

Appendices

A Additional Results

A.1 Sensitivity analysis on physical parameters

In addition to the evaluation on policy generalization in Section 5.4, we modify the important physical parameters one at a time to understand the sensitivity of the policy performance to these parameters (Figure 9). We compare the same baselines as Section 5.4: policies trained over a fixed environment (*Fixed Env*), policies trained with ADR (*With ADR*) and open-loop trajectories generated by rolling out the fixed env policy in the default environment (*Open Loop*). The ranges of parameters are chosen to create a performance drop for all the baselines as a stress test. Similar to what we observe in Section 5.4, the policy can cover a wider range of physical parameters with closed-loop execution and with ADR.

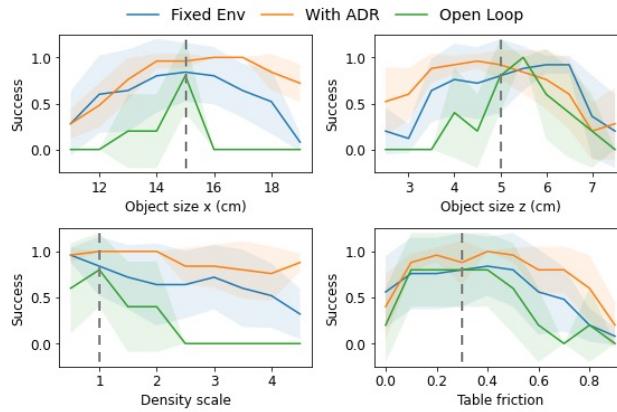


Figure 9: We evaluate the generalization of policies by changing one parameter at a time. The dashed lines indicate the default values of these parameters in the fixed environment.

A.2 Sensitivity analysis on object pose estimation noise

The proposed system takes the 6D object pose as policy input. In the real world, object pose estimation might be noisy. In this section, we evaluate the policies trained with ADR with different levels of pose estimation noise for each dimension of the 6D object pose (Figure 10). During evaluation, for each timestep across the episode, we sample a scalar noise from a Gaussian distribution $\mathcal{N}(\mu = 0, \sigma = x)$ and add it to one dimension of the object pose. The standard deviations $\sigma = x$ are shown as the x-axis in the plots. The shaded area indicates the standard deviation of the success rates across seeds.

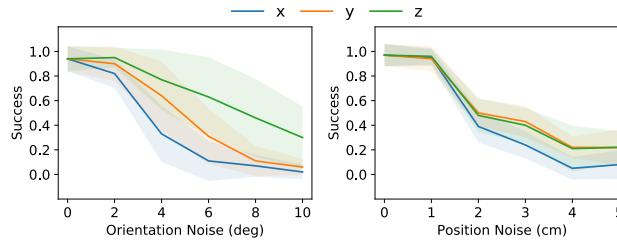


Figure 10: We evaluate the sensitivity of the ADR policies on object pose estimation noise.

A.3 Reward term weights

The reward function shown in Equation 1a is composed of three terms with weights α_1 , α_2 and β . α_1 and α_2 weight the translation and rotation error between the target grasp and the current end-effector. β weights the target grasp occlusion penalty which is to penalize the agent if the target grasp

configuration is in collision with the table. We use $\alpha_1 = 50$, $\alpha_2 = 2$, $\beta = 200$ in all the experiments. In this section, we train the policies with different weight values to see how much reward tuning is required to achieve reasonable performance for the occluded grasping task. Figure 11 below shows that the policy is not too sensitive in most of the case we tested except that a higher $\alpha_1 = 70$ leads to a 50% drop in performance.

