degrees. The Critic Map predicts a multimodal solution of pushing from either end of the object. Fig. 2.6(d) and (e) show out-of-plane motions of the object of knocking over and flipping up the objects.

Additional Results. We include additional experiment results in Appendix A.3, including additional ablations, extending the motion parameters, cluttered scenes, longer training steps, longer episode lengths, and success rate breakdown for each object category.

2.8 Real robot experiments

In the real robot experiments, we aim to evaluate the ability of the trained policy to generalize to novel objects and execute dynamic motions in the real world. We evaluate the policy with a diverse set of objects with different shapes, surface frictions, and densities (Fig. 2.7). We use random initial poses and random 6D goals (referred to in the previous section as "All Objects + 6D Goals"). For example, the red mug (Fig. 2.7(g)) has goal poses of being upright on the table or lying on the side. An episode is considered a success if the average distance of corresponding points between the object and the goal is smaller than 3 cm; we also mark an episode as a failure if there is a failure in the point cloud registration between the observation and the goal. Implementation details of the real robot experiments can be found in Appendix A.4.

Fig. 2.7 (right) summarizes the quantitative results of the real robot experiments. We run the evaluations without manual reset, which may create uneven numbers of planar versus out-of-plane goals. It achieves 70% success rate on planar goals and 40% success rate on non-planar goals. Non-planar goals are more difficult than the planar goals because they require dynamic motions to interact with the object. A small error in the action may result in large changes in the object movement. Videos of the real robot experiments can be found on the website: `https://hacman-2023.github.io`. The real robot experiments demonstrate that the policy is able to generalize to novel objects in the real



Object Name	Planar Goals	Non-planar Goals	Total
(a) Blue cup	4/7	4/13	8/20
(b) Milk bottle	6/7	10/13	16/20
(c) Box	2/5	10/15	12/20
(d) Red bottle	4/7	0/13	4/20
(e) Hook	5/8	5/12	10/20
(f) Black mug	4/7	0/13	4/20
(g) Red mug	5/7	3/13	8/20
(h) Wood block	6/7	6/13	12/20
(i) Toy bridge	9/10	5/10	14/20
(j) Toy block	2/2	10/18	12/20
Total	47/67	53/133	100/200
Percentage	70%	40%	50%

Figure 2.7: Real robot experiments. HACMan achieves a 50% overall success rate over unseen objects with different geometries and physical properties, with 6D goal poses.

world, despite the sim2real gap of the simulator physics and inaccuracies of point cloud registration for estimating the goal transformation. More discussion can be found in Appendix A.4.

2.9 Limitations

Since the contact location in our action space is defined over the point cloud observation, our method requires relatively accurate depth readings and camera calibration. Further, the contact location is currently limited to the observed part of the object. In addition, for this goal-conditioned task, we represent the goal as per-point flow (Section 2.5.3) which relies on point cloud registration algorithms. Inaccuracies in the registration algorithm sometimes lead to failure cases in real robot experiments. More discussion of the failure cases can be found in Appendix A.4.3. In addition, finetuning the policy on the real robot can potentially improve the success rate further.

2.10 Conclusion

In this work, we propose to learn Hybrid Actor-Critic Maps with reinforcement learning for non-prehensile manipulation. The learned policy shows complex object interactions and strong generalization across unseen object categories. Our method achieves a significantly higher success rate than alternative action representations, with a larger performance gap for more difficult task variants. In addition, it would be interesting to explore alternative 3D representations other than point clouds such as implicit representation [10, 41]. We hope the proposed method and the experimental results can pave the way for future work on more skillful robot manipulation over diverse objects.

Chapter 3

HACMan++: Spatially-grounded Motion Primitives for Manipulation

Although end-to-end robot learning has shown some success for robot manipulation, the learned policies are often not sufficiently robust to variations in object pose or geometry. To improve the policy generalization, we introduce spatially-grounded parameterized motion primitives in our method HACMan++. Specifically, we propose an action representation consisting of three components: *what* primitive type (such as grasp or push) to execute, *where* the primitive will be grounded (e.g. where the gripper will make contact with the world), and *how* the primitive motion is executed, such as parameters specifying the push direction or grasp orientation. These three components define a novel discrete-continuous action space for reinforcement learning. Our framework enables robot agents to learn to chain diverse motion primitives together and select appropriate primitive parameters to complete long-horizon manipulation tasks. By grounding the primitives on a spatial location in the environment, our method is able to effectively generalize across object shape and pose variations. Our approach significantly outperforms existing methods, particularly in complex scenarios demanding both high-level sequential reasoning and object generalization. With zero-shot sim-to-real transfer, our policy succeeds in challenging real-world manipulation tasks, with generalization to unseen objects. Videos can be found on the project website: `https://sgmp-rss2024.github.io`.

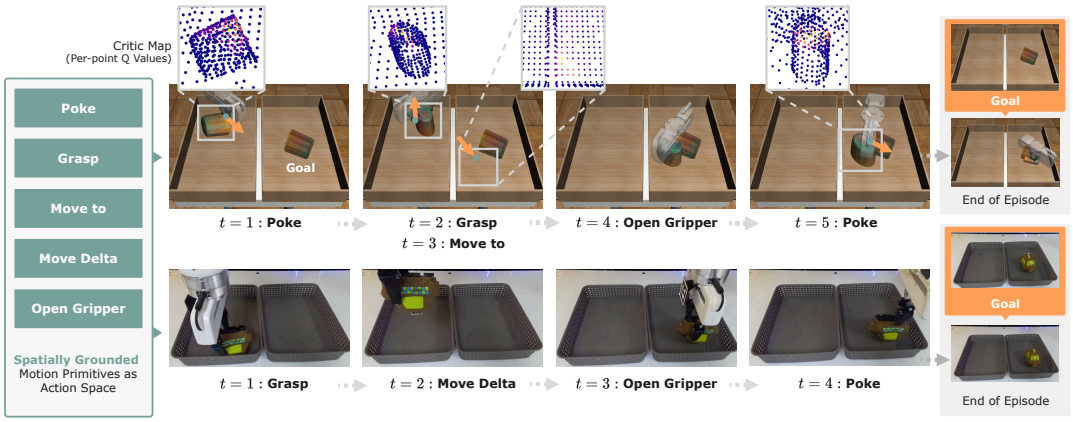


Figure 3.1: Our method consists of a library of parameterized, *spatially-grounded* motion primitives (left), consisting of a primitive type, primitive location (where the primitive will be grounded), and primitive parameters. These three components form the action space for a policy that we train with reinforcement learning. Our method learns to select a sequence of primitives (and their corresponding locations and parameters) to perform a long-horizon manipulation task. In the task shown here, the object is placed in one bin in an initial pose, and it must be moved into a second bin in a target pose. At the top, we visualize the spatial grounding for the selected primitive; for each point we visualize the learned Q-value of selecting that point in the form of heatmaps as the grounding location for each primitive.

3.1 Introduction

Despite recent progress in training manipulation policies with reinforcement learning (RL), it remains challenging to scale RL training to longer-horizon problems with broader task variations [9, 18, 57, 31]. A significant limitation is that most robot manipulation policies reason over the space of granular robot-centric actions, such as gripper or joint movements [28, 53, 58, 56]. These action spaces are highly inefficient for longer-horizon tasks due to exploration, credit assignment, and training stability challenges in deep reinforcement learning [9, 31].

Instead of learning policies over low-level timesteps, the robot should reason about long-horizon manipulation problems with general, reusable primitives. For example, to make coffee, the robot may segment the task into picking up a mug and then placing it under the coffee machine. This process involves decomposing the task into a "grasping" stage followed by a "placing" stage. With a similar idea, prior work has proposed applying a hierarchical structure in robot decisions, such as options or skill primitives [6, 29, 50]. These methods decouple the high-level decisions of "what" to do from the low-level decisions of "how" to execute robot motions. However, our experiments demonstrate that this prior work in using skill primitives shows limited generalization across different object geometries and poses.

We desire a model that can both reason over temporal abstractions (i.e. reasoning about a sequence of parameterized skill primitives) as well as achieve object pose and shape generalization. In this work, we propose to learn manipulation policies with RL using a set of *spatially grounded* motion primitives. The motion primitives consist only of basic manipulation motions such as grasping, placing, or pushing. Each primitive is parameterized by a location selected from the observed point cloud, which the primitive is defined relative to, and a vector of additional parameters defining the details of the gripper motion. For example, "grasping" is parameterized by a location on the object point cloud to grasp and a gripper orientation; "placing" selects a location on the background point cloud and a gripper orientation; "pushing" selects a contact location and a push direction. We also include two "move" primitives to allow for more generic robot motions.

To train a reinforcement learning policy with this action space, we leverage hybrid actor-critic maps [57, 7]. Given a 3D object point cloud, our method trains a critic to output per-point, per-primitive scores, which form a primitive-conditioned "Critic Map." Our method selects the best primitive and the corresponding location with the highest score in the primitive-conditioned critic map. Compared to previous work on 3D hybrid actor-critic maps [57] which is limited to one non-prehensile poking skill, we include a comprehensive set of heterogenous primitives to enable the robot to perform a wider variety of tasks. Another related line of work [7] is demonstrated on only a single task and 4 objects, whereas we demonstrate our approach on 4 different tasks and a wide variety of object geometries.

The contributions of our paper include:

1. A set of **diverse and generic spatially-grounded motion primitives** that can solve a range of complex tasks that could not be solved by prior work.
    - Compared to prior work that uses diverse motion primitives [6, 50], our primitives are spatially-grounded and outperform prior work.
    - Compared to prior work that uses specially-designed spatially-grounded primitives for a single task [7, 57], our primitive set is more generic and applies to a wide range of tasks.
2. An RL training framework that incorporates the primitive selection and spatial-grounding selection using the critic.

Our experimental contributions include:

1. We demonstrate that our method learns complex skills that generalizes over unseen objects, achieving an 89.5% success rate on training objects and an 84.9% success rate on unseen object categories on our Double Bin task.
2. We show that our method significantly outperforms prior work that includes diverse primitives that are not spatially-grounded on diverse simulation tasks.

3. We also perform real robot experiments for a DoubleBin object pose alignment task, which achieves 73% success rate.

In addition to our main experiments, we also show preliminary results of extending the concept of spatially grounded motion primitives to dexterous hand manipulation tasks in Appendix B.2.6, demonstrating the potential for this approach to generalize to other robot morphologies.

3.2 Related Work

Hierarchical Reinforcement Learning. Prior work has integrated a hierarchical structure into reinforcement learning to reduce the challenge of long-horizon reasoning for RL algorithms [45]. In hierarchical reinforcement learning, a high-level policy will communicate with one or more low-level policies to finish the task. However, it can be difficult to jointly optimize both the high and low-level policies [11]. Alternatively, prior work has proposed to first learn a set of low-level skills from an offline dataset [32, 2, 38, 37, 42]. Instead, we follow prior work in the robotics domain and define the low-level policies as commonly used primitives such as grasping, placing, and pushing [6, 29]. We compare our method to other methods that use "skill libraries," including some hierarchical RL methods, explained below.

Skill Libraries. Prior work [59] has specifically designed a set of primitives including approach, contact, fit, align, and insertion, as a skill library. However, this set of primitives is not generalizable to other tasks. Furthermore, it assumes a pre-specified order of primitives to be executed to finish a given task. Another line of work defines the action space of the RL policy based on a more general set of pre-defined parametrized primitives such as RAPS [6], MAPLE [29], and Parameterized DQN [50]. The RL policy learns to automatically chain different primitives together to achieve a task without assuming a fixed order of primitives. This also means that the agent can re-execute primitives when a failure occurs. Our method inherits the benefits of those work in RL policy with parameterized primitives and also differs from them in that we spatially ground the primitives to improve spatial reasoning. We compare our method to these prior methods in the experiments and demonstrated significantly improved performance.

Spatial Action Maps. Spatial action maps connect a dense action representation with visual inputs using segmentation-style models [55, 36, 39, 25, 51, 48]. Our method proposes a novel combination of motion primitives with spatial action maps to incorporate both temporal abstraction and spatial reasoning. Most prior work on spatial action maps has limitations on requiring expert demonstrations for imitation learning [55, 36, 39] or is limited to one-step decisions without sequential reasoning [25, 51, 48]. Our work is the most related to [57, 7]; however, Zhou et al. [57] is limited to one non-prehensile skill (pushing) while we use a set of heterogeneous skills that can be combined to achieve more complex tasks. Feldman et al. [7] uses 2 skills, grasp and shift, and their horizon is limited to 2 (shift and then grasp), whereas our method allows

the algorithm to chain the skills together in different sequences as appropriate for different tasks. Further, Feldman et al. [7] demonstrates their method on a single task with 4 object types, whereas we demonstrate our method on 4 different tasks and a wide variety of object shapes. More detailed discussions between our method and [57, 7] are included in the Appendix B.5.

3.3 Background

A stochastic sequential decision problem can be formalized as a Markov Decision Process (MDP), characterized by (S, A, P, r, γ). Here, S denotes the states, A represents the actions, $P(s_{t+1}|s_t, a_t)$ is the likelihood of transitioning from state s_t to state s_{t+1} given action a_t, and $r(s_t, a_t, s_{t+1})$ is the reward obtained at time t. The goal within this framework is to optimize the return R_t, which is the sum of discounted future rewards, expressed as $R_t = \sum_{i=0}^{\infty} \gamma^i r_{t+i}$. Under a policy π, the expected return for taking action a in state s is described by the Q-function $Q^\pi(s, a) = \mathbb{E}_\pi[R_t|s_t = s, a_t = a]$.

HACMan++ leverages Q-learning-based algorithms for continuous action spaces [20, 8]. These methods are characterized by a policy π_θ with parameters θ, and a Q-function Q_ϕ with parameters ϕ. Training involves collecting a dataset D of state transitions (s_t, a_t, s_{t+1}), with the Q-function's loss formulated as:

$$L(\phi) = \mathbb{E}_{(s_t, a_t, s_{t+1}) \sim D}[(Q_\phi(s_t, a_t) - y_t)^2], \tag{3.1}$$

with y_t being the target value, determined by:

$$y_t = r_t + \gamma Q_\phi(s_{t+1}, \pi_\theta(s_{t+1})). \tag{3.2}$$

The optimization of the policy π_θ is described by the loss function:

$$J(\theta) = -\mathbb{E}_{s_t \sim D}[Q_\phi(s_t, \pi_\theta(s_t))]. \tag{3.3}$$

3.4 Method

Assumptions. We assume that the robot agent records a point cloud observation of a scene $\mathcal{X}$, which may be obtained from one or more calibrated depth cameras. We further assume that this point cloud is segmented into object $\mathcal{X}^{obj}$ and background $\mathcal{X}^b$ components. See Appendix B.1 for details.

To address the challenges of long-horizon manipulation tasks, our method uses a set of parameterized motion primitives, and learns how to both 1) chain these primitives together to achieve a task and 2) select appropriate parameters for the execution of each primitive. Section 3.4.1 defines the structure of the proposed action representation. Section 3.4.2 lists the specific choices of parameterized motion primitives. Section 3.4.3 describes how we train the policy with the proposed action space with the RL algorithm.

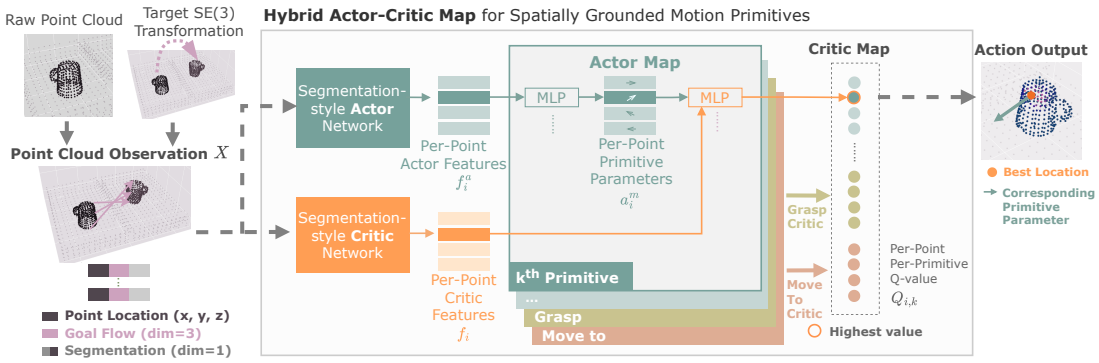


Figure 3.2: Our method processes a point cloud to estimate a set of per-point primitive parameters a_i^m for each point x_i in the point cloud and for each primitive in our primitive set. We then compute a set of "Critic Maps" (one per primitive) which estimate the Q-value $Q_{i,k}$ of using each primitive k, grounded at each point x_i, and parameterized by the estimated primitive parameters a_i^m. We either sample from the Critic Map (during training) or choose the point and primitive with the highest score (during evaluation) for robot execution.

3.4.1 Action Representation

Our action representation comprises three key elements: the primitive type a^{prim}, the primitive location a^{loc}, and the primitive parameters a^m. These components collectively define the "What", "Where", and "How" of each sequential skill execution.

Primitive Type a^{prim} determines the type of primitive the robot will execute, such as poking, grasping, or placing (see the full list in Section 3.4.2). The robot policy aims to learn to perform different tasks by chaining the primitives in appropriate order based on the observations. Each type of primitive is uniquely parameterized to allow for variations in execution, adapting to the specific demands of the task. Once the parameters are specified, these primitives are executed with a low-level controller.

Primitive Location a^{loc} is a selected point of interaction in the scene, chosen from the observed point cloud $\mathcal{X}$. The selected point *grounds* each primitive in the observed world: the robot action will be applied at a location *relative to* the selected point, as defined by the primitive parameters a^m.

