Appendix

Table of Contents

A Simulation Environment

A.1 Object dataset preprocessing

We use the object models from Liu et al. [35]. 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

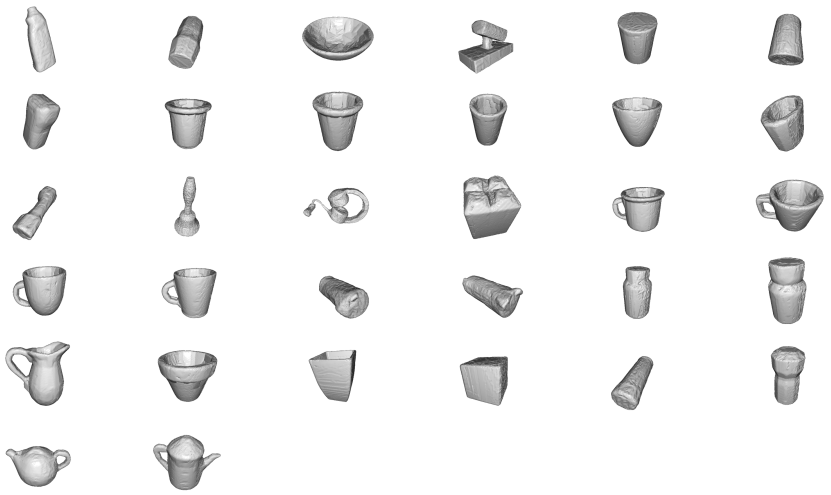

Figure 8: **Training objects.** 32 objects used in training.

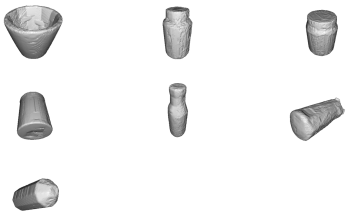

Figure 9: **Evaluation objects (unseen instance).** 7 objects used in unseen instance evaluations. These instances are from the same categories as the training objects.

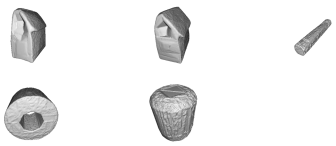

Figure 10: **Evaluation objects (unseen category).** 5 objects used in unseen category evaluations. They come from 4 randomly chosen categories.

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. 8, Fig. 9 and Fig. 10 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.

A.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.3 Representing the goal as per-point goal flow

As mentioned in Section 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 C.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.

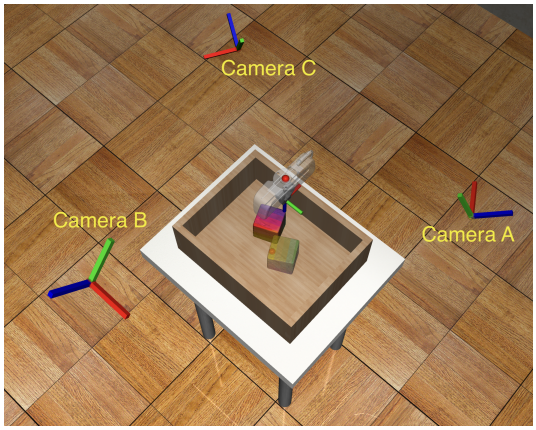

Figure 11: Camera locations in simulation.

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. In the real robot experiments, we estimate the correspondences using point cloud registration methods (see Appendix D 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" [36] or "PLoss" [37] used in previous work, except here we use it as a reward function.

A.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.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 C.6.

A.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. 11). 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 5.3 and Appendix A.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 cal-

culated according to Section A.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[33] during simulation. Details of obtaining segmentation labels in real robot experiments are discussed in Appendix D.

A.6 Action

As mentioned in Section 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 D. 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 C.3 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 [1, 2].

B Algorithm and Training Details

B.1 HACMan (Ours)

HACMan is implemented as a modification on top of TD3 [6] 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 3. 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. 4.

Table 3: Hyperparameters.

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

B.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,

$$J_{\text{dist}} = \begin{cases} -\lambda_{dist}(d_{\text{min}} - 0.05), \ d_{min} > 0.05 \text{ m} \\ 0, \ otherwise \end{cases} \tag{8}$$

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 4; any hyperparameter not listed in Table 4 is the same as our method (Table 3).