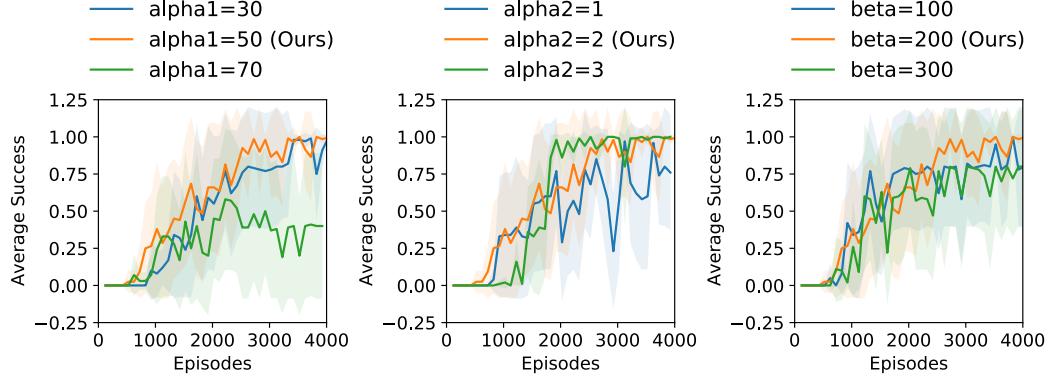


Figure 11: Training curves with different reward weights. For each plot, we train the policies by changing one of the weight terms to three different values.

B Implementation Details

B.1 Simulation environment

We build the simulation environment with Robosuite [38] which uses the MuJoCo simulator [39]. Each episode has a length of 40 timesteps which corresponds to 20 seconds of real time execution. At the beginning of each episode, we set the robot arm to an initial joint configuration with Gaussian noise in the joint angles of 0.02 rad. We use a box-shaped object in the simulation environment. The dimensions of the box are randomized in the ADR experiments. One important note on the simulator environment is the parameters of the MuJoCo solver. We notice that MuJoCo sometimes creates unrealistic contacts with the default solver. We reduce the simulation solver timestep from the default value of 0.002 to 0.001 and set the ‘noslip iterations’ to 20 which significantly improved simulation quality on contacts.

B.2 Grasp configurations

In this work, we focus on grasping large objects from the side because this is a task that may demonstrate the benefits of extrinsic dexterity. For single grasp experiments, a default grasp location is shown in Figure 1. In multi-grasp experiments, the grasps are sampled from a distribution shown in Figure 6. The grasps are sampled along the side of the box and they are 2 cm away from the edges. These grasp configurations are supposed to be the input to our proposed system, and could be replaced by other grasp generation methods.

B.3 Success rate calculation

In simulation, the success of the task is computed as $\mathbb{1}(\Delta T < 3 \text{ cm}) \cdot \mathbb{1}(\Delta\theta < 10 \text{ deg})$ at the end of an episode. As defined in Section 3, ΔT is the position difference between the end-effector and the target grasp and $\Delta\theta$ is the orientation difference. The success is defined in this way because we focus on reaching the desired grasp. One alternative is to evaluate the final grasping success by closing the gripper and lift the object. However, this will increase the simulation time during training. To confirm that the pose difference is a good proxy for the final grasping success, we evaluated a trained policy and verified that if the robot closes the gripper at the end of a successful episode according to the pose difference metric, it is able to lift the object 100% of the time.

For the real robot experiments, we evaluate success by closing the gripper and lifting the object; if the object was successfully lifted, we will mark it as a successful episode.

B.4 Observation and action space

As mentioned in Section 4.2, the observation includes a target grasp configuration in the object frame Og , the pose of the end-effector in the world frame WE and the object pose in the world frame WO . One implementation detail is that we also include the pose of the end-effector in the object frame ${}^OE = ({}^WO)^{-1}({}^WE)$ because we found that it sometimes speeds up learning. Each pose is represented as a 3D translation vector and a 4D quaternion representation of the rotation.

The action space of the policy is the delta pose of the end-effector ΔE in its local frame represented by a vector of translation $p \in \mathbb{R}^3$ and a 3D vector of rotation $q \in SO(3)$ with axis-angle representation. An outline of the policy execution pipeline is shown in Figure 12. ΔE is then passed into a collision check function to form a desired pose E_d which will be sent to a low-level controller.