Primitive Parameters a^m detail how the robot will execute the chosen primitive at the selected location a^{loc}. It includes aspects like gripper orientation while approaching the object, an offset with respect to the chosen primitive location, and post-contact movement. Details are primitive-type-dependent and are described below.

3.4.2 Parameterized Motion Primitives

We use five distinct and generic motion primitives, that collectively satisfy the needs of a wide range of manipulation tasks, following the primitive designs from previous work [6, 29]. Each primitive has its own specific parameters described below. More details of the motion parameters for each primitive can be found in the Appendix B.1.

Poke: This primitive applies a non-prehensile poking motion to the target object [57, 54, 7, 1]. The robot moves the fingertip of the gripper to the selected primitive location a^{loc} on the object as the initial contact point (see Appendix B.1 for details). The motion parameters a^m consists of two parts: 1) the 2D gripper orientation while approaching the initial contact point, and 2) parameters that describe the poking motion after the gripper reaches the initial contact point on the object, defined as a 3D vector of gripper translation.

Grasp: This primitive grasps the target object and then lifts it up [27, 43, 7]. The primitive location a^{loc} under the grasp primitive type defines a grasping point on the object. The motion parameters a^m detail the 2D gripper orientation while approaching the grasping point. Upon reaching the grasping point, the gripper closes to grasp the object. It then lifts up by a pre-specified distance (see Appendix B.1).

Move to: This primitive moves the gripper to a location that is defined relative to a point a^{loc} selected from the background point cloud $\mathcal{X}^b$. The primitive parameters a^m contain two parts: 1) the 2D gripper orientation when approaching the location, and 2) a 3D vector defining an offset from the selected location a^{loc}; the target point for the gripper to move to is given by the selected location a^{loc} plus this offset. The selected location a^{loc} grounds this motion on the point cloud, whereas the added offset gives the robot more flexibility in where to move. To speed up exploration, we restrict the primitive location a^{loc} to be selected from the background points and we only execute this primitive when the gripper is already grasping an object.

Open Gripper: This primitive opens the gripper. The selected location a^{loc} has no influence on the action, and this primitive does not require any motion parameters.

Move delta: To account for any nuanced movements that are difficult to achieve with the above primitives, we include the "Move delta" primitive to move the gripper by a 3D delta movement and 2D orientation. Motion parameters for this primitive specify a delta translation and rotation of the gripper. We restrict this primitive to only be selected when the robot is already grasping an object.

3.4.3 Hybrid RL Algorithm

HACMan++ integrates a multi-primitive approach with existing Q-learning-based RL algorithms [8, 12, 20]. Our action space includes both discrete and

continuous components: the primitive type a^{prim} is selected from K primitives; for each primitive type, the primitive location a^{loc} is selected from N points from the observed point cloud; whereas the motion parameters a^m are a vector of continuous values.

The overall architecture of our approach is shown in Figure 3.2. The agent receives as input a point cloud observation of size N. We first use a segmentation-style point cloud network to output per-point actor features f_i^a for each point x_i. These features are shared across the K different primitives. We then input each of these features into a per-primitive MLP to output motion parameters $a_{i,k}^m$ for each point x_i and each primitive k. We refer to these outputs as an "Actor Map."

Our method also extracts per-point critic features f_i for each point x_i through a segmentation-style point cloud critic feature extractor. These features are shared across the K different primitives. The per-point motion parameters $a_{i,k}^m$ are then concatenated with per-point critic features f_i and input into a multi-layer perceptron (MLP) to calculate per-point Q values $Q_{i,k}$ for each point x_i and each primitive k; this Q-value represents the effectiveness of executing the k^{th} primitive with the motion parameters $a_{i,k}^m$ at the primitive location x_i. The above procedure generates a "Critic Map" with a total of KN Q-values across all points and all primitives (see Figure 3.2).

The optimal action is chosen by selecting the highest Q-value $Q_{i,k}^{max}$ from the critic map, which corresponds to primitive type k, primitive location x_i, and motion parameters $a_{i,k}^m$. During training, the policy selects primitive types and locations by sampling from a softmax distribution over Q-values to balance exploration and exploitation, formalized as:

$$\pi^{discrete}(k, x_i \mid s) = \frac{\exp(Q_{i,k}/\beta)}{\sum_{k=1}^{K}\sum_{i=1}^{N}\exp(Q_{i,k}/\beta)} \tag{3.4}$$

where β is a temperature parameter modulating the softmax function, guiding the agent's exploratory behavior.

The Q-function is updated according to the bellman equation (Equation 3.1) following TD3 [8]. To update the primitive parameters $a_{i,k}^m$, we similarly follow the TD3 algorithm [8]: If we define the actor $\pi_{i,k}^{\theta}(s)$ as the function parameterized by θ that maps from the observation s to the action parameters $a_{i,k}^m$ for a given point x_i and primitive k, then the loss function for this actor is given by:

$$J(\theta) = -Q_\phi(f_i, a_{i,k}^m) = -Q_\phi(f_i, \pi_{i,k}^{\theta}(s)), \tag{3.5}$$

where Q_ϕ is the critic network and f_i is the critic feature of the point x_i.

To assist the network in understanding the relationship between the observation and the goal, we compute the correspondence between the points in the observation and the points in the goal (see Appendix B.1 for details). For every point in the observation, we append to the input a 3-dimensional vector indicating the delta to its corresponding goal location, which we refer to as "goal flow" (see Figure‘3.2).

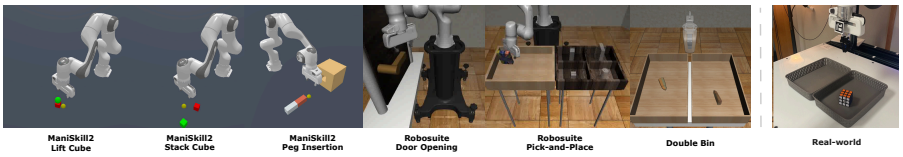


Figure 3.3: We evaluate our method on multiple object manipulation tasks that require picking, placing, and poking objects. From left to right, we show the six simulation tasks: ManiSkill2 Lift Cube, ManiSkill2 Stack Cube, ManiSkill2 Peg Insertion, Robosuite Pick-and-Place, Robosuite Door Opening, and a customized Robosuite DoubleBin environment. We also show our real-world experiment setup which mimics the DoubleBin simulation environment.

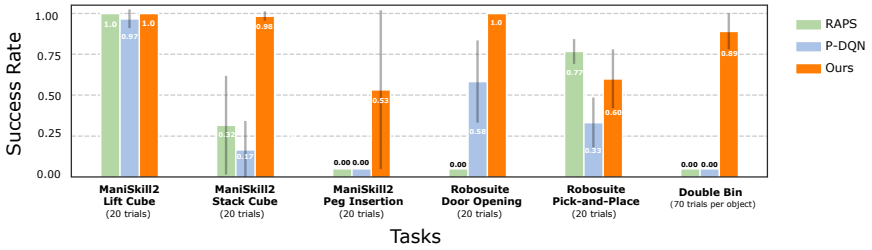


Figure 3.4: Performance of our method compared to baselines RAPS [6] and P-DQN [50] on six different tasks. For all the *ManiSkill* tasks and *Robosuite* tasks, we report the success rate averaged over 20 trials. For *DoubleBin* tasks, we report the average success rate over 32 objects, each tested with 70 trials. These baselines use the same skill primitives as our approach but they are not spatially grounded, e.g. they do not ground the primitives on a point selected by the policy from the observed point cloud.

3.5 Experimental Setup

We evaluate our method on three ManiSkill tasks (Sec. 3.5.1), two Robosuite tasks (Sec. 3.5.2), as well as a DoubleBin task (Sec. 3.5.3) as illustrated in Figure 3.3. This section outlines the setup, objective, and reward function for each task.

3.5.1 ManiSkill Tasks

We evaluate our method with three tasks from ManiSkill [28] (Figure 3.3, Left). For these tasks, we train the hybrid actor-critic map with the default reward functions defined in the ManiSkill benchmark [28].

Lift Cube: The agent is tasked with picking up a cube and lifting it to a specified height threshold. The initial cube position and orientation are randomized.

Stack Cube: This task involves stacking a red cube on top of a green cube, requiring precision in alignment. The initial position and orientation of both

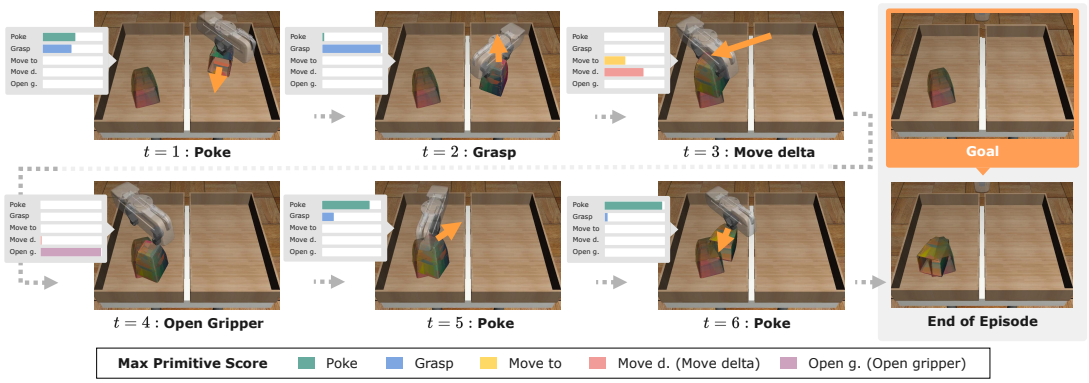


Figure 3.5: A simulation rollout of our policy. The goal is shown in the top right, and also overlayed on each observation. At each step, we visualize the scores that we assign to each of the primitives. We also visualize the selected primitive location and parameters (orange arrow). As shown, our method learns to chain a sequence of grounded primitives to accomplish a challenging long-horizon manipulation task.

cubes are randomized.

Peg Insertion: This task involves inserting a peg horizontally into a hole in a box. As the original ManiSkill paper [28] reports a 0 success rate on this task, we slightly simplify this task by removing the variations in both the hole's location and the peg's initial pose as well as marginally increasing the clearance of the hole. We compare to baseline approaches with these same environment modifications.

3.5.2 Robosuite Task

We also evaluate our method with two tasks from Robosuite [58](Figure 3.3, Left). For these tasks, we train with the default dense reward functions in the Robosuite benchmark [58].

Pick-and-Place: The task is initialized with one object at a random position in a large single bin and the goal is to place the object into a specified small container on the side. There are four containers in total and four objects, including cube, box, can and milk carton.

Door Opening: A door with a handle is placed in front of a single robot arm in this task. The agent needs to learn to rotate the door handle and open the door.

3.5.3 DoubleBin Task

To further demonstrate the benefits of spatial grounding, we design the DoubleBin task (Figure 3.3, Right). It is built on top of Robosuite [58] with the Mujoco simulator [46]. Compared to the ManiSkill tasks, it requires longer

	Spatially Grounded	Primitive Selection
Ours	✓	Argmax of Critic Scores
P-DQN [50]	×	Argmax of Critic Scores
RAPS [6]	×	Argmax of Actor Probabilities

Table 3.1: Differences Between Our Method and Baselines.

horizon reasoning and has more object shape variations. Each episode starts with two bins with one object in a randomly selected bin. The objective of the robot agent is to perform a sequence of motions to manipulate the object to a pre-specified 6D goal pose in the opposite bin. This resembles a common scenario in warehouse automation and assembly lines. The reward function is the average norm of the point cloud correspondence vectors between the object's current state and goal state state (see Appendix B.2).

At each episode, we sample one object from a set of 32 objects with diverse geometries for training. The agent needs to dynamically adapt its manipulation strategies to suit the unique geometry of each object. We evaluate our method on 7 unseen object instances (from training object categories) and 5 objects from unseen object categories.

3.6 Simulation Results

In our simulation experiments, we aim to answer the following questions:

- Do spatially grounded primitives enable better performance in high precision tasks than previous methods?
- Does our method reasonably select appropriate primitives at each step from a set of primitives and strategically compose them together to solve long-horizon tasks?
- Does the learned policy generalize to unseen objects?

The comparison between our method and baselines over 4 tasks is reported in Figure 3.4. The details of the training and evaluation procedures can be found in the Appendix B.2.

Effect of Spatial Grounding. To demonstrate the benefits of spatial grounding, we compare our method to two baselines, P-DQN and RAPS [6, 50]. Both of the baselines use parameterized primitives as the action space of their RL policies, but the primitives are **not** spatially-grounded. For primitives that involve location parameters, both of the baselines directly regress the location parameters, instead of selecting a location from the observed point cloud as in our method. P-DQN selects primitives based on the critic scores of each primitive type (rather than the critic scores of each primitive type and *location* in our method), while RAPS directly outputs both action probabilities and the primitive parameters from the actor. Table 3.1 highlights the differences between our

Object Set Split	Success Rate (10 steps)	Success Rate (20 steps)	Success Rate (30 steps)	# of Objects
Train	0.676 ± .010	0.845 ± .010	0.892 ± .010	32
Train (Common Categories)	0.746 ± .020	0.903 ± .016	0.937 ± .011	13
Unseen Instance (Common Categories)	0.737 ± .020	0.903 ± .023	0.952 ± .023	7
Unseen Category	0.601 ± .003	0.784 ± .027	0.849 ± .003	5

Table 3.2: Generalization to Unseen Objects. HACMan++ shows strong generalization to previously-unseen instances of classes in the training data, and even generalizes well to unseen object categories.

method and these baselines. A more detailed description of the implementation of the baselines can be found in Appendix B.1.

The results are shown in Fig. 3.4. Although these baselines perform well on the easiest task (**Lift Cube**), they struggle with the other tasks which require more precise spatial reasoning (**Stack Cube** and **Peg Insertion**) and/or generalization to object shape variations (**DoubleBin**). For Robosuite tasks, *Pick-and-Place* 1) does not require precise placing since the goal can be at any position inside the container 2) and does not require generalization to object shapes because there are limited geometries (4 objects compared to 32 objects in the *Double Bin* task). The *Door Opening* task, on the contrary, requires more geometric reasoning so our method outperforms the baselines by a large threshold. In general, ours is the *only* method that maintains reasonable performance across the six different tasks. Note that the manipulation tasks in our experiments require higher precision than the tasks reported in RAPS [6]. These results demonstrate the benefits of spatial grounding for precise manipulation tasks.

Effect of Primitive Chaining The proposed method is able to chain primitives together in appropriate orders to solve different tasks, without requiring a pre-specified sequence of primitive types [40]. For example, in the DoubleBin task, our method learns to chain both the prehensile and non-prehensile primitives together with different orders under different situations. If the initial pose of the object is impossible for top-down grasping, our policy will first poke the object to a graspable pose as shown in the first step in Figure 3.5. After grasping, it also learns to use a move primitive (e.g. *Move to* or *Move delta*) to relocate the object to the other bin (e.g. step 3 in Figure 3.5 and select *Open Gripper* to release it. In some cases, the policy may perform a few *Poke* primitives again to move the object into the correct pose if necessary (e.g., step 5-6 in Figure 3.5). This process of moving the objects across bins and into the correct poses is not possible without chaining the primitives strategically. Similarly, our method also demonstrates such strategic reasoning in ManiSkill Tasks, e.g., chaining grasp with multiple move primitives to complete the Peg Insertion Task.

Generalization to Unseen Objects. To demonstrate the generalization capabilities of the proposed method, we evaluate our model on the DoubleBin task with unseen objects, whose results are summarized in Table 3.2 and Figure 3.6. We report the performance averaged over $70 \times n$ trials, where n refers to the

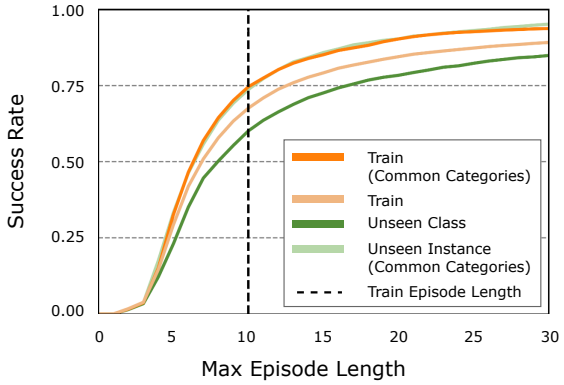


Figure 3.6: Success rate as a function of the episode length, for training objects (all), training objects from categories with many instances (common categories), unseen instances from those same common categories, and for unseen object classes. We train with an episode length of 10 but evaluate with varying episode lengths up to 30.

number of objects in the evaluation set. The overall success rate for achieving the target 6D goal transformation on the training objects is 89.2% when the policy is evaluated with an episode length of 30. We evaluate the generalization capabilities of the model in three settings. First, we evaluate our method on unseen object instances. These evaluation objects are within the training object categories, but the exact object models are unseen. The unseen instances are randomly selected from the most common categories of objects from the full object dataset (for which there are many object instances), including plant container, salt shaker, pencil case, pill bottle, bottle, canister, and can. The performance of the model on these common object categories is at 93.7% for seen object instances. An evaluation on unseen instances from these same categories has nearly the same performance (95.2%), demonstrating our model's ability to adapt to different object geometries within these categories. We also evaluate our method on objects in unseen categories that were not included in training (e.g., lunch bag) and achieve a success rate of 84.9%, demonstrating the ability of our model to generalize to novel shapes. A visualization of the training and unseen testing objects can be found in the Appendix B.2.

Furthermore, we conducted additional experiments with varying maximum episode length for evaluation; the results in Figure 3.6 show that the success rate increases with a longer episode length. An episode length of 10 is used for training, so this figure also demonstrates the ability of our model to continue to improve performance beyond the training episode length.