Table 4: Baseline-specific Hyperparameters.

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

C Supplementary Experiment Results

C.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. 12.

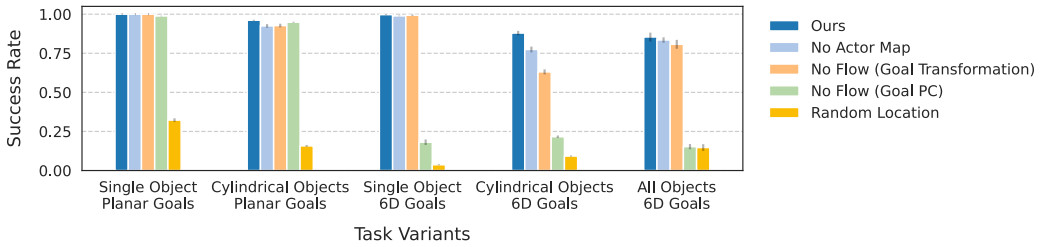

Figure 12: **Additional ablations.** All of the components of our method are essential to achieve the best performance when the task becomes more difficult.

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. 12, 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 5.3 and Appendix A.3, in our method, we represent the goal by first computing the 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 [17, 18]. We use an additional segmentation label in the point features to distinguish the goal points from the observed points. From Fig. 12, 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. 6 (middle).

C.2 Training curves and tables

In this section, we include the full training results for all the methods with additional task variants. Fig. 13 and Fig. 14 include the training curves for the baselines and the ablations. Table 5 and Table 6 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. 4 and Fig. 12.

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 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.

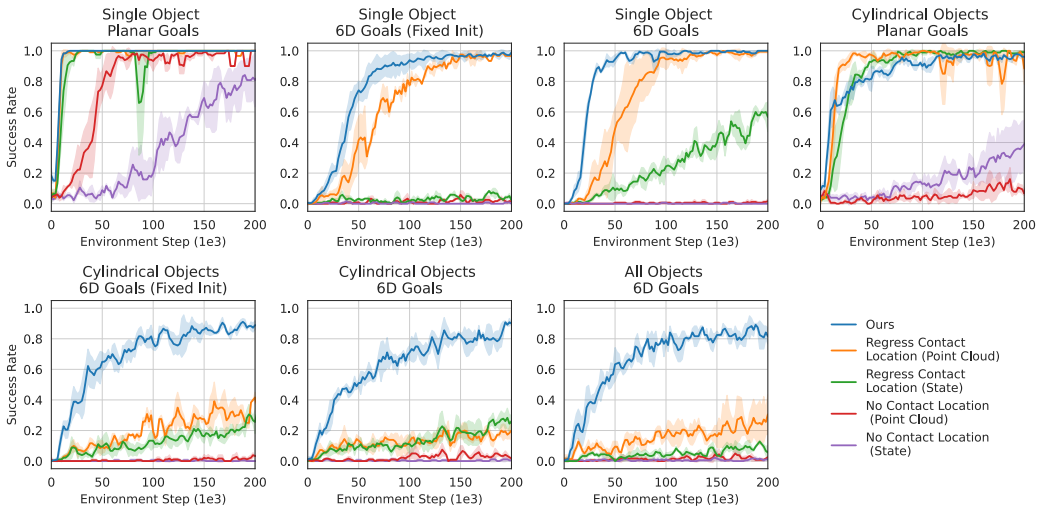

Figure 13: **Baselines.** It shows success rates on the train dataset over environment steps. The shaded area represents the standard deviation across three training seeds.