B.5 Low-level controller

Handling joint limit: Although we may use nullspace in the operational space controller to avoid reaching joint limit, in practice, certain desired end-effector poses still reach joint limits that cannot be avoided by nullspace. Thus, we handle the joint limit in the following way. If the corresponding joint configuration of the desired pose is going to reach joint limits, we will overwrite the policy action to output the desired pose of the previous timestep to the low-level controller. In detail, we use the Jacobian J to estimate the joint configuration of the desired pose:

$$\theta_{joints}^{t+1} = \theta_{joints}^t + J^{-1} \cdot \Delta E \quad (2)$$

where θ_{joints} are the joint angles and ΔE is the output of the policy. If any joint in θ_{joints}^{t+1} is close to the limit, the low-level controller will use the previous desired pose E_d instead.

Parameters of the Operational Space Controller: We use $K_p = 300$ for position error, $K_p = 30$ for orientation error, and $K_d = \sqrt{K_p}$. These values are chosen by making sure that the real robot is compliant enough to safely collide with the object and the bin without damage. In Figure 4, the baseline of ‘‘High-gain OSC’’ uses $K_p = 600$ for position error, $K_p = 60$ for orientation error, and $K_d = \sqrt{K_p}$. This baseline with less compliance is not only slower to train in the simulation, but also not safe to execute on the real robot for our task which involves rich contacts and relies on environment constraints. During our initial experiments, with the high-gain OSC, the robot deforms the object and the bin surface.

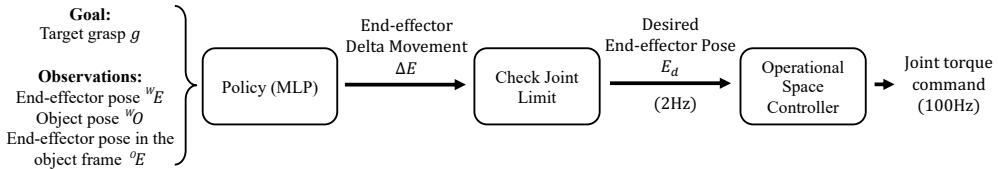


Figure 12: Outline of policy execution: Given the observation, the policy outputs an end-effector delta movement. If the desired pose is within the joint limit of the robot, it will be sent to the low-level controller which operates at a higher frequency.

B.6 Multi-Grasp Training with Curriculum

Here are more details on multi-grasp training. When the success rate of policy on a boundary case of the training range is above 0.8, it will expand the range of grasps by 0.25 (See Figure 6 for parameterizations of the grasp configurations). For example, if the policy is currently training with grasps [1, 2], and the success rate evaluated at grasp ID 1 is above 0.8, the new training range will be [0.75, 2]. This is following a similar procedure as Automatic Domain Randomization, but randomizing goals instead of simulation parameters.

B.7 RL Training

We use Soft Actor Critic [37] to train the RL policy with the implementation from rlkit (<https://github.com/rail-berkeley/rlkit>). Hyperparameters for SAC training are included in Table 2. Since the task is conditioned on the target grasp as a goal, we use Hindsight Experience Replay [40] for all the experiments with 60% original goals and 40% of the goals sampled from the same rollout.

We compare the policies across 5 random seeds of each method and plot the average performance with standard deviation across seeds. We use 10 episodes for each evaluation setting.

Table 2: Hyperparameters for RL training.

| Hyperparameters | Values |
|-------------------------------|---------------------|
| Optimizer | Adam |
| Learning rate - Policy | 1e-3 |
| Learning rate - Q-function | 5e-4 |
| Networks | [512, 512, 512] MLP |
| Batch size | 256 |
| Nonlinearity | ReLU |
| Soft target update (τ) | 0.005 |
| Replay buffer size | 1e6 |
| Discount factor (γ) | 0.99 |
| HER rollout goals | 40% |