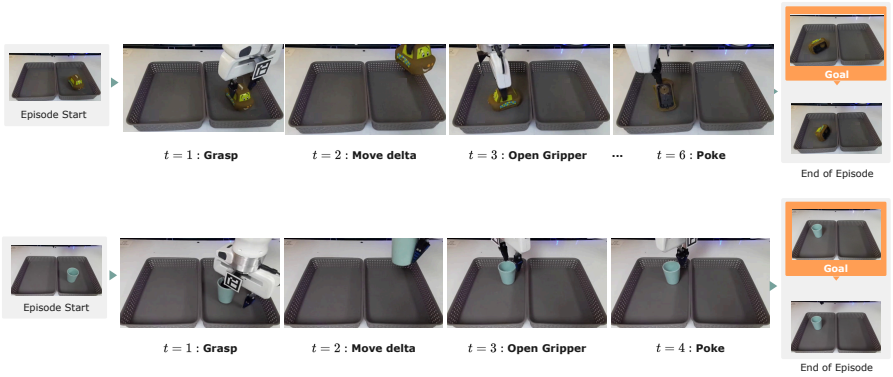


Figure 3.7: Two examples of real-world rollout of our policy. Our method learns to chain a sequence of actions to lift the object, move it across to the other bin, release the object, and then poke it to match the target pose more precisely. The first row shows the rollout of the car (toy) with a SE(3) goal. The second row shows rollout of the cup with a translation goal.

3.7 Real-World Experiments

We perform evaluations on the real world DoubleBin task with the policy trained in simulation as discussed in the previous sections. At the beginning of each episode, we place the object at a random pose in a randomly chosen bin. We also specify a goal SE(3) transformation, which can be either in the same bin as the initial object pose or in the opposite side bin. Among all the objects we are testing, Rubik's Cube, Bowl, Cup, Tennis are evaluated with the translation goals because of their rotation-symmetric shape. At each step, we perform point cloud registration to compute the correspondence from the current observation to the goal (see details in Appendix B.4).

Similar to our simulation environment, an episode is deemed a success when the mean distance between each observation point on the object and its corresponding goal point is less than 3 cm. We set a maximum episode length of 15 time steps (each time step corresponds to one primitive action).

In our experiments, we select six objects with different materials and geometries, as shown in Figure 3.8. Figure 3.3 shows the real-world experiment setup. Figure 3.7 demonstrates an example real-world trajectory rollout. Table 3.3 shows the quantitative evaluation results. Our method is able to achieve an overall 73% success rate.

3.8 Conclusion

In this work we present spatially grounded motion primitives for robot manipulation tasks, leveraging hybrid actor-critic maps with reinforcement learning.

Figure 3.8: Real-world Objects. From left to right, the six objects are: *Car (Toy)*, *Cardboard*, *Tennis*, *Cup*, *Rubik's Cube* and *Bowl*.

Object	**Same Side Goal**	**Opposite Side Goal**	**Subtotal**
Rubik's Cube	14/20	12/20	26/40
Bowl	19/20	16/20	35/40
Cup	11/20	14/20	25/40
Tennis	19/20	19/20	38/40
Cardboard	11/20	15/20	26/40
Car (Toy)	12/20	14/20	26/40
Subtotal	86/120	90/120	176/240
Percentage	72%	75%	73%

Table 3.3: Real-World Experiment Results

Our agent learns to chain different spatially-grounded primitives with appropriately selected primitive parameters to complete a task. Our method adapts to diverse manipulation tasks and generalizes to diverse objects, succeeding in tasks that require both high-level sequential reasoning and low-level motion precision - where previous methods have fallen short. The effectiveness of our approach suggests the importance of primitives that are spatially grounded on points in the environment.

Limitations. Our approach to breaking down a manipulation task into motion primitives has shown adaptability across a range of scenarios; nonetheless, there are complexities in designing general primitives to accommodate every task. Although we have added some preliminary experiments exploring other gripper morphology (i.e. the dexterous Shadow hand task in Appendix B.2.6), more exploration is needed to determine the best way to apply spatially-grounded primitives to different gripper designs.

Chapter 4

Conclusion

In this thesis, we have presented novel approaches for learning robotic manipulation skills using reinforcement learning with spatially-grounded action spaces. Our work addresses the challenges of learning complex, long-horizon manipulation tasks with high-dimensional, continuous action spaces and the need for generalization to novel objects and environments.

In our first paper, "HACMan: Learning Hybrid Actor-Critic Maps for 6D Non-Prehensile Manipulation," we introduced a hybrid actor-critic model that maps discrete and continuous actions to the 3D object point cloud. This approach enables the robot to perform complex non-prehensile interactions based on the spatial features of the object, demonstrating strong performance and generalization to unseen objects in tasks such as 6D object pose alignment. The success of this work highlights the importance of spatial reasoning in manipulation tasks and the effectiveness of grounding actions in the observed point cloud.

Building upon the concept of spatially-grounded action spaces, our second paper, "Spatially-Grounded Motion Primitives for Manipulation," extends the framework to include a diverse set of parameterized motion primitives. By defining an action space that includes the type of primitive, the location in the environment where the action is applied, and the parameters of the motion, we enable the robot to perform a wide range of tasks by chaining together these primitives. This approach significantly improves generalization across different objects and tasks, outperforming existing methods in complex manipulation scenarios. The results demonstrate the power of combining spatial reasoning with temporally-extended actions in the form of motion primitives.

The contributions of this thesis have several implications for the field of robotic manipulation. First, our work emphasizes the importance of selecting the right level of abstraction for the action space when applying reinforcement learning to manipulation tasks. By grounding actions in the spatial properties of the environment and using parameterized motion primitives, we can learn policies that are more sample-efficient, generalize better to novel objects, and solve complex, long-horizon tasks. Second, our experiments demonstrate the

effectiveness of learning manipulation skills directly from visual observations, such as point clouds, without the need for explicit state estimation or modeling of the environment dynamics. This suggests that deep reinforcement learning can be a powerful tool for learning adaptive, versatile manipulation policies that can cope with the challenges of unstructured environments.

There are several exciting directions for future research building upon the contributions of this thesis. One direction is to further extend the set of motion primitives and explore more advanced ways of composing them to solve even more complex tasks. Another direction is to investigate methods for learning the motion primitives themselves from data, rather than using hand-designed primitives. This could lead to the discovery of novel, task-specific primitives that are optimized for particular environments or objects. Finally, an important direction is to develop methods for sim-to-real transfer and adaptation, allowing policies learned in simulation to be effectively deployed on real robots.

In conclusion, this thesis has presented novel approaches for learning robotic manipulation skills using reinforcement learning with spatially-grounded action spaces. Our work demonstrates the effectiveness of spatial reasoning and temporally-extended actions in the form of motion primitives for learning complex, long-horizon manipulation tasks with strong generalization to novel objects. We believe that the contributions of this thesis bring us closer to the goal of enabling robots to autonomously learn and adapt manipulation strategies for diverse environments and tasks, and we look forward to future research building upon these ideas.

Bibliography

[1] Pulkit Agrawal, Ashvin V Nair, Pieter Abbeel, Jitendra Malik, and Sergey Levine. Learning to poke by poking: Experiential learning of intuitive physics. *Advances in neural information processing systems*, 29, 2016.

[2] Anurag Ajay, Aviral Kumar, Pulkit Agrawal, Sergey Levine, and Ofir Nachum. Opal: Offline primitive discovery for accelerating offline reinforcement learning. In *Proceedings of (ICLR) International Conference on Learning Representations*, May 2021.

[3] Tao Chen, Jie Xu, and Pulkit Agrawal. A system for general in-hand object re-orientation. *Conference on Robot Learning*, 2021.

[4] Tao Chen, Megha Tippur, Siyang Wu, Vikash Kumar, Edward Adelson, and Pulkit Agrawal. Visual dexterity: In-hand dexterous manipulation from depth. *arXiv preprint arXiv:2211.11744*, 2022.

[5] Xianyi Cheng, Eric Huang, Yifan Hou, and Matthew T Mason. Contact mode guided motion planning for quasidynamic dexterous manipulation in 3d. In *2022 International Conference on Robotics and Automation (ICRA)*, pages 2730–2736. IEEE, 2022.

[6] Murtaza Dalal, Deepak Pathak, and Russ R Salakhutdinov. Accelerating robotic reinforcement learning via parameterized action primitives. *Advances in Neural Information Processing Systems*, 34:21847–21859, 2021.

[7] Zohar Feldman, Hanna Ziesche, Ngo Anh Vien, and Dotan Di Castro. A hybrid approach for learning to shift and grasp with elaborate motion primitives. In *2022 International Conference on Robotics and Automation (ICRA)*, pages 6365–6371. IEEE, 2022.

[8] Scott Fujimoto, Herke Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In *International conference on machine learning*, pages 1587–1596. PMLR, 2018.

[9] Abhishek Gupta, Vikash Kumar, Corey Lynch, Sergey Levine, and Karol Hausman. Relay policy learning: Solving long horizon tasks via imitation and reinforcement learning. *Conference on Robot Learning (CoRL)*, 2019.

[10] Jung-Su Ha, Danny Driess, and Marc Toussaint. Deep visual constraints: Neural implicit models for manipulation planning from visual input, 2022.

[11] Tuomas Haarnoja, Kristian Hartikainen, Pieter Abbeel, and Sergey Levine. Latent space policies for hierarchical reinforcement learning. In *International Conference on Machine Learning*, pages 1851–1860. PMLR, 2018.

[12] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*, pages 1861–1870. PMLR, 2018.

[13] Matthew Hausknecht and Peter Stone. Deep reinforcement learning in parameterized action space. *arXiv preprint arXiv:1511.04143*, 2015.

[14] Stefan Hinterstoisser, Vincent Lepetit, Slobodan Ilic, Stefan Holzer, Gary Bradski, Kurt Konolige, and Nassir Navab. Model based training, detection and pose estimation of texture-less 3d objects in heavily cluttered scenes. In *Computer Vision–ACCV 2012: 11th Asian Conference on Computer Vision, Daejeon, Korea, November 5-9, 2012, Revised Selected Papers, Part I 11*, pages 548–562. Springer, 2013.

[15] Yifan Hou and Matthew T Mason. Robust execution of contact-rich motion plans by hybrid force-velocity control. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 1933–1939. IEEE, 2019.

[16] Wenlong Huang, Igor Mordatch, Pieter Abbeel, and Deepak Pathak. Generalization in dexterous manipulation via geometry-aware multi-task learning. *arXiv preprint arXiv:2111.03062*, 2021.

[17] Andrew Hundt, Benjamin Killeen, Nicholas Greene, Hongtao Wu, Heeyeon Kwon, Chris Paxton, and Gregory D Hager. "good robot!": Efficient reinforcement learning for multi-step visual tasks with sim to real transfer. *IEEE Robotics and Automation Letters*, 5(4):6724–6731, 2020.

[18] Youngwoon Lee, Edward S Hu, and Joseph J Lim. Ikea furniture assembly environment for long-horizon complex manipulation tasks. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6343–6349.

[19] Jacky Liang, Xianyi Cheng, and Oliver Kroemer. Learning preconditions of hybrid force-velocity controllers for contact-rich manipulation. *arXiv preprint arXiv:2206.12728*, 2022.

[20] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[21] Minghua Liu, Xuanlin Li, Zhan Ling, Yangyan Li, and Hao Su. Frame mining: a free lunch for learning robotic manipulation from 3d point clouds. In *6th Annual Conference on Robot Learning*, 2022.

[22] Weiyu Liu, Tucker Hermans, Sonia Chernova, and Chris Paxton. Structdiffusion: Object-centric diffusion for semantic rearrangement of novel objects. *arXiv preprint arXiv:2211.04604*, 2022.

[23] Matthew T Mason. Progress in nonprehensile manipulation. *The International Journal of Robotics Research*, 18(11):1129–1141, 1999.

[24] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[25] Kaichun Mo, Leonidas J. Guibas, Mustafa Mukadam, Abhinav Gupta, and Shubham Tulsiani. Where2act: From pixels to actions for articulated 3d objects. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 6813–6823, October 2021.

[26] Igor Mordatch, Emanuel Todorov, and Zoran Popović. Discovery of complex behaviors through contact-invariant optimization. *ACM Transactions on Graphics (TOG)*, 31(4):1–8, 2012.

[27] Arsalan Mousavian, Clemens Eppner, and Dieter Fox. 6-dof graspnet: Variational grasp generation for object manipulation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 2901–2910, 2019.

[28] Tongzhou Mu, Zhan Ling, Fanbo Xiang, Derek Cathera Yang, Xuanlin Li, Stone Tao, Zhiao Huang, Zhiwei Jia, and Hao Su. Maniskill: Generalizable manipulation skill benchmark with large-scale demonstrations. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*, 2021.

[29] Soroush Nasiriany, Huihan Liu, and Yuke Zhu. Augmenting reinforcement learning with behavior primitives for diverse manipulation tasks. In *2022 International Conference on Robotics and Automation (ICRA)*, pages 7477–7484.

[30] Michael Neunert, Abbas Abdolmaleki, Markus Wulfmeier, Thomas Lampe, Tobias Springenberg, Roland Hafner, Francesco Romano, Jonas Buchli, Nicolas Heess, and Martin Riedmiller. Continuous-discrete reinforcement learning for hybrid control in robotics. In *Conference on Robot Learning*, pages 735–751. PMLR, 2020.

[31] Shubham Pateria, Budhitama Subagdja, Ah-hwee Tan, and Chai Quek. Hierarchical reinforcement learning: A comprehensive survey. *ACM Computing Surveys (CSUR)*, 54(5):1–35, 2021.

[32] Karl Pertsch, Youngwoon Lee, and Joseph Lim. Accelerating reinforcement learning with learned skill priors. In *Conference on Robot Learning (CoRL)*, pages 188–204. PMLR, 2021.

[33] Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. *arXiv preprint arXiv:1612.00593*, 2016.

[34] Charles R Qi, Li Yi, Hao Su, and Leonidas J Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. *arXiv preprint arXiv:1706.02413*, 2017.

[35] Yuzhe Qin, Binghao Huang, Zhao-Heng Yin, Hao Su, and Xiaolong Wang. Dexpoint: Generalizable point cloud reinforcement learning for sim-to-real dexterous manipulation. *Conference on Robot Learning (CoRL)*, 2022.

[36] Daniel Seita, Yufei Wang, Sarthak Shetty, Edward Li, Zackory Erickson, and David Held. ToolFlowNet: Robotic Manipulation with Tools via Predicting Tool Flow from Point Clouds. In *Conference on Robot Learning (CoRL)*, 2022.

[37] Tanmay Shankar and Abhinav Gupta. Learning robot skills with temporal variational inference. In *Proceedings of (ICML) International Conference on Machine Learning*, pages 8624 – 8633, July 2020.

[38] Tanmay Shankar, Shubham Tulsiani, Lerrel Pinto, and Abhinav Gupta. Discovering motor programs by recomposing demonstrations. In *Proceedings of (ICLR) International Conference on Learning Representations*, April 2020.

[39] Mohit Shridhar, Lucas Manuelli, and Dieter Fox. Perceiver-actor: A multi-task transformer for robotic manipulation. In *Proceedings of the 6th Conference on Robot Learning (CoRL)*, 2022.

[40] Anthony Simeonov, Yilun Du, Beomjoon Kim, Francois Hogan, Joshua Tenenbaum, Pulkit Agrawal, and Alberto Rodriguez. A long horizon planning framework for manipulating rigid pointcloud objects. In *Conference on Robot Learning*, pages 1582–1601. PMLR, 2021.

[41] Anthony Simeonov, Yilun Du, Andrea Tagliasacchi, Joshua B. Tenenbaum, Alberto Rodriguez, Pulkit Agrawal, and Vincent Sitzmann. Neural descriptor fields: Se(3)-equivariant object representations for manipulation, 2021.

[42] Avi Singh, Huihan Liu, Gaoyue Zhou, Albert Yu, Nicholas Rhinehart, and Sergey Levine. Parrot: Data-driven behavioral priors for reinforcement learning. In *Proceedings of (ICLR) International Conference on Learning Representations*, April 2020.

[43] Martin Sundermeyer, Arsalan Mousavian, Rudolph Triebel, and Dieter Fox. Contact-graspnet: Efficient 6-dof grasp generation in cluttered scenes. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 13438–13444.

[44] Martin Sundermeyer, Arsalan Mousavian, Rudolph Triebel, and Dieter Fox. Contact-graspnet: Efficient 6-dof grasp generation in cluttered scenes. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 13438–13444, 2021. doi: 10.1109/ICRA48506.2021.9561877.

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

[46] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033. IEEE, 2012. doi: 10.1109/IROS.2012.6386109.

[47] Lirui Wang, Yu Xiang, Wei Yang, Arsalan Mousavian, and Dieter Fox. Goal-auxiliary actor-critic for 6d robotic grasping with point clouds. In *The Conference on Robot Learning (CoRL)*, 2021.

[48] Jimmy Wu, Xingyuan Sun, Andy Zeng, Shuran Song, Johnny Lee, Szymon Rusinkiewicz, and Thomas Funkhouser. Spatial action maps for mobile manipulation. In *Proceedings of Robotics: Science and Systems (RSS)*, 2020.

[49] Yu Xiang, Tanner Schmidt, Venkatraman Narayanan, and Dieter Fox. Posecnn: A convolutional neural network for 6d object pose estimation in cluttered scenes. *Robotics: Science and Systems (RSS)*, 2018.