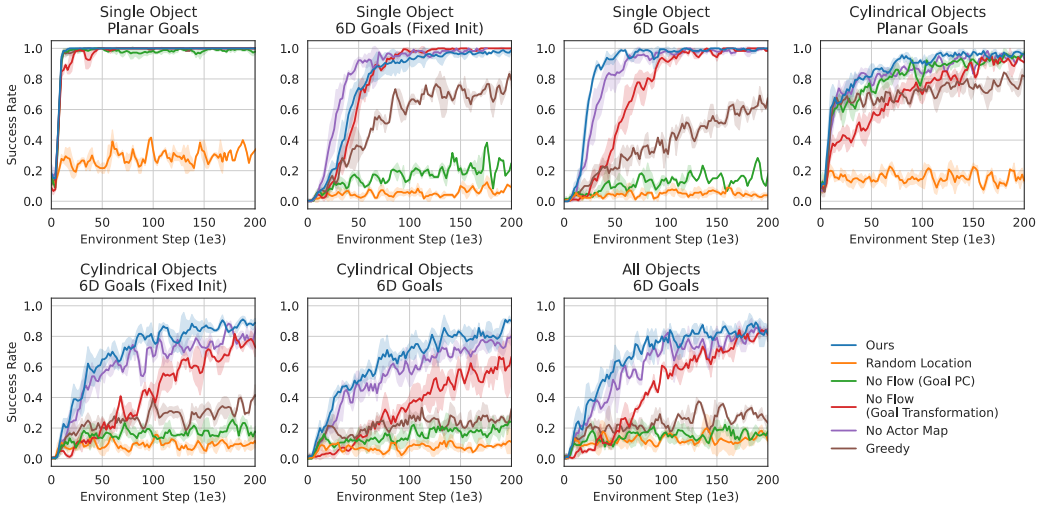

Figure 14: **Ablations.** It shows success rates on the train dataset over environment steps. The shaded area represents the standard deviation across three training seeds.

Table 5: **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**				
		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 6: **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.

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**

C.3 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 7 presents the results of our evaluation for each modified action space. The success rates are reported along with their corresponding standard deviations. We find that including 6D motion in the motion parameters allows for more dexterous movements. These enhanced motion capabilities improve the efficiency of HACMan's actions, resulting in higher success rates. However, the addition of 6D contact or contact offset does not seem to provide significant benefits to HACMan's performance. Instead, these modifications lead to slower training speeds and lower success rates.

Table 7: **Success rates of different motion parameters in HACMan.** All methods are evaluated on all train objects with 6D goals.

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

C.4 Experiments on 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 8, as expected, the task becomes more challenging when there are more obstacles in the bin. As illustrated in Fig 15, the policy tends to push the object directly toward the goal by pushing the scene object aside.

Table 8: **Success rates under different cluttered scenes.** All methods are evaluated with 6D goals.

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

Figure 15: **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 16: **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.

C.5 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 16). 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).

C.6 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 17, 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.

C.7 Per-category result breakdown

Fig. 18 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.

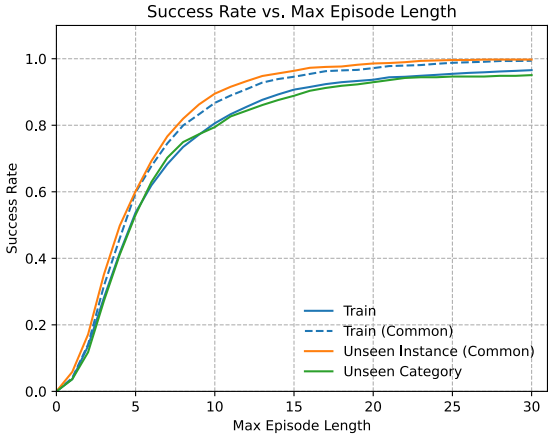

Figure 17: **Success rates at various maximum episode lengths.** This line plot shows the success rates of HACMan evaluated on the four datasets. It is worth noting that the success rates for Unseen Instance (Common) and Train (Common) are marginally higher compared to Train and Unseen Category, similar to the pattern in Table 2.