C Automatic Domain Randomization

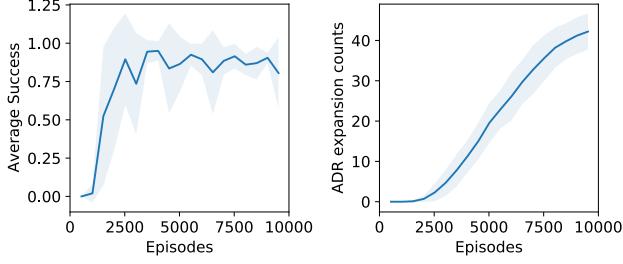
As discussed in Section 4.6, we use Automatic Domain Randomization [14] to improve policy generalization across environment variations. In ADR, the policy is first trained with an environment with very little randomization, and then we gradually expand the variations based on the evaluation performance. For a set of environment parameters λ_i , each λ_i is sampled from a uniform distribution $\lambda_i \sim U(\phi_i^L, \phi_i^H)$ at the beginning of each episode. During training, the policy will be evaluated at these boundary values $\lambda_i = \phi_i^L$ or $\lambda_i = \phi_i^H$. If the performance is higher than a threshold, the boundary value will be expanded by an increment Δ . For example, if the performance at $\lambda_i = \phi_i^H$ is higher than the threshold, the training distribution becomes $\lambda_i \sim U(\phi_i^L, \phi_i^H + \Delta)$ in the next iteration. Compared to directly training the policy with the entire variations, Automatic Domain Randomization can reduce the need of manually tuning a suitable range of variations for each environment parameter.

Table 3 summarized the simulation parameters in the experiment. All the parameters are uniformly sampled from these ranges at the beginning of each episode. The ranges of the parameters start from a single initial value and gradually expand to a wider range according to the pre-specific increment step $+\Delta$ on the upper bound and the decrement step $-\Delta$ at the lower bound.

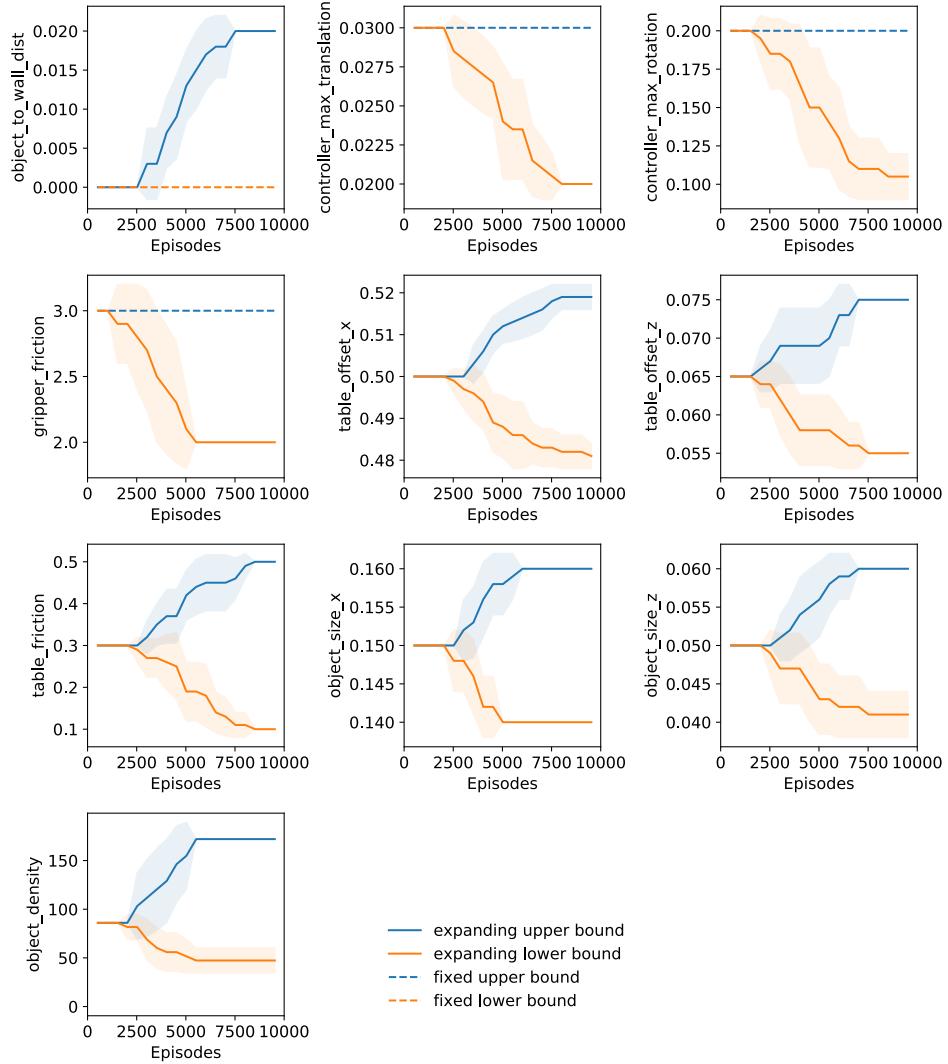
Table 3: Simulation parameters in Automatic Domain Randomization