[50] Jiechao Xiong, Qing Wang, Zhuoran Yang, Peng Sun, Lei Han, Yang Zheng, Haobo Fu, Tong Zhang, Ji Liu, and Han Liu. Parametrized deep q-networks learning: Reinforcement learning with discrete-continuous hybrid action space. *arXiv preprint arXiv:1810.06394*, 2018.

[51] Zhenjia Xu, He Zhanpeng, and Shuran Song. Umpnet: Universal manipulation policy network for articulated objects. *IEEE Robotics and Automation Letters*, 2022.

[52] Kuan-Ting Yu, Maria Bauza, Nima Fazeli, and Alberto Rodriguez. More than a million ways to be pushed. a high-fidelity experimental dataset of planar pushing. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 30–37, 2016. doi: 10.1109/IROS.2016.7758091.

[53] Tianhe Yu, Deirdre Quillen, Zhanpeng He, Ryan Julian, Karol Hausman, Chelsea Finn, and Sergey Levine. Meta-world: A benchmark and evaluation

for multi-task and meta reinforcement learning. In *Conference on Robot Learning (CoRL)*, 2019. URL `https://arxiv.org/abs/1910.10897`.

[54] Andy Zeng, Shuran Song, Stefan Welker, Johnny Lee, Alberto Rodriguez, and Thomas Funkhouser. Learning synergies between pushing and grasping with self-supervised deep reinforcement learning. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4238–4245. IEEE, 2018.

[55] Andy Zeng, Pete Florence, Jonathan Tompson, Stefan Welker, Jonathan Chien, Maria Attarian, Travis Armstrong, Ivan Krasin, Dan Duong, Vikas Sindhwani, and Johnny Lee. Transporter networks: Rearranging the visual world for robotic manipulation. *Conference on Robot Learning (CoRL)*, 2020.

[56] Wenxuan Zhou and David Held. Learning to grasp the ungraspable with emergent extrinsic dexterity. In *Conference on Robot Learning (CoRL)*, 2022.

[57] Wenxuan Zhou, Bowen Jiang, Fan Yang, Chris Paxton, and David Held. Hacman: Learning hybrid actor-critic maps for 6d non-prehensile manipulation. In *Conference on Robot Learning*, pages 241–265. PMLR, 2023.

[58] Yuke Zhu, Josiah Wong, Ajay Mandlekar, Roberto Martín-Martín, Abhishek Joshi, Soroush Nasiriany, and Yifeng Zhu. robosuite: A modular simulation framework and benchmark for robot learning. In *arXiv preprint arXiv:2009.12293*, 2020.

[59] Lars Johannsmeier, Malkin Gerchow, and Sami Haddadin. A framework for robot manipulation: Skill formalism, meta learning and adaptive control. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 5844–5850.

Appendix A

Experiment Details for HACMan

A.1 Simulation Environment

A.1.1 Object dataset preprocessing

We use the object models from Liu et al. [22]. Before importing the object models to MuJoco, we perform convex decomposition using V-HACD (`https://github.com/kmammou/v-hacd`) and generate watertight meshes using Manifold (`https://github.com/hjwdzh/Manifold`). The objects are first scaled to 10 cm according to the maximum lengths along x, y, and z axis. The object sizes are randomized with an additional scale within $[0.8, 1.2]$ for the "All Objects" task variants.

We filter out a part of the objects in the original dataset due to simulation artifacts such as wall penetration and unstable contact behaviors. For example, some of the long and thin objects can be pushed into the walls and bounce back like springs. Some of the objects cannot remain stable on the table. The filtering procedure proceeds as follows: 1) we drop an object with an arbitrary quaternion and translation for 100 times; 2) we calculate the percentage of rollouts where the objects are still unstable after 80 simulation steps; 3) we filter out objects with larger than 10% instability rate. We also filter out flat objects because they are hard to flip. Flat objects are defined as objects for which the ratio between the second smallest dimension to the smallest dimension is larger than 1.5. After filtering, we are left with 44 objects. We split the 44 objects into three datasets: train (32 objects), unseen instances (7 objects), and unseen categories (5 objects). The object models of the three datasets are visualized in Fig. A.1, Fig. A.2 and Fig. A.3 respectively. The **Cylindrical Objects** used in the experiments is a subset of the **All Objects** dataset. **Cylindrical Objects** consist of 9 train objects, 3 unseen instance objects, and 4 unseen category objects.

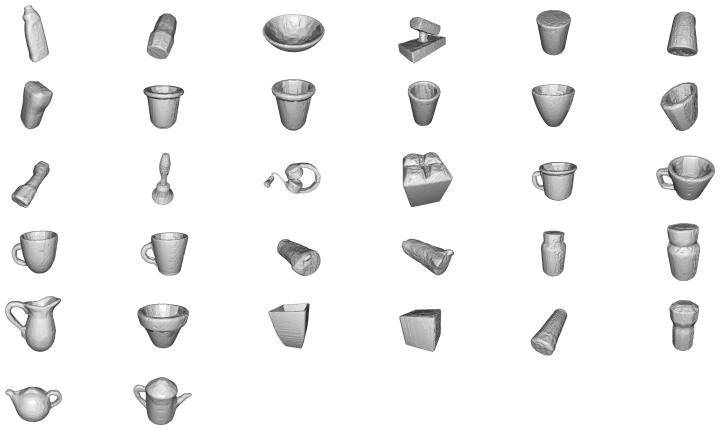


Figure A.1: Training objects. 32 objects used in training.

Figure A.2: Evaluation objects (unseen instance). 7 objects used in unseen instance evaluations. These instances are from the same categories as the training objects.

Figure A.3: Evaluation objects (unseen category). 5 objects used in unseen category evaluations. They come from 4 randomly chosen categories.

A.1.2 Collecting goal poses

To collect stable goal poses, we sample an SE(3) object pose in the air above the center of bin, drop the object in the bin, and then wait until it becomes stable to record the pose. We collect 100 goal poses for each object. At the beginning of each episode, a goal is sampled from the list of stable poses. Furthermore, we randomize the location of the sampled stable goal pose within the bin.

A.1.3 Representing the goal as per-point goal flow

As mentioned in Section 2.5.3, we represent the goal as the "goal flow" of each object point from the current point cloud to the corresponding point in the transformed goal point cloud. In other words, suppose that point x_i in the initial point cloud corresponds to point x_i' in the goal point cloud; then the goal flow is given by $\Delta x_i = x_i' - x_i$. The goal flow Δx_i is a 3D vector which is concatenated to the other features of the input point cloud to represent the goal. In the ablations in Appendix A.3.1, we show that such a representation of the goal significantly improves training, compared to other goal representations such as concatenating the goal point cloud with the observed point cloud.

In order to compute the flow to the goal, we need to estimate correspondences between the observation and the goal. In simulation, we calculate the goal flow based on the ground truth correspondences, based on the known object pose and goal pose. In the real robot experiments, we estimate the correspondences using point cloud registration methods (see Appendix A.4 for details).

Further, for training the RL algorithm, we need some measure of the distance between the initial pose and the goal pose as the reward. Rather than computing a weighted average of the translation and rotation distance (which requires a weighting hyperparameter), we instead define the reward at each timestep r_t as the negative of the average goal flow: $r_t = -\frac{1}{N}\sum_{i=1}^{N} ||\Delta x_i||$, in which $||\cdot||$ denotes the L2 distance and Δx_i is the "goal flow" as defined above. This computation is similar to the "matching score" [14] or "PLoss" [49] used in previous work, except here we use it as a reward function.

A.1.4 Success rate definition

An episode is marked as a success when the average distance of the corresponding points between the object and the goal is smaller than 3 cm. More specifically, this is calculated by the average norm of the per-point goal flow vectors as described in Appendix A.1.3. The episode terminates when it reaches a success. If the episode does not reach a success within 10 steps, it is marked as a failure. We include an additional experiment on longer episode length in Appendix A.3.7.

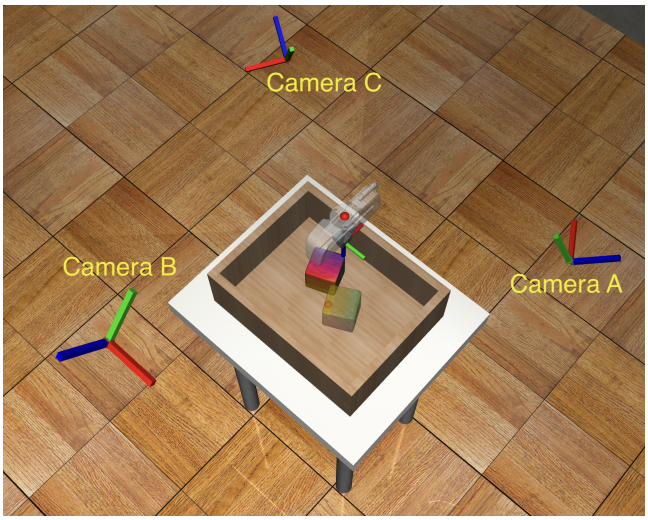


Figure A.4: Camera Locations in Simulation.

A.1.5 Observation

The observation space includes a point cloud of the entire scene $\mathcal{X}$. It contains background points $\mathcal{X}^b$ and object points $\mathcal{X}^{obj}$. Note that we move the gripper to a reset pose after every action before taking the next observation. Thus, the gripper is not observed in the point cloud. To get the point cloud, we set three cameras around the bin (Fig. A.4). The depth readings from the cameras are converted to a set of point locations in the robot base frame and combined.

The object points are then downsampled with a voxel size of 0.005 m $\times$ 0.005 m $\times$ 0.005 m and the background points are downsampled with a voxel size of 0.02 m $\times$ 0.02 m $\times$ 0.02 m. We empirically find that using a slightly denser object point cloud may increase the performance. More specifically, using a 0.005 m $\times$ 0.005 m $\times$ 0.005 m voxel downsample is slightly better than 0.01 m $\times$ 0.01 m $\times$ 0.01 m. We suspect that the policy can perform more precise manipulation of the object with a denser point cloud.

After downsampling, we estimate the normals of the object points using Open3D (`http://www.open3d.org/docs/0.7.0/python_api/open3d.geometry.estimate_normals.html`). The estimated normals will be used during action execution (discussed in the next section).

As mentioned in Section 2.5.3 and Appendix A.1.3, the feature of each point contains the goal flow and the segmentation mask (foreground vs background). The goal flow of the object point is calculated according to Section A.1.3. The goal flow of the background point is set to zero. We obtain the segmentation labels of the object points and the background points from Robosuite[58] during simulation. Details of obtaining segmentation labels in real robot experiments are discussed in Appendix A.4.

A.1.6 Action

As mentioned in Section 2.5, the proposed method uses an action space with a contact location selected from the object points and a set of motion parameters. We discuss the implementation details of executing such an action in the simulation environment in this section. Note that we use a floating gripper as the robot in simulation since we only focus on gripper interactions with the objects.

Once the policy selects a point on the object point cloud, we obtain the corresponding location and estimated normal of the point as described in the previous section. The robot first moves to a "pre-contact" location which is 2 cm away from the contact location along the surface normal. In simulation, this is implemented by directly setting the gripper to the desired pose. In real experiments, we adopt a workaround solution discussed in Appendix A.4. If the gripper encounters a collision at the desired pose, we mark this action as failure and skip the remaining action execution procedure. After reaching the pre-contact location, the gripper will approach to the desired contact location using a low-level controller.

After that, the robot will execute the motion parameters which is the end-effector delta position command that was output by the policy. For the delta actions, we use an action scale of 2 cm. The delta action is executed with an action repeat of 3. We use Operation Space Controller with relatively low gains to allow compliant contact-rich motions with the object. Note that we only consider translation commands (3 dimensions) without rotation in the main experiments because it leads to sufficiently complex object motion for our task. Appendix A.3.4 discusses the effect of including rotation in the gripper movements.

The gripper may not exactly reach the desired location in both sim and real, due to the compliant low-level controller and the gripper geometry. We consider this imperfect execution as a part of the environment dynamics. We do not enforce assumptions such as keeping the contact while executing the motion parameter or avoiding other contact points. Avoiding such assumptions on contacts is a strength of the proposed method compared to some of the classical methods [5, 15].

A.2 Algorithm and Training Details

A.2.1 HACMan (Ours)

HACMan is implemented as a modification on top of TD3 [8] based on the implementation from Stable-Baselines3 (`https://github.com/DLR-RM/stable-baselines3`). We use PointNet++ segmentation-style backbones for both the actor and the critic using the implementation from PyG (`https://pytorch-geometric.readthedocs.io`). Weights are not shared between the actor and the critic. Hyperparameters are included in Table A.1. The actor and the critic use the same network size and the same learning rate. To improve the stability of policy training, we clamp the target Q-values according to an estimated upper and lower bound

of the return for the task. The location policy temperature β is described in Eqn. 3.4.

Hyperparameters	Values
Initial timesteps	10000
Batch size	64
Discount factor (γ)	0.99
Critic update freq per env step	2
Actor update freq per env step	0.5
Target update freq per env step	0.5
Learning rate	0.0001
MLP size	[128, 128, 128]
Critic clamping	[-20, 0]
Location policy temperature (β)	0.1

Table A.1: Hyperparameters.

A.2.2 Baselines

The baselines share the same code framework as HACMan. We discuss their differences with HACMan in this section.

Regress Contact Location. Unlike HACMan, this baseline does not use the object surface for contact point selection. Instead, it directly predicts a location (3 dimensions) and a motion parameter (3 dimensions, represented as a delta end-effector movement). For each action execution, the end-effector moves to the selected location, moves according to the motion parameters, and then resets to the default pose. To improve the performance of this baseline, we project the contact location output to be within the bounding box of the object. Thus, in this baseline, for a location output of the policy, a value of 0 corresponds to the center of the object along a specific dimension, while 1 and -1 represent the maximum and minimum boundaries of the bounding box along that dimension, respectively. Since the location output is no longer a point selected from the object surface, we can no longer use the surface normal vector to determine the approach direction as in HACMan. Instead, this baseline always approaches the location from the top at a height equal to the maximum side length of the object bounding box.

No Contact Location. This baseline does not use the idea of a contact point. Instead, the policy only predicts a motion parameter (3 dimensions, represented as a delta end-effector movement). For each action execution, the end-effector moves according to the motion parameter starting from where it ends after the previous action, without resetting to the default pose. To reduce the exploration difficulties, we make two additional changes: 1) we always start the end-effector right above the object (at a height equal to the maximum side length of the object bounding box) at the beginning of an episode, and 2) we add an extra term to the reward function that penalizes the end-effector for being too far from the object,

$$J_{\text{dist}} = \begin{cases} -\lambda_{dist}(d_{\text{min}} - 0.05),\ d_{min} > 0.05 \text{ m} \\ 0,\ otherwise \end{cases} \tag{A.1}$$

where d_{min} is the minimum distance from the end-effector to the object point cloud vertices, and λ_{dist} is the weight for this reward term.

Point Cloud. Unlike HACMan, these point cloud baselines use PointNet++ classification-style backbones from PyG (`https://pytorch-geometric.readthedoc.io`). For each point cloud, it extracts a single global feature vector instead of per-point feature.

State. In the state-based baselines, the input consists of the pose of the current object, the goal, and optionally the end-effector if the baseline is using "No Contact Location". Each pose is a vector (dim=7) that consists of a position (dim=3) and a quaternion (dim=4). The model concatenates all the pose vectors into a single vector as the input to an MLP.

We report the best results of the baselines in the paper by searching over different hyperparameters for each baseline, including learning rate, actor update frequency, initial timesteps, and EE distance weight λ_{dist}. The best hyperparameters for each baseline that are different from HACMan are summarized in Table A.2; any hyperparameter not listed in Table A.2 is the same as our method (Table A.1).

Baselines	**Hyperparameters**	**Values**
Regress Contact Location (Point Cloud)	Actor update freq per env step	0.25
No Contact Location (Point Cloud)	Actor update freq per env step	0.25
	EE Distance Weight λ_{dist}	1
No Contact Location (State)	EE Distance Weight λ_{dist}	5

Table A.2: Baseline-specific Hyperparameters.

A.3 Supplementary Experiment Results

A.3.1 Additional ablations

We perform additional ablation studies to analyze each component of the proposed method with all the variants of the object pose alignment task. The results of the ablations are summarized in Fig. A.5.

Effect of Contact Location: To test the hypothesis that contact location matters for non-prehensile manipulation, we design a **"Random Location"** ablation: the policy randomly selects a contact location on the object instead of learning to predict a contact location. From Fig. A.5, we observe a performance drop for not predicting the contact location even for the simplest task variant.

Effect of Goal Representations: As described in Section 2.5.3 and Appendix A.1.3, in our method, we represent the goal by first computing the

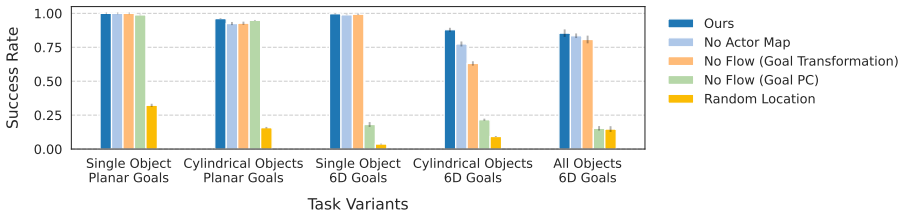


Figure A.5: Additional ablations. All of the components of our method are essential to achieve the best performance when the task becomes more difficult.

correspondence between the observation and goal point clouds and concatenating a per-point "goal flow" to the observation. We include two alternative goal representations to justify the use of goal flow in our pipeline: **"No Flow (Goal PC")** concatenates the goal point cloud with the observed point cloud [3, 4]. We use an additional segmentation label in the point features to distinguish the goal points from the observed points. From Fig. A.5, this ablation only works well on planar goals for this task. In **"No Flow (Goal Transformation)"**, we represent the goal as the transformation between the current observation pose and the goal pose. We represent this transformation as a 7D vector that includes a translation vector and a quaternion. We concatenate the 7D goal pose to the observation at all of the object points. Note that, similar to our method, this baseline also requires computing correspondences between the observation and the goal. This approach performs well but slightly worse than our method in the last two task variants.