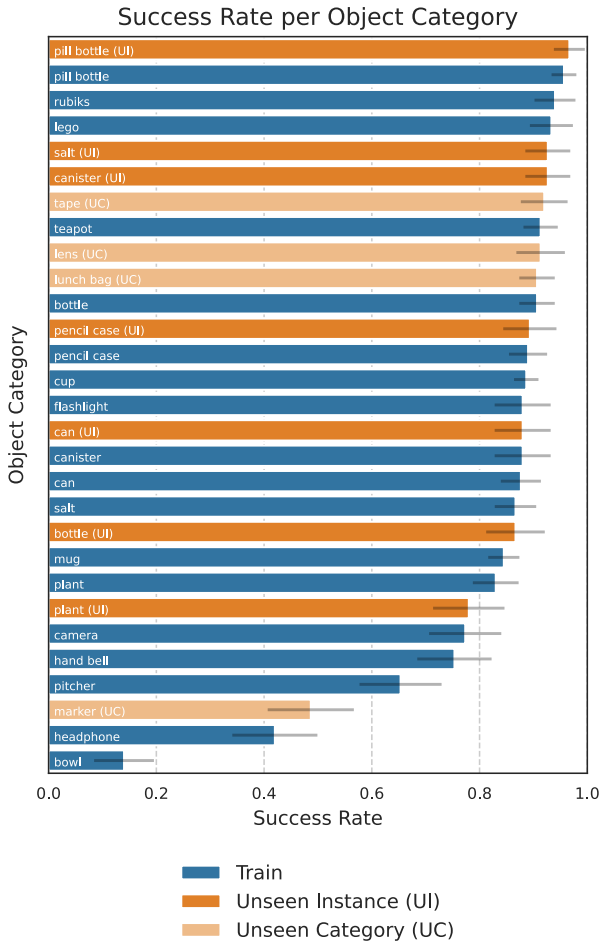

Figure 18: **Results breakdown.** Object categories in the unseen instance set (orange) can be compared to the same object categories in the train set (blue) to see the level of instance generalization.

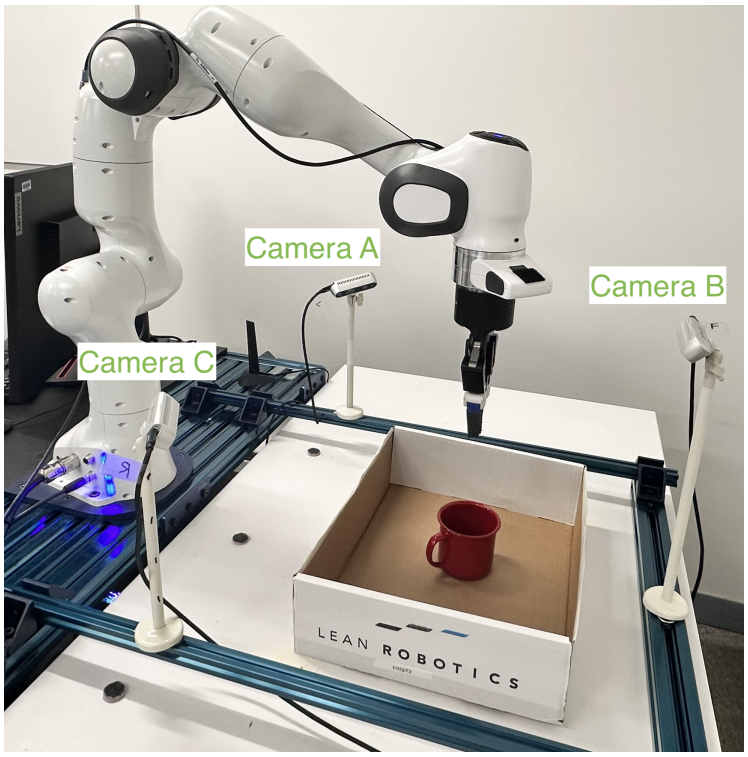

Figure 19: Real robot setup.

D Real Robot Experiments

D.1 Real robot setup

The robot setup is shown in Fig. 19. We use three cameras on the real robot to get a combined point cloud. We follow a similar procedure as Appendix A 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.4) are both calculated automatically.

For the real robot experiments, we use the policy trained in the "6D goals" configuration with 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.

D.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.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. 7 in the main text reports the actual success. We include both success metrics in Table 9 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.

Table 9: **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.

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%**

D.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 9 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 simulation.

In addition, the object dynamics might be different from 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.

E More discussion on the related work

E.1 Compared to Chen et al. [17, 18]

Our work is substantially different from Chen et al. [17, 18] from the follow aspects:

Approach: The approach in Chen et al. [17, 18] 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. [18]). 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. [17, 18] takes point cloud observation, it is trained using imitation learing 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 [17, 18]. 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. [17, 18] only considers orientation.

E.2 Compared to Cheng et al. [1], Hou and Mason [2]

Unlike Cheng et al. [1], Hou and Mason [2], our method does not rely on quasi-static assumptions, 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. As shown in our experiments, our approach exhibits superior motion complexity and object diversity compared to previous work.