| | Initial Value | $+\Delta$ | $-\Delta$ | Final Range |
|------------------------------|---------------|-----------|-----------|----------------|
| Object size x (m) | 0.15 | 0.01 | -0.01 | [0.14, 0.16] |
| Object size z (m) | 0.05 | 0.01 | -0.01 | [0.04, 0.06] |
| Table friction | 0.3 | 0.1 | -0.1 | [0.1, 0.5] |
| Gripper friction | 3 | / | -1 | [2, 3] |
| Object Density (g/m^3) | 86 | 86 | 43 | [43, 172] |
| Action translation scale (m) | 0.03 | / | -0.005 | [0.02, 0.03] |
| Action rotation scale (rad) | 0.2 | / | -0.05 | [0.1, 0.2] |
| Initial distance to wall (m) | 0 | 0.01 | / | [0, 0.02] |
| Table offset x (m) | 0.5 | 0.01 | -0.01 | [0.48, 0.52] |
| Table offset z (m) | 0.07 | 0.01 | 0.01 | [0.055, 0.075] |

We include the training plots of the ADR policies in Figure 13. Dashed lines in Figure 13 indicate fixed parameter boundaries where we do not intent to expand. The final ranges are used when we sample 100 environments for evaluation in Section 5.4.



(a) Overall training performance of the ADR policies: Success rate over the entire training range (left) and total number of expanded parameter boundaries.



(b) Training progress of individual ADR parameters. Each plot for the physical parameters has two curves indicating the upper and lower bound of the expanded training range. Dashed lines indicate fixed parameter boundaries where we do not intent to expand.

Figure 13: Training curves for the ADR policies.

D Real robot experiment

In this section, we include more details and discussion for the real robot experiments. Quantitative results can be found on the website ² where we include all the videos for the real robot experiments, video examples of failure cases, recovery behaviors and ICP results.

D.1 Implementation details

The robot setup is shown in Figure 14. The code for controlling the real robot is built on top of FrankaPy [41]. The policies are trained in the simulator and zero-shot transferred to a physical Franka Emika Panda robot. For the real robot experiments, we train a policy in the XZ plane from the side view to reduce the sim2real gap of the policy, since the motion is mostly in the XZ plane for the side grasp.

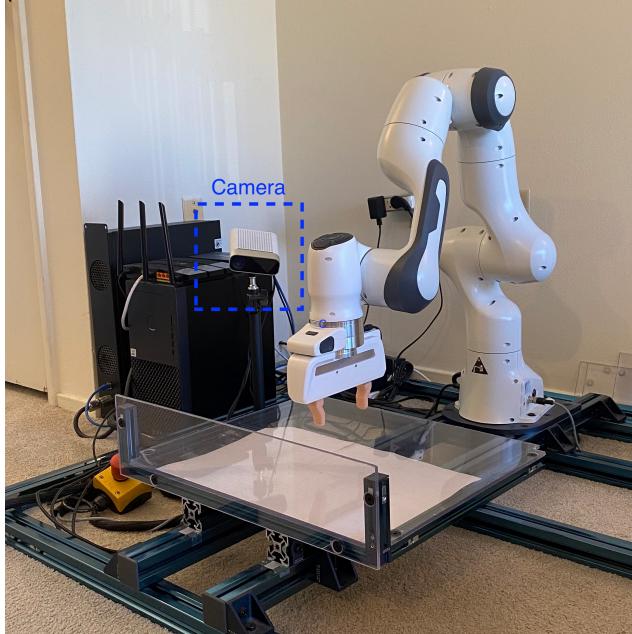


Figure 14: Robot setup. We use one Azure Kinect camera for object pose estimation.