Effect of Actor Map: Instead of using an Actor Map which has per-point outputs, this ablation uses an actor that outputs a single vector of motion parameters while keeping the Critic Map. This is different from the baselines in the previous section that remove both the Actor and Critic Maps. In the **"No Actor Map"** experiments, we observe a relatively minor performance drop compared to the full method. Nonetheless, using the per-point action output from an Actor Map instead of a single output may allow the agent to reason more effectively about different actions for different contact locations, such as the multimodal solution shown in Fig. 2.6 (middle).

A.3.2 Training curves and tables

In this section, we include the full training results for all the methods with additional task variants. Fig. A.6 and Fig. A.7 include the training curves for the baselines and the ablations. Table A.3 and Table A.4 are recorded at 200k environment interaction steps from the training curves for all the methods. The numbers in the tables are used to generate the bar plots in Fig. 2.4 and Fig. A.5.

Note we also interpolate between the tasks "Planar Goals" and "6D Goals" and include an additional task configuration with a fixed initial object pose and a randomized 6D goal, "6D Goals (Fixed Init)". This task configuration

is combined with the Single Object dataset and the Cylindrical Object dataset. Thus, we include 7 variants in total (5 variants in the main paper).

As discussed in Section 2.7, the baselines and ablations have poor performance when the task becomes more challenging. Our method achieves the best converged performance across all task variants while being more sample efficient.

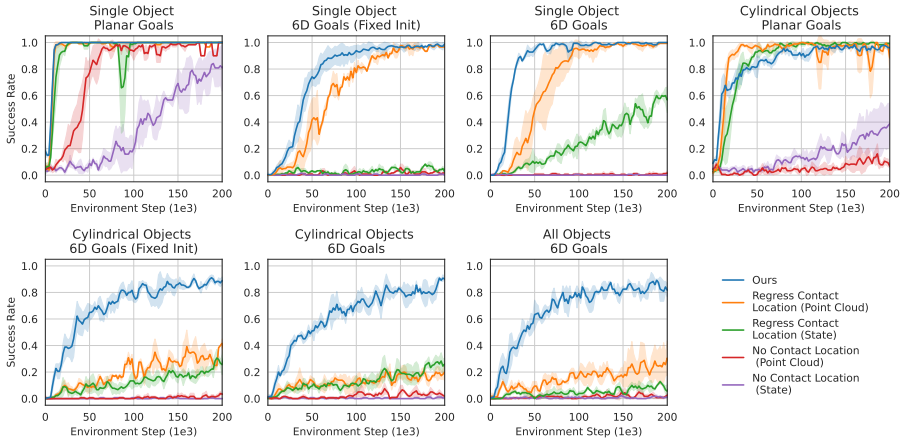


Figure A.6: Baselines. It shows success rates on the train dataset over environment steps. The shaded area represents the standard deviation across three training seeds.

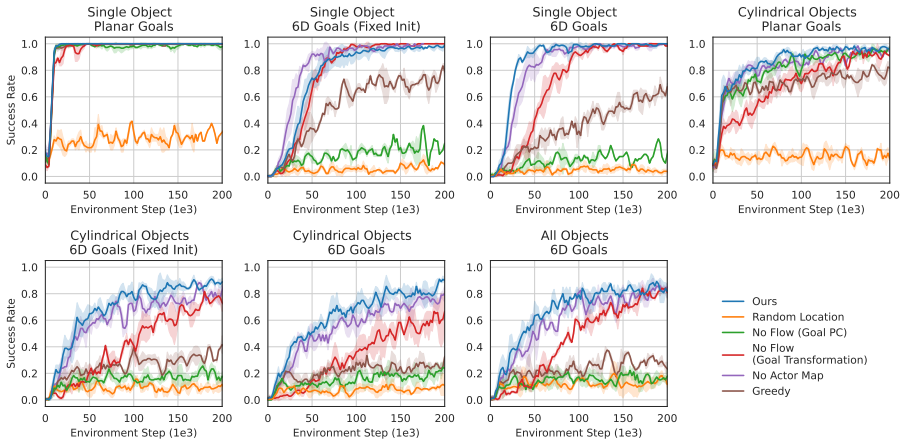


Figure A.7: Ablations. It shows success rates on the train dataset over environment steps. The shaded area represents the standard deviation across three training seeds.

Object Dataset	Task Configuration	Methods				
		No Contact Location		Regress Contact Location		Ours
		State	Point Cloud	State	Point Cloud	
Single Object	Planar Goal	**0.812 ± .012**	**0.973 ± .016**	**1.000 ± .000**	**0.996 ± .005**	**1.000 ± .000**
	6D Goal (Fixed Init)	0.003 ± .000	0.020 ± .002	0.060 ± .014	**0.971 ± .005**	**0.982 ± .004**
	6D Goal	0.000 ± .000	0.009 ± .001	0.573 ± .015	**0.991 ± .004**	**0.997 ± .003**
Cylindrical Objects	Planar Goal	0.361 ± .019	0.107 ± .007	**0.990 ± .002**	**0.924 ± .027**	**0.961 ± .003**
	6D Goal (Fixed Init)	0.001 ± .001	0.021 ± .002	0.264 ± .017	0.324 ± .014	**0.885 ± .004**
	6D Goal	0.006 ± .002	0.035 ± .002	0.258 ± .010	0.187 ± .012	**0.879 ± .014**
All Objects	6D Goal	0.012 ± .004	0.016 ± .009	0.094 ± .018	0.243 ± .028	**0.854 ± .028**

Table A.3: Baselines. We compare our method with baselines with different action representations and observations. Our approach outperforms the baselines, with a larger margin for more challenging tasks. The success rate is reported with the mean and standard deviation across three seeds.

Object Dataset	Task Configuration	Methods					
		Random Location	Greedy	No Flow (Goal PC)	No Flow (Goal Pose)	No Action Map	Ours
Single Object	Planar Goal	0.323 ± .011	**1.000 ± .000**	**0.989 ± .002**	**1.000 ± .000**	**1.000 ± .000**	**1.000 ± .000**
	6D Goal (Fixed Init)	0.075 ± .005	0.754 ± .023	0.198 ± .025	**1.000 ± .000**	**0.991 ± .002**	**0.982 ± .004**
	6D Goal	0.037 ± .003	0.633 ± .014	0.181 ± .018	**0.994 ± .004**	**0.989 ± .002**	**0.997 ± .003**
Cylindrical Objects	Planar Goal	0.158 ± .006	0.767 ± .017	**0.949 ± .003**	**0.927 ± .012**	**0.925 ± .012**	**0.961 ± .003**
	6D Goal (Fixed Init)	0.097 ± .006	0.346 ± .012	0.189 ± .009	0.746 ± .015	0.805 ± .016	**0.885 ± .004**
	6D Goal	0.093 ± .004	0.262 ± .011	0.216 ± .008	0.631 ± .016	0.775 ± .018	**0.879 ± .014**
All Objects	6D Goal	0.147 ± .021	0.293 ± .026	0.153 ± .017	0.808 ± .028	**0.835 ± .017**	**0.854 ± .028**

Table A.4: Ablations. We show that all of the components are essential to achieve the best performance when the task becomes more difficult. Each success rate is reported with the mean and standard deviation across three seeds.

A.3.3 Additional baseline: Global feature with query contact location

We consider an additional baseline in this section where both the actor and the critic use a global point cloud feature and a query contact location as input. The query contact location is represented as a 3D coordinate (x, y, z). More specifically, the actor takes as input a global feature and a contact location and outputs a continuous vector of motion parameters. The critic takes as input a global feature and a contact location and outputs a Q-value. Since both the actor and the critic require a contact location as input, we still need a way of selecting the query contact location. We follow a similar way as our method to use the observed points on the object as candidate queries and select the contact location based on the highest Q-value. In this way, the action space remains a discrete-continuous action space as our method, but it uses a global point cloud feature rather than a segmentation-style per-point feature.

As shown in Fig. A.8, this alternative baseline performs worse than our method. We hypothesize that a segmentation-style point cloud network can

reason about the local point features more effectively than a global feature extractor due to skip connections.

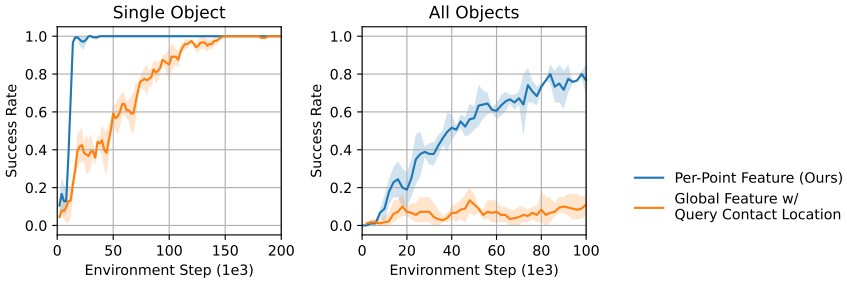


Figure A.8: Comparison between our method and the additional baseline with query contact locations. The left figure shows the success rate of the simplest task variant - a single object with planar goals. The right figure shows the most challenging task variant - all objects with 6D goals. The shaded area represents the standard deviation across three training seeds. Our method performs better than the baseline in both cases.

A.3.4 Extending Motion Parameters

The motion parameters in the main results are defined as a 3D vector that describes the translation motion of the gripper. In this section, we extend the motion parameters in different ways:

6D Contact. The motion parameters also predict the orientation of the gripper when the gripper approaches the contact location. The orientation is in the form of ZYX Euler angles. To account for the physical constraints of our task setup, we restrict the y and x angles to the range of $[-0.5\pi, 0.5\pi]$, and the z angle to the range of $[-\pi, \pi]$.

6D Motion. We introduce the ability for the gripper to change orientation while executing the motions after making contact. Similar to the translation motion parameters, the rotation motion parameters (ZYX Euler angles) represent the delta rotation at each action repeat step.

Per-point Contact Location Offset. We conduct an experiment where the agent learns a per-point contact location offset combined with 3D motion. The agent's continuous action space is defined as (contact_offset, 3D motion parameters). For each action on a given point at location x, the agent uses $(x + x_{\text{offset}})$ as the contact location. Notably, the x_{offset} value is mapped to scale with the bounding box since it ranges between $[-1, 1]$.

Table A.5 compares the performance of HACMan with the modified action spaces. The success rates are reported along with their corresponding standard deviations. We find that including 6D motion in the motion parameters results in a slightly higher success rate. However, 6D contact or contact offset does not seem to provide significant benefits. We also try to include 6D motion in the **Regress Contact Location** baseline. As shown in Table A.5, 6D motion

Method	Success Rate
HACMan Default	0.833 ± .018
+ with 6D Motion	0.866 ± .090
+ with 6D Contact	0.819 ± .077
+ with Contact Offset	0.800 ± .011
Regress Contact Location Default	0.243 ± .028
+ with 6D Motion	0.356 ± .133

Table A.5: Success rates with different motion parameters. All methods are evaluated on all train objects with 6D goals.

# of Scene Objects	Success Rate
0 (Default)	0.833
1	0.773
5	0.580

Table A.6: Success rates under different cluttered scenes. All methods are evaluated with 6D goals.

improves the performance of the baseline, but the success rate is still much worse than our method.

A.3.5 Experiments in cluttered environments

We can directly apply HACMan to a setting of manipulating objects in cluttered scenes. We conduct preliminary experiments in which we introduce varying numbers of scene objects into the bin. The scene objects serve as obstacles that add challenges to the task. We train HACMan under two conditions: **with one scene object** and **with five scene objects**, and we compare the results with the performance achieved in the absence of any scene objects (default setting). From Table A.6, as expected, the task becomes more challenging when there are more obstacles in the bin. As illustrated in Fig A.9, the policy tends to push the object directly toward the goal by pushing the scene object aside.

A.3.6 Effect of longer training time

Although we report the success rate at 200k training steps for all the results due to computational limitations, our method continues to improve performance with longer training (Figure A.10). The graph illustrates the success rate achieved by our method as the number of training steps increases. Notably, after 500k training steps, our method achieves a success rate of $91.1 \pm 7.3\%$, significantly improving from the 83.3% success rate reported in the main text (at 200k training steps).

Figure A.9: Qualitative results for object pose alignment tasks in cluttered environments. HACMan shows complex non-prehensile behaviors that move objects to goal poses (shown as the transparent objects). The scene objects are colored in brown to distinguish from the target object to be manipulated to the goal pose.

Figure A.10: Success rate with extended training. The success rate of our method reaches $91.1 \pm 7.3\%$ after 500k training steps, compared to 83.3% after 200k training steps.

A.3.7 Effect of longer episode lengths

We conducted an additional experiment to explore the relationship between success rates and maximum episode length. In the main paper, our episodes were limited to a maximum of 10 steps, and any episode exceeding this limit was deemed a failure. During this additional evaluation, we relaxed the episode length restrictions and allowed the agent to operate with a maximum episode length of 30. As shown in Fig A.11, HACMan achieves more than 95% success rates across all datasets (Train 96.6%, Train (Common) 99.4%, Unseen Instance (Common) 99.7%, Unseen Category 95.1%) when the maximum episode length is extended to 30. This suggests that providing the agent with a longer time horizon enables it to achieve higher success rates without the need for retraining.

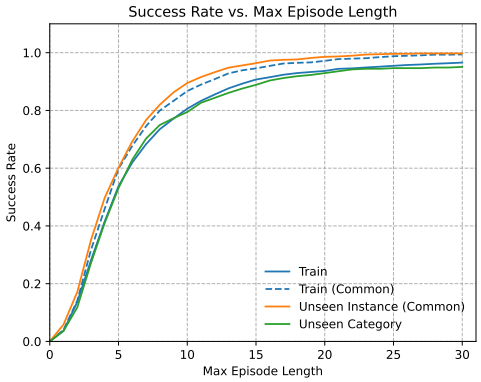


Figure A.11: Success rates at various maximum episode lengths. This line plot shows the success rates of HACMan evaluated on the four datasets. It is worth noting that the success rates for Unseen Instance (Common) and Train (Common) are marginally higher compared to Train and Unseen Category, similar to the pattern in Table 2.2.

A.3.8 Per-category result breakdown

Fig. A.12 shows the breakdown of the results for each object category. Although our method demonstrates consistent performance across the majority of objects, there are certain objects with geometries that pose intrinsic challenges for our approach. For example, our method is limited to poking a bowl from the top due to occlusions, making it difficult to flip an upward-facing bowl downwards.

A.3.9 Final Distance to Goal

To further analyze the performance of our method, Fig. A.13 visualizes the distribution of distances to goal of our method at the end of the episodes for the "All Objects 6D Goals" task variant. These distances are computed as the norms of the flow distances between the objects and their respective goals.

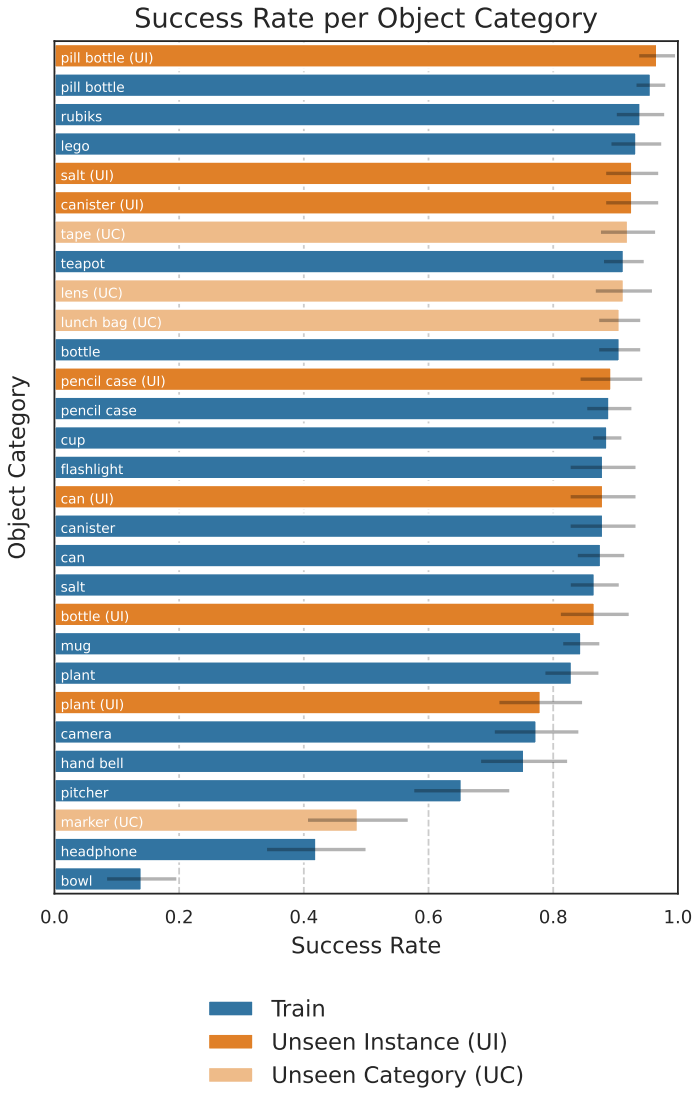


Figure A.12: Results breakdown. Object categories in the unseen instance set (orange) can be compared to the same object categories in the train set (blue) to see the level of instance generalization.

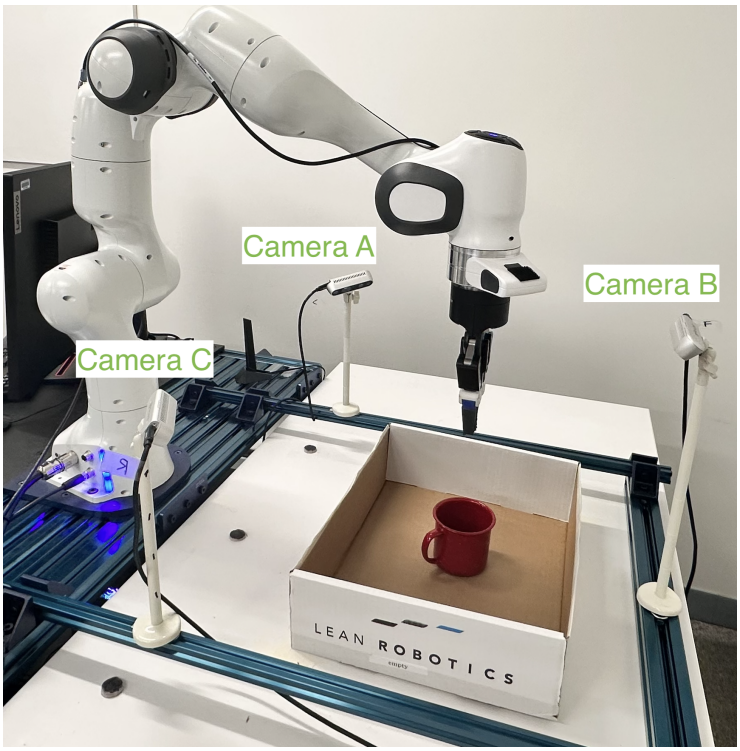


Figure A.14: Real robot setup.

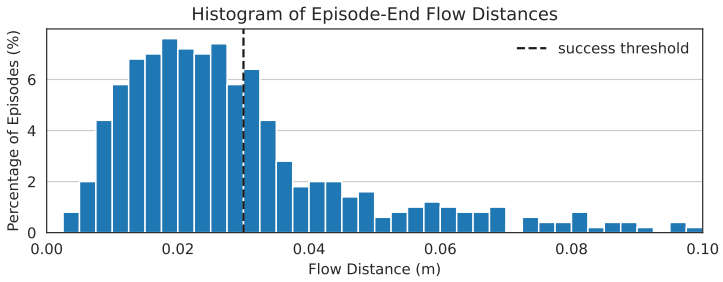


Figure A.13: Distribution of distances to the goal at the end of the episode for our method in the "All Objects 6D Goals" experiment. The vertical dashed line represents the success threshold at a distance of 0.03m. The distribution has a median of 2.57cm, a mean of 3.66cm, and a standard deviation of 4.27cm.

A.4 Real Robot Experiments

A.4.1 Real robot setup

The robot setup is shown in Fig. A.14. We use three cameras on the real robot to get a combined point cloud. We follow a similar procedure as Appendix A.1 to process the point cloud and to execute the action except the following details: We segment the object points from the full point cloud based on the location and the dimension of the bin instead of using the ground truth segmentation labels from Robosuite. To move the gripper to the pre-contact location, we first move the robot to a location above the pre-contact location and then move down to the pre-contact location, instead of "teleporting" the gripper in simulation.

To obtain goals for the real world evaluation, we record 10 goal point clouds for each object by manually setting the objects into different stable poses. During each timestep, we use point cloud registration algorithm to estimate the goal transformation to calculate the goal flow. Specifically, we use the global registration implementation from Open3D (`http://www.open3d.org/docs/release/tutorial/pipelines/global_registration.html`) and then use Iterative Closest Point (ICP) for local refinement. Note that we only match the shapes of the object instead of matching both the colors and the shapes due to the limitation of the registration algorithms.

Note that the evaluation process can be done automatically without any manual resets. The reward and the episode termination condition (Appendix A.1.4) are both calculated automatically.

For the real robot experiments, we use the policy trained with "6D goals" and the "All Objects" dataset. We perform zero-shot sim2real transfer without finetuning or additional domain randomization. We have tried to add noise to the contact location execution and add noise to the point cloud observation. However, these modifications did not result in better real robot performance.

A.4.2 Analysis

We include additional analysis on the real robot results in this section. The proposed method assumes an estimated goal transformation as input. To estimate the transformation from the object to the goal, we use point cloud registration, as described above. However, the estimation of the transformation might not be perfect in the real world. To better understand the performance of our system, we define two types of evaluation criteria: The "flow success" is automatically calculated based on the *estimated* point cloud registration according to the evaluation metric in Appendix A.1.4. Hence, "flow success" will sometimes mark an episode as a success or failure incorrectly due to errors in the point cloud registration. For the "actual success" evaluation metric, we manually mark as failures the cases among the flow success episodes where the goal estimation is significantly wrong. Thus, "flow success" indicates the performance of the trained policy (assuming perfect point cloud registration at termination) while the "actual success" indicates the performance of the full system (accounting

for errors in the point cloud registration). Fig. 2.7 in the main text reports the actual success. We include both success metrics in Table A.7 below. The policy achieves a 61% success rate based on the flow success, indicating that some of our errors are due to failures in point cloud registration.

Object Name	Planar Goals		Non-planar Goals		Total	
	Flow	**Actual**	**Flow**	**Actual**	**Flow**	**Actual**
(a) Blue cup	4/7	4/7	7/13	4/13	5/20	4/20
(b) Milk carton	6/7	6/7	10/13	10/13	16/20	16/20
(c) Box	2/5	2/5	10/15	10/15	12/20	12/20
(d) Red bottle	7/7	4/7	6/13	0/13	13/20	4/20
(e) Hook	5/8	5/8	5/12	5/12	10/20	10/20
(f) Black mug	4/7	4/7	2/13	0/13	6/20	4/10
(g) Red mug	5/7	5/7	7/13	3/13	12/20	8/20
(h) Wood block	6/7	6/7	8/13	6/13	14/20	12/20
(i) Toy bridge	9/10	9/10	7/10	5/10	16/20	14/20
(j) Toy block	2/2	2/2	10/18	10/18	12/20	12/20
Total	50/67	47/67	72/133	53/133	122/200	100/200
Percentage	75%	70%	54%	40%	61%	**50%**

Table A.7: Additional analysis on the real robot experiments. An episode is considered a "flow success" if the average norm of the estimated flow is less than 3 cm. An episode is considered as an "actual success" if the object is aligned with the goal pose without point cloud registration failure.

A.4.3 Failure cases

We discuss the failure cases of the real robot experiments in this section and include the videos on our website: https://hacman-2023.github.io/. The most noticeable failure cases are due to the errors of point cloud registration. The challenges of the registration methods come from noisy depth readings and partial point clouds. The error of the point cloud registration methods will lead to unexpected actions during the episode. In addition, it may end the episode early because the episode termination depends on the goal estimation. This motivates us to separate out the success criteria in Table A.7 based on the failures of the registration method.

On the action side, both the contact location and the motion parameters might have execution errors. Since the contact location is selected from the observed point cloud, when the camera calibration is not accurate enough, the robot might not be able to reach the desired contact location of the object. In addition, since we use a compliant low-level controller to execute the motion parameters, the robot might not be able to execute the desired motion the same way as in the simulation.

In addition, the object dynamics might be different from the simulation due to the surface friction and the density of the object. The performance of our method could be further improved with domain randomization over the physical parameters.

A.5 More discussion on non-prehensile manipulation

Non-prehensile manipulation is an important aspect of the robot's capabilities, particularly in scenarios where grasping encounters limitations, as demonstrated in Fig. A.15. This section discusses the importance of non-prehensile actions in two key contexts:

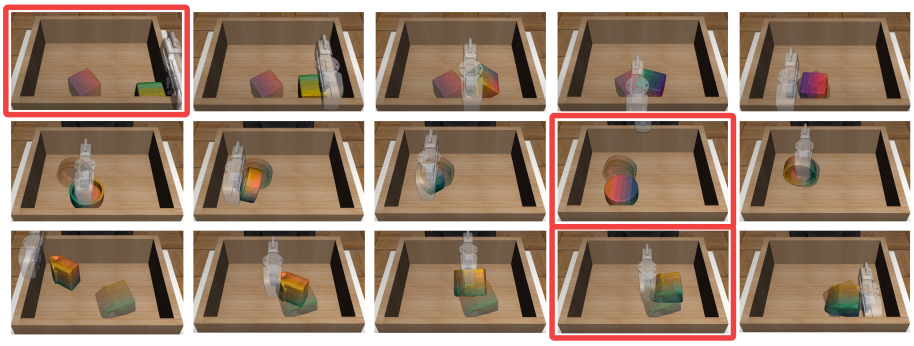


Figure A.15: Examples showcasing limitations of prehensile manipulation. The frames where prehensile manipulation is challenging are highlighted. The first row shows a cube placed at the corner of the bin, where any grasp is obstructed by the bin wall. Both the second and third rows depict instances where objects are too large to be grasped at specific poses.

Environment Occlusion. The first row of Figure A.15 shows an example where the potential grasp poses of the object are occluded by the wall. Non-prehensile moves, like nudging objects to a better position, offer a practical solution in such a scenario.

Oversized Objects. The last two rows of Figure A.15 include scenarios where certain dimensions of the object are larger than the width of the gripper.

A.6 More discussion on the related work

A.6.1 Compared to Chen et al. [3, 4]

Our work is substantially different from Chen et al. [3, 4] from the follow aspects: **Approach:** The approach in Chen et al. [3, 4] follows a student-teacher training pipeline. The teacher training is equivalent to the "No Contact Location" baseline with "states" observations in our paper. The policy takes all the relevant robot state and object state information and outputs delta robot actions. Note that they train a single teacher policy across all shapes without using the point cloud which results in a state-observation policy that is "robust" to shapes instead of "adaptive" to shapes (see Discussion section in Chen et al. [4]). As shown in Table II, this baseline performs significantly worse than our method

in our task because it lacks shape information from the point cloud and the robot-centric action space is not as efficient as our object-centric action space. On the other hand, although the student policy in Chen et al. [3, 4] takes point cloud observation, it is trained using imitation learning from the teacher, so its performance is upper bounded by the teacher policy which has been shown to be worse than our proposed method.
Task: We investigate a completely different task and thus the numbers are not really comparable with the numbers from previous work [3, 4]. First, we use a simple gripper instead of a dexterous hand. Second, we consider matching the orientation and position of the goal pose while Chen et al. [3, 4] only considers orientation.

A.6.2 Compared to Cheng et al. [5], Hou and Mason [15]

Unlike Cheng et al. [5], Hou and Mason [15], our method is not limited to a simplified gripper model and does not require the knowledge of object environment contact modes which are challenging to estimate during real robot execution. In addition, Cheng et al. [5], Hou and Mason [15] are restricted by quasi-static assumptions. Although our method requires static point cloud observations in between robot actions, the robot interaction with the object is not restricted to quasi-static motion. As shown in Figure A.16, the flipping motion could be non-quasi-static. However, we also want to point out that static point cloud observations may limit the method from more complex dynamic motions.

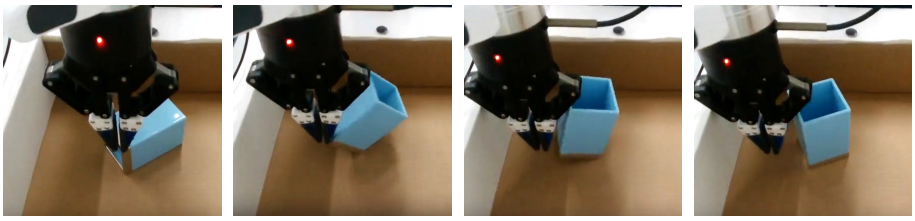


Figure A.16: An example of non-quasi-static motion. The figure shows an example of executing the motion parameters to flip a mug upright. After the gripper pushes against the edge of the mug (first two images), it relies on the inertia of the mug to finish the motion which is not quasi-static (last two images).

Appendix B

Experiment Details for HACMan++

B.1 Algorithm and Implementation Details

B.1.1 Observations

The robot agent first perceives a point cloud $\mathcal{X} \in \mathcal{R}^{M \times 3}$ for the entire scene by stacking multiple cameras' views together, where M is the number of points in the raw point cloud observations. Our method assumes that we pre-process the scene by segmenting the object point cloud $\mathcal{X}^{obj}$ from the background point cloud $\mathcal{X}^{b}$; details of the segmentation process are listed in Appendix B.2.1 and Appendix B.4.1. After segmentation, we downsample $\mathcal{X}^{obj}$ and $\mathcal{X}^{b}$ with voxel sizes of 1cm and 2cm respectively. We then randomly sample 400 and 1000 points from $\mathcal{X}^{obj}$ and $\mathcal{X}^{b}$ respectively.

To make our policy also goal-conditioned, we append the goal information into the observation as "goal flow" in which we compute the per-point correspondence from the current object point cloud to the goal object point cloud. Specifically, for each point x_i in the object point cloud, the goal flow is $\Delta x_i = x_i^g - x_i$, where x_i^g is a corresponding point of x_i in the goal point cloud. In the simulation, we use the ground-truth point correspondence given the object pose and the goal pose. In the real world, we use point cloud registration to align the observation to the goal (see Appendix B.4.2).

Therefore, the entire observation space (o_p, o_g, o_m) of our robot agent includes three parts: the point cloud o_p representing the 3D position (x, y, z) of the points (3-dimensions per point), the goal flow o_g indicating the flow from the current object point cloud to the point cloud of the object in the goal pose (3-dimensions per point), and the segmentation mask o_m (1-dimension per point).

B.1.2 Primitive Implementation Details

We have five generic motion primitives that can be used strategically and collectively to solve long-horizon manipulation tasks. The details of the actual execution of those primitives are listed below.

Poke: The Poke primitive is parameterized by a location parameter $a^{loc} \in \mathcal{X}^{obj}$ and a motion parameter $a^m = (x^m, y^m, z^m, \theta_x, \theta_y) \in (-1, 1)^5$, described below. The gripper first estimates the surface normal a^{norm} of the object at a^{loc} and then goes to the pre-contact location $a^{loc}_{pre} = a^{loc} + a^{norm} \times d_1$, for a hyperparameter $d_1 = 0.04$. After reaching the pre-contact location, the gripper moves to the actual contact location a^{loc} with a rotation $\theta = arctan(\frac{\theta_x}{\theta_y})$ along the z axis. In the last step, the robot moves a delta position (x^m, y^m, z^m) from the contact location. After the poking motion, the gripper returns to the reset pose before it captures the next step observation.

Grasp: The Grasp primitive is parameterized by a location parameter $a^{loc} \in \mathcal{X}^{obj}$ and a motion parameter $a^m = (\theta_x, \theta_y) \in (-1, 1)^2$. The robot opens the gripper and goes to a pre-contact location $a^{loc}_{pre} = x^{loc} + (0, 0, d_2)$ above the actual location ($d_2 = 0.1m$) with a gripper orientation $\theta = arctan(\frac{\theta_x}{\theta_y})$ around the z axis. Then the gripper goes down to the actual contact location and closes the gripper. After grasping, the gripper moves upward in the z axis by $d_3 = 0.15m$.

Move to: The Move to primitive is parameterized by a location parameter $a^{loc} \in \mathcal{X}^b$ and a motion parameter $a^m = (x^m, y^m, z^m, \theta_x, \theta_y) \in (-1, 1)^5$. The policy chooses a background point a^{loc} at which to place the object. Since the gripper is grasping the object, there is an offset between the gripper position and the placed point. Therefore, the actual location the gripper moves to is $a^{loc} + d_4 \cdot (x^m, y^m, z^m)$, where (x^m, y^m, z^m) is a learned offset and d_4 is the maximum dimension of the object. During the movement, the gripper also rotates to the orientation $\theta = arctan(\frac{\theta_x}{\theta_y})$ in the z axis.

Move delta: The Move delta primitive is parameterized by a location parameter $a^{loc} \in \mathcal{X}^b$ and a motion parameter $a^m = (x^m, y^m, z^m, \theta_x, \theta_y) \in (-1, 1)^5$. The gripper moves a delta position (x^m, y^m, z^m) with a gripper rotation $\theta = arctan(\frac{\theta_x}{\theta_y})$ about the z axis.

Open gripper: The Open gripper primitive has no parameters and the selected location a^{loc} also doesn't influence the action. It is an atomic robot action which opens the gripper to the full extent.

B.1.3 Baseline Implementation

This section provides details of the baselines' key implementation features. Table 3.1 summarizes the main differences between our method and the baseline approaches.

P-DQN [50]. P-DQN uses parameterized primitives, similar to our method, but it lacks spatial grounding in its primitive location selection. In our implementation of this baseline, we processes the point cloud input to derive a global

critic feature f_k and a global actor feature f_k^a for each primitive k using a classification-style network. P-DQN predicts K vectors of primitive parameters and K scores, corresponding to the K primitives. In contrast to our method, for each vector of primitive parameters, P-DQN predicts additionally three dimensions as the regressed location, which are mapped to the predicted Area of Interests (as explained in Section B.1.5). P-DQN then selects the primitive with the highest score during inference or samples from the softmaxed scores during exploration.

RAPS [6]. Instead of handling different primitives with separate networks, RAPS extracts a single global actor feature f^a and a single global critic feature f from the input point cloud using a classification-style network. It predicts an action which includes the primitive parameters for all K primitives as well as the log-probabilities of executing each primitive.Similar to P-DQN [50], RAPS regresses to three dimensions for the primitive location for each primitive, which are mapped to the Area of Interest (as explained in Sec B.1.5). RAPS selects the primitive with the highest log-probability in execution and samples from the predicted log-probability during exploration. Note that while our comparison uses TD3 [8] for the baseline to maintain similarity with our method, the original RAPS study experimented with various RL algorithms [6]

B.1.4 Hyper-parameters

Table B.1 lists the training hyper-parameters used in training.

Hyperparameter	**Ours**	**P-DQN**	**RAPS**
Target Update Interval	4	1	1
Actor Update Interval	4	1	1
Learning Rate	1e-4	1e-4	1e-4
Batch Size	64	64	64
Epsilon Greedy	0.1	-	-
Action Noise	0	0.1	0.1
Exploration Temperature	0.1	0.1	-
Tau	0.005	0.005	0.005

Table B.1: Training Hyper-parameters.

B.1.5 Regressed Location Mapping

By spatially-grounding the primitive locations on the input point cloud, our method naturally has an object-centric action space. This allows the more frequent interactions with the objects during exploration. To make it a fair comparison between our method and the baselines, we map the regressed primitive locations predicted by the baseline methods to the Area of Interest (AoI)

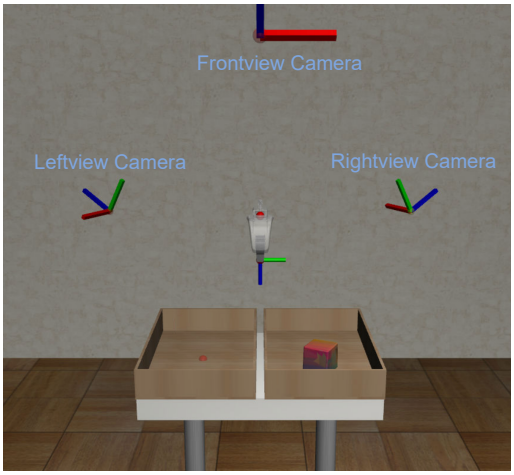


Figure B.1: Simulation DoubleBin Setup. The visualization of the simulation environment and the positions of the cameras.

of each primitive. Specifically, for a raw primitive location prediction $a^loc = (x^m, y^m, z^m) \in (-1, 1)^3$, we scale and translate it to the

1. the bounding box of the target object for **object-centric** primitives such as *Poke*, *Grasp*.
2. the entire workspace for **background-centric** primitives such as *Move to*.

This change significantly improves the baseline performance, although our experiments show that these baselines still perform significantly worse than our method.

B.2 Simulation Experiment Details

Below we describe additional details of our simulation experiments.

B.2.1 Simulation Tasks

1. ***Lift Cube (ManiSkill)*:** We use the Lift Cube task from ManiSkil2 [28]. The goal of this task is to lift the cube to a goal height of $0.2m$. The initial cube position is uniformly sampled from $[-1, 1]^2$ with a rotation uniformly sampled from $[0, 2\pi]$. The reward function is composed of a reaching reward, grasping reward and lifting reward (see the Maniskill2 documentation for details). We substract ManiSkill2's original reward function by its max value such that the returned reward is always negative,

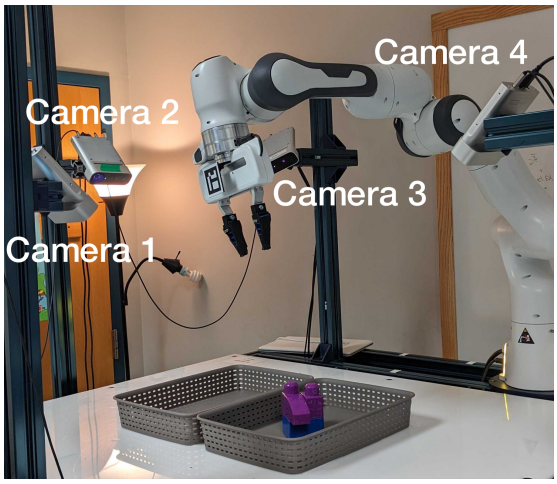


Figure B.2: Real-world DoubleBin Setup. The visualization of the real-world DoubleBin environment and the positions of the cameras.

thus encouraging the agent to achieve the success condition as soon as possible thereby ending the episode.

2. ***Stack Cube (ManiSkill)*:** We use the Stack Cube task from ManiSkil2 [28]. The goal of the task is to pick up a red cube and place it onto a green one. When the red cube is placed on the green cube and not grasped by the robot, the episode is a success (see the ManiSkill2 documentation for details). Similarly, we substract ManiSkill2's original reward function by its max value such that the returned reward is always negative.

3. ***Peg Insertion (ManiSkill)*:** This task is a modified version of the Peg Insertion Task in ManiSkill2. The goal is to insert a peg into the horizontal hole in a box. We slightly simplify the original task by removing any randomization of the hole's location and peg's initial pose. We also marginally enlarge the hole by adding $1cm$ clearance. Similarly, we substract ManiSkill2's original reward function by its max value such that the returned reward is always negative.

4. ***DoubleBin Task*:** We build the customized DoubleBin task environment in Robosuite [58]. The environment has two bins on a table and three cameras, looking over the bins from the left, the right and the front, as shown in the Figure B.1. The size of the bin is $40cm \times 24cm \times 6cm$ and the distance between the centers of the bins is $13.5cm$. For each camera, we record the depth image and project all the points back to the tabletop frame, which has an origin at the middle between the two bins. In Robosuite, we have access to ground-truth segmentation labels of the object and we use these labels to compute a segmentation mask for all the points.

We then combine the points from all of the cameras to obtain the point cloud observation. Each episode, an object is randomly chosen from our dataset and loaded into the environment. The object is dropped from the air above one of the bins, after which it reaches a stable initial pose. The reward function for training our RL policy is $r_t = -\frac{1}{N}\sum_{i=1}^{N} ||x_i^g - x_i||$ where $||\cdot||$ is the L2 norm, N is the total number of points on the object point cloud, x_i is a point in the point cloud of the current object pose, and x_i^g is a corresponding point in the point cloud of the goal pose.

The goal poses in the *DoubleBin Task* are sampled by dropping the objects from a certain height and waiting until they stabilize.

The task is considered a success only when $r_t > -0.03$, which is equivalent to $\frac{1}{N}\sum_{i=1}^{N} ||x_i^g - x_i|| < 0.03$m.

B.2.2 Training and Evaluation Details

In this section, we include the training curves of our methods and baselines over four different task variants. Figure B.3 shows the success rate of the task with regard to the environment interaction steps. Here every environment interaction step refers to one primitive step in the environment which may involve several low-level atomic steps to complete. For every method, we run the experiment across three training seeds. The variance of the seeds is plotted in the figure with the shading area.

We first fill the replay buffer with trajectories by performing 1e4 random actions. Then we start training our policy. During training, we evaluate the current policy every 5e3 environment interaction steps over 20 episodes and report the average success rate across those episodes.

B.2.3 Object Dataset Processing and Visualization

The object dataset we are using is from Liu et al. [22]. We use convex decomposition for the objects and generate watertight mesh following Zhou et al. [57]. We also scale the objects so that the maximum object dimension is 10cm. During training, we randomly sample an object and apply additional proportional scaling to all the dimensions within a range of $[0.8, 1.2]$ to simulate objects of different sizes. We also follow the procedure from Zhou et al. [57] to filter out the objects that have some simulation artifacts or flat and thin objects that are too difficult to poke. After the filtering, we have 44 objects remaining, including cube, bottle, cup, mug, etc. Those objects are divided into three subsets including 32 objects in **Train**, 7 objects in **Unseen Instance (Common Categories)** and 5 objects in **Unseen Category**. Figure B.4 shows the **Train** objects and Figure B.5 shows the **unseen** objects we test during evaluation.

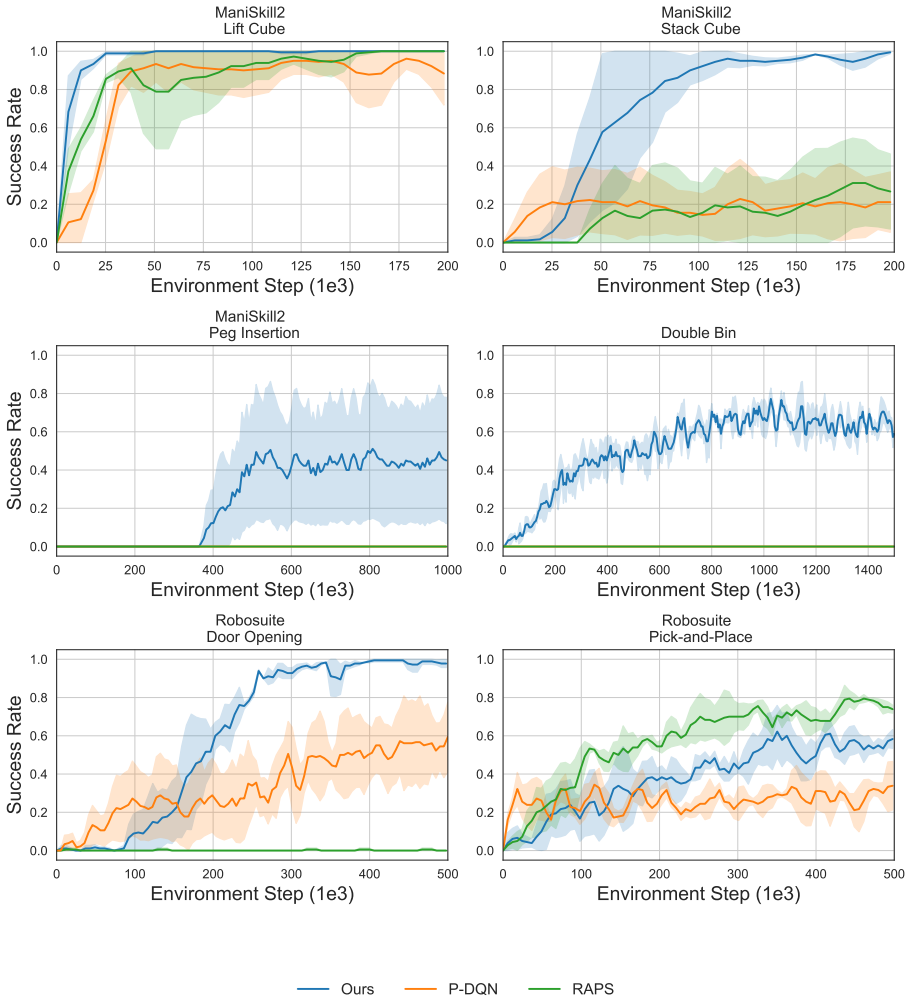


Figure B.3: The success rate of six different tasks over environment steps. Each method is averaged over three different seeds and the standard deviation is represented in the shaded area. Our method (blue) consistently outperforms both of the baselines in almost all tasks.

B.2.4 Primitive State Estimation

In simulation, we need to determine whether an object is grasped because certain primitives can only be used when an object is grasped (Move to and Move delta). In order to accurately estimate the *grasped* state, we have two conditions. First, we use the Mujoco contact detection to detect if any inner side of the fingertip is in contact with the object. If both inner sides of the fingertip are in contact, we return *true* for the *grasped* state. However, only checking the contact brings

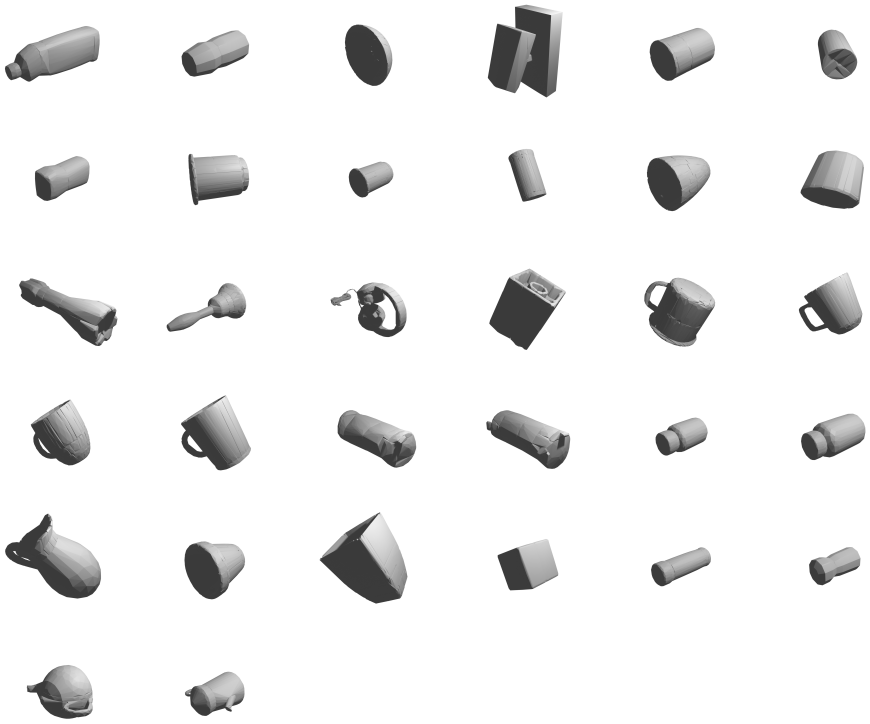
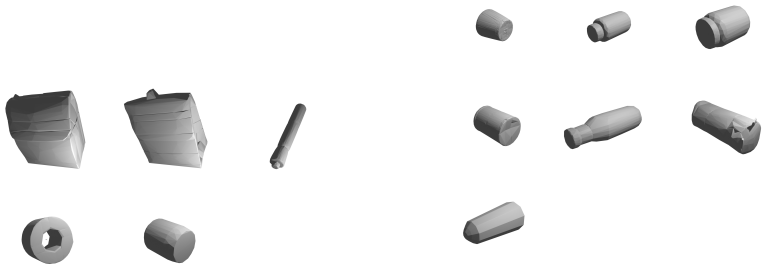


Figure B.4: 32 **Train** objects used in the training



(a) 5 **Unseen Category** objects

(b) 7 **Unseen Instances (Common Category)** objects

Figure B.5: 12 Unseen objects used in the evaluation

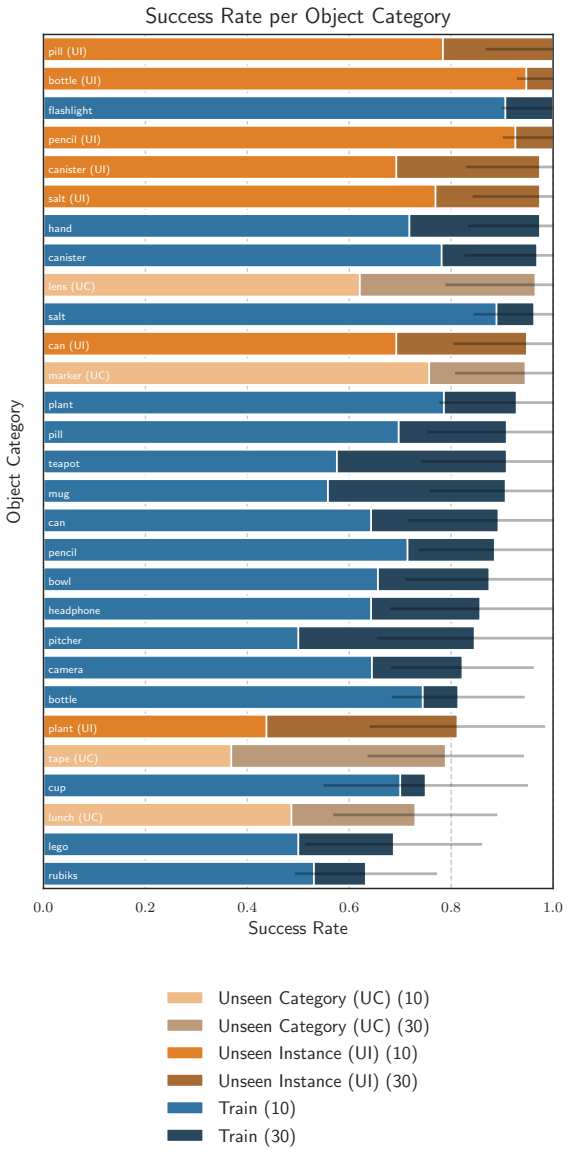


Figure B.6: Results breakdown for different object categories with the different maximum episode lengths annotated in the legend as (10) or (30). The Unseen Object Instance (Common Category) (orange) has comparable performance with the same object category in the Train (blue). The overall success rate of 30 maximum episode steps is higher than the one with 10 maximum episode steps.

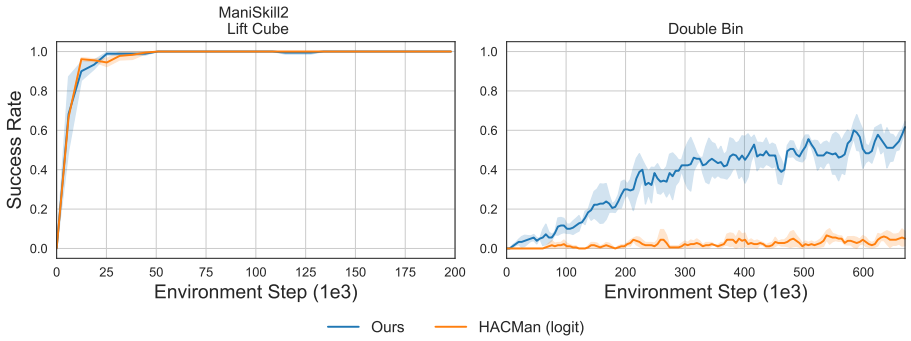


Figure B.7: The success rate of two different tasks over environment steps. Each method is averaged over three different random seeds and the standard deviation is represented in the shaded area. The baseline HacMan(logit) (orange) achieves comparable performance on easy task *ManiSkill Lift Cube* but fails to match the performance of our method (blue) in more challenging *Double Bin* task.

some false negative cases because the simulation sometimes cannot detect the contact for some grasping poses due to simulation artifacts. For example, when the gripper grasps the object in corner and it is able to lift the object, the simulation only detects one inner side of the fingertip is in contact of the object. Therefore, to reduce the false negatives, we add the second condition, that is to check if the object is above the table. To be more specific, we check if the object z position is above the table by at least two times of the object's maximum dimension. The final *grasped* state is evaluated to be *true* if either of these two conditions is satisfied. Otherwise, the *grasped* state is *false*.

B.2.5 Simulation: Additional Evaluation

In order to have a comprehensive analysis of our method's performance across different geometries and shapes. In this section, we report the breakdown results, i.e., average success rate for each object category. Figure B.6 shows that our method performs consistently well across a large category of objects. However, there are some categories with which our method struggles because of the irregular geometries.

B.2.6 Simulation: Dexterous Hand Task

To demonstrate that our set of primitives applies to different morphologies of end-effectors, we demonstrate the experimental results on tasks with dexterous hands in simulation apart from the main results we have for grippers.

In the dexterous hand task, we use the *Relocate* task from Adroit simulation benchmark [**?**]. The goal is to grasp the red sphere with the ShadowHand and move it to the goal position. The accepted range of the goal is denoted as a

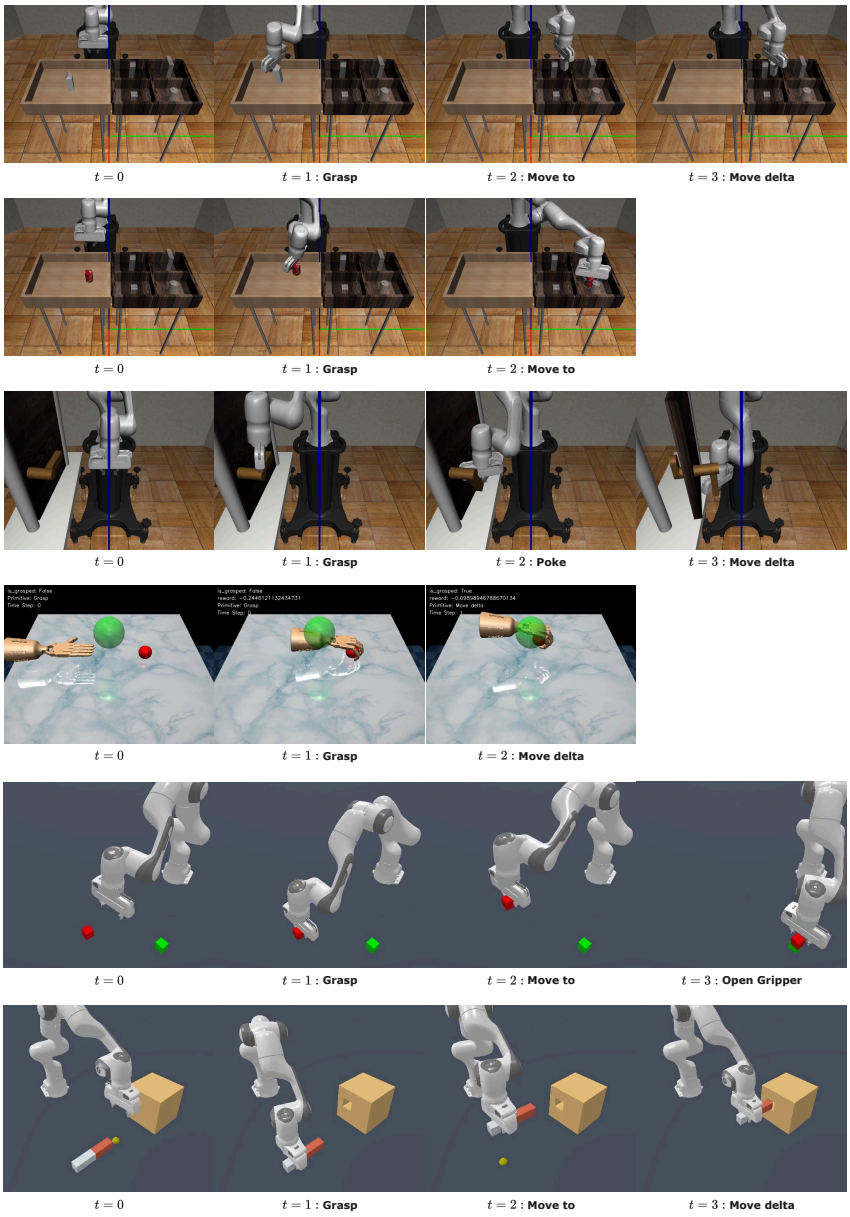


Figure B.8: Demonstration of our method's rollouts on additional tasks. The first two rows demonstrate our method performing the Pick-and-Place (RoboSuite) task with different objects. The third row shows our method performing the Door Opening (RoboSuite) task. The last two rows show our method performing the Relocate (Adroit) task with different initializations.

large green sphere. We adapt the implementations of the five primitives to be compatible with the robot morphology. When controlling the position of the hand as in *Poke*, *Grasp*, and *Move to*, we align the palm center of the hand to the target position. *Grasp* and *Open Gripper* crunches (all finger joints set to 1) and stretches (all finger joints set to 0) all the fingers respectively. For simplicity, we also remove all the parameters controlling the z-axis rotation. In other words, the hand does not change its orientation during any primitive execution.

Figure B.8 includes an example rollout of our agent. Our method achieves 100% success rate on this dexterous hand task within 20k training steps, proving our method's generality.

B.2.7 Simulation: Additional Baseline

Prior work [57] proposes a hybrid discrete-continuous action space for using a single spatially-grounded poking primitive to align the object 6d pose in a single bin. There are multiple ways to extend this single primitive framework to incorporate more primitives to solve a diverse range of general manipulation tasks. In order to demonstrate that designing a framework for multi-primitive setting is non-trivial, we add an additional baseline in this section to show that a naive extension of [57] can work on easier tasks like *ManiSkill Lift Cube* task but fail to match our method's performance in more challenging tasks like *Double Bin* task.

We introduce a new baseline, named as HACMan (logit), which extends the continuous action of [57] to also include logits. Specifically, the discrete action a^{loc} is to choose a point out of N points in the point cloud and the agent uses that as a location to apply the primitive motion. The continuous action (a^m_{all}, a^{logit}) includes the continuous motion parameters $a^m_{all} = (a^m_{grasp}, a^m_{poke}, a^m_{move\_to}, a^m_{move\_delta}, a^m_{open\_gripper})$ of all the five primitives and additional 5d logits $a^{logit} \in \mathbb{R}^5$. In execution, the location a^{loc} is chosen based on the Q value of the network. It first chooses the point index $\arg\max_i Q_i(s, a^m_i), 1 \leq i \leq N$ and maps the index to a 3d point location. Then we get the corresponding logits a^{logit}_i. The primitive a^{prim} is selected by sampling through the softmax over these logits. The final primitive motion parameter is selected by indexing the motion parameters $a^m_{all}[a^{prim}]$. We conduct the experiments in two different tasks and the results are shown in Figure B.7. Compared to our method, which predicts a separate Q value for point and primitive, HACMan (logit) predicts only separate Q value for the point. This structure makes it difficult to learn spatial reasoning with different primitives, leading to its failure on challenging *Double Bin* tasks.

B.3 Primitive Heatmap Visualization

In Figure B.9, we visualize the spatial Critic maps for different primitives, which vary based on the primitive type and in different regions of an object, based on

the geometry of the object. This visualization showcases the agent's capacity for multi-modal reasoning and geometric adaptability in task execution.

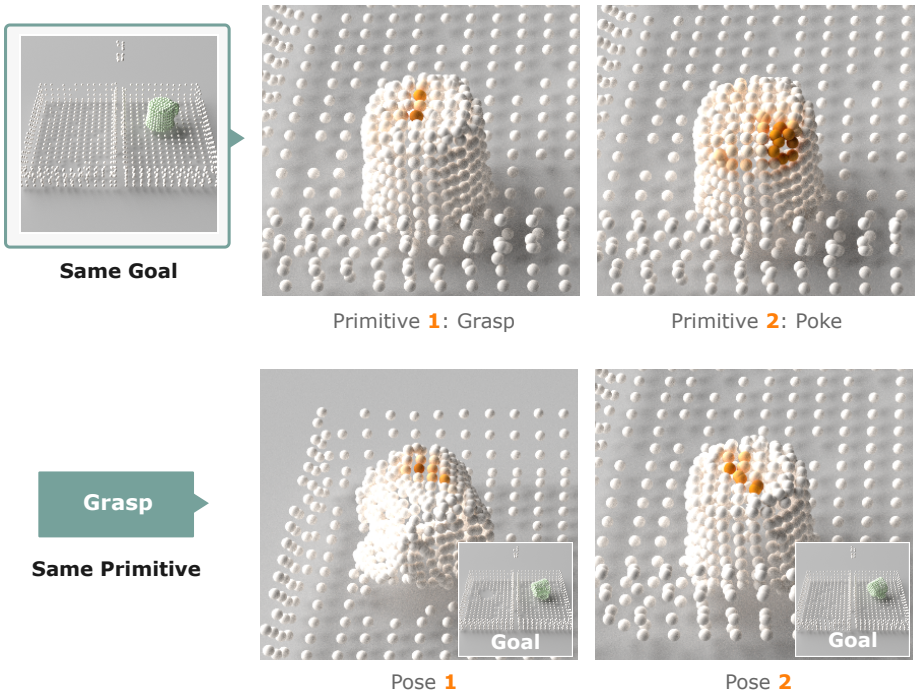


Figure B.9: Primitive Heatmap. **The first row** shows two critic heatmap for two different primitives at the same time step in an rollout. The agent learns to apply different primitives at different regions of the mug, based on its geometric features: *Grasp* needs to be applied to the center of the mug; *Poke* needs to be applied to the side of the mug to flip it into a more easily graspable pose. **The second row** shows two critic heatmaps for the same object and the same primitive at two different poses. The agent learns to adapt its grasp location when there is a pose difference of the mug.

B.4 Real-world Experiments

B.4.1 Real World Setup

Figure B.2 demonstrates the setup for our real-world DoubleBin experiment. We employ 4 Azure Kinect cameras to capture multi-view point clouds, minimizing the observation occlusion. We use two plastic bins with dimensions similar to the simulation ones, albeit with slight shape differences.

We use a Franka Emika robot equipped with a Franka hand for our experiments. We replace the original Franka hand fingertips with the Festo DHAS-

GF-60-U-BU fingertip for improved compliance during contact.

B.4.2 Observation and Goal Processing

Our input point cloud to the policy contains "flow", e.g. vectors of correspondences between the observation point cloud and each point's corresponding point in the goal point cloud. The computation of flow requires us knowing the transformation between the current observation and the goal. In order to estimate this transformation in the real-world, we use point cloud registration. Specifically, this process involves:

1. **Global registration** using RANSAC with FPFH features.
2. **Local refinement** via Point-to-Plane ICP. We only match the object shape, which empirically produces more robust performance than matching both the shape and the color.

We predetermine 4 goal poses for each object by placing them at 2 random positions inside each of the two bins. The robot operates autonomously across episodes without manual intervention. When calculating the success rates, we mark episodes with "fake" successes (the episode fails but the agent believes it as a success due to point cloud registration failure) as failures.

B.4.3 Primitive State Estimation

In real-world experiments, we also estimate the primitive state *grasped*. The state is set to $true$ when the camera detects the object's lowest point is at least 4cm above the bin. Conversely, the state switches to $false$ as soon as the gripper releases.

B.4.4 Accuracy and Repeatability

While our method demonstrates promising results in real-world scenarios, there are several sources of error that impact the accuracy and repeatability of the system:

1. **Camera calibration errors:** Inaccuracies in camera calibration can lead to misalignment between the perceived and actual positions of objects in the workspace. Right now the camera calibration error is at around 0.3cm. This can affect the precision of primitive actions, especially for tasks requiring high accuracy.
2. **Robot controller inaccuracies:** The current implementation uses roughly tuned controller parameters. The current average controller error is around

0.6cm. This error tends to be higher on long-distance movements, resulting in imprecise movements This is particularly noticeable in tasks requiring fine manipulation or when executing primitives that demand high precision.

3. **Depth measurement noise:** The consumer-grade depth cameras used in our setup introduce noise in the point cloud observations. This can cause the policy to occasionally select points in empty space, leading to unsuccessful or inefficient actions.

4. **Sim-to-real discrepancies:** Differences in object and background geometries between simulation and reality contribute to the sim-to-real gap. These discrepancies can affect the generalization of learned policies when transferred to the real world.

To address these challenges and improve system performance, we propose the following solutions:

1. **Industrial-grade RGBD cameras:** Upgrading to high-quality industrial cameras would significantly reduce sensor errors, particularly depth measurement noise. This would provide more accurate and reliable point cloud data, enabling more precise primitive execution.

2. **Enhanced robot controller tuning:** Implementing a more systematic and thorough tuning process for the robot controller parameters could substantially improve control accuracy. This may involve using advanced optimization techniques or adaptive control methods to achieve more precise movements across different tasks and object interactions.

3. **Pretrained vision models:** Leveraging large pretrained vision models for RGB feature extraction could potentially mitigate sim-to-real gaps. These models, trained on diverse datasets, may provide more robust and generalizable features, helping the system better adapt to real-world variations in objects.

4. **Data augmentation:** Incorporating more diverse and realistic synthetic data during training, including variations in object geometries, textures, and lighting conditions, could help bridge the sim-to-real gap. This approach could be complemented by domain randomization techniques to further enhance robustness.

5. **Online adaptation:** Implementing an online adaptation mechanism could allow the system to fine-tune its behavior based on real-world feedback, potentially overcoming some of the limitations imposed by sim-to-real discrepancies.

These enhancements could all theoretically improve the accuracy and repeatability of our method in real-world applications. Their exact effect needs to be determined through further experiments.

B.5 Extended Discussion with Related Work

B.5.1 Compared to Zhou et al. [57]

This previous work only demonstrated a spatially-grounded action space with one type of primitive and one task; we are the first to demonstrate that spatially-grounded manipulation can be extended to a wide range of tasks.

To achieve this generality, we design 2 additional spatially-grounded primitives (grasp and move-to) and 2 non-spatially-grounded primitives (open gripper and move-delta) to enable a range of tasks to be achieved. We have presented a set of spatially-grounded primitives that we show to be sufficiently general to be applied to a wide range of tasks and can be adopted by others in the community.

How to spatially-ground each of these primitives is non-obvious and we experimented with different choices of spatial grounding before we found this set of spatially-grounded primitives that work well and cover a range of tasks.

Further, there are multiple ways in which one can imagine extending [57] to try to incorporate multiple primitives. In our current approach, we predict a separate Q-value for each point and each primitive, and we choose the point and primitive combination with the highest Q-value. We have added an experiment to compare this approach to an alternative: Similar to RAPS [6], we extend the actor's action space to include the log probability (logit) of selecting each primitive; we can treat the logits as a continuous action output which we update using reinforcement learning (e.g. TD3) and then select the action based on a softmax over these logits. This second approach is similar to the approach used in RAPS and this baseline can be viewed as a spatially-grounded variant of RAPS. We conduct experiments to test this approach, as shown in the Appendix B.2.7. The results show that while the alternative method solves the easier tasks, our method achieves significantly better performance on the more challenging double-bin task.

B.5.2 Compared to Feldman et al. [7]

Our method has additional primitives that enable our method to achieve a wide range of tasks, compared to the single task shown in [7]. As mentioned above, designing a set of spatially-grounded primitives that could achieve a wide range of tasks was not straight-forward in our experience.
For example, the method in [7] only includes 2 primitives, referred to as "shift" and "grasp" (which are similar to our "poke" and grasp" primitives). These primitives would not be capable of achieving our double-bin task which requires placing an object into a specific 6D pose. For this task, we needed the additional primitives that we included: "move to", "move delta", and "open gripper". Other differences compared to [7] include:

1. **2D vs. 3D**. The previous work [7] provides a solution based on the

assumption that their spatial grounding can be represented in a top-down 2D pixel space, while we provide a generic solution for manipulation in full-3D-space. The 2D representation limits both the set of available locations and the action flexibility: the set of points that can be selected in [7] are limited to only those visible from a top-down camera, whereas our method can select any visible point on the object surface (such as on the sides of an object); furthermore, the 2D pushing actions used in the previous work are insufficient for more dexterous non-prehensile manipulation motions such as flipping an object, which require 3D pushing actions as we use in our method.

2. **Limited horizon**. The previous work [7] assumes their task can be completed within 2 primitive steps. It requires a separate value function for each step, which is not scalable for longer horizon-tasks like our Double-Bin task. Additionally, the limited two-step horizon restricts the exploration of multiple solution modalities, as it yields only one possible solution.

3. **Limited experiments**.

    - Tasks diversity. The prior work [7] focuses on how to pick up objects, with two specialized primitives designed for this task (grasping and 2D shifting). We focus on how an agent can discover longer-horizon strategies using a diverse set of primitives on more general manipulation tasks.
    - Object generalization: [7] shows their method working on limited objects geometry (cubes, spheres, and cylinders). Prior work [57] has shown that the difficulty is much lower when there is limited geometric diversity. In contrast, our method maintains good performance across diverse and even diverse unseen geometries.
    - The prior work [7] does not show any real-world experiments and does not explore how the method can be transferred to real-world.