Sim2Real gap of the low-level controller: We observe a noticeable sim2real gap on the low-level controller when deploying the policy. The same command of moving the end-effector to a certain pose in free space may not have the same resulting movement. This is a combination of two factors: First, there is a significant discrepancy between the robot model in simulation and the real robot. The real robot has more damping and friction on the joints. Second, we use a compliant controller for this task, which is more susceptible to the noise in the system. As a result, the real robot always executes a smaller delta movement than the simulator. To compensate for this sim2real gap, we slightly increase the action scale and reduce the policy execution rate from 2Hz to 1Hz. Both of these changes will allow the real robot to compensate for the smaller movement caused by the damping and friction of its joints. Note that the sim2real gap still exists after these changes. However, the remaining gap could be further compensated by using a closed-loop policy. During our experiments, we first use the default object Box-0 to tune the controller until we observe several successes in a row. After that, we keep the same controller setting for the entire evaluation process.

Defining object pose: The pose of a box can simply be defined at the center of its volume and with the axes defined parallel to the edges. For non-box object, we define the pose to be the center of its bounding box. We scan the non-box objects into point clouds with the Qlone app on the phone (Figure 15 top row). To obtain the bounding box, we first run Principle Component Analysis of the scanned object to get the principle axes. Then, we take the min and max values along the axes to

²<https://sites.google.com/view/grasp-ungraspable>

get the dimension of the bounding box. The axes are then aligned to global axes based on the initial pose (Figure 15 middle row).

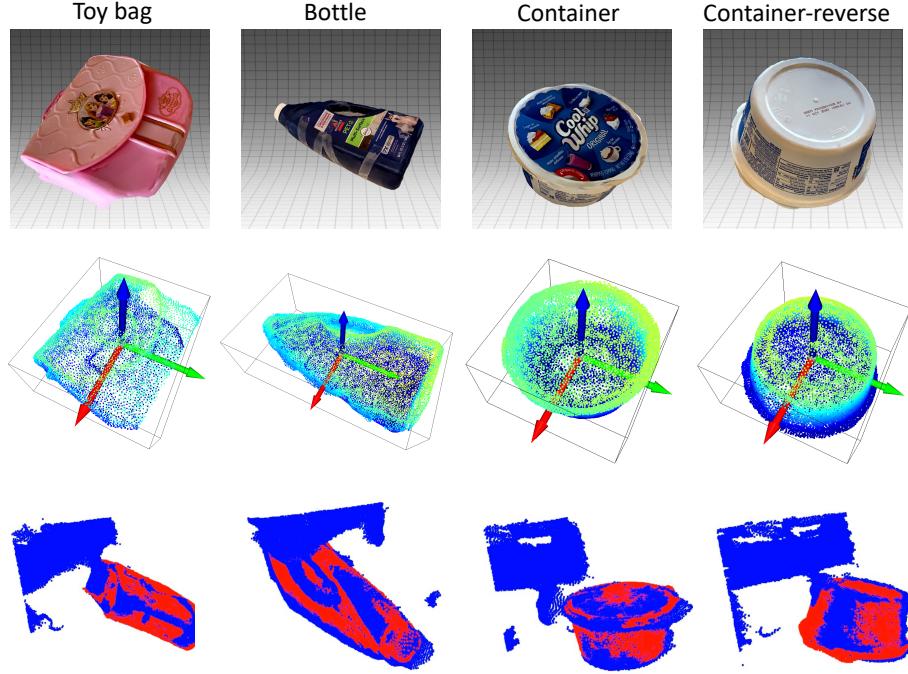


Figure 15: Illustrations of pose estimation pipeline for the non-box objects. The top row shows the scanned object model. The middle row shows bounding box calculation and pose definition. The last row shows an example of ICP.

Pose estimation with Iterative Closest Point: To get the pose of the object as the observation of the policy, we use Iterative Closest Point (ICP) which matches the current point cloud to a template point cloud of the object [32]. We use the implementation from Open3D. For box objects, we simply create a box shape template with measured size. For non-box objects, we use the scanned object point clouds as mentioned above. Figure 16 shows an example of the results from ICP across an episode. Figure 15 includes examples of ICP results for non-box objects potentially with partial point cloud. More visualizations of ICP results can be found on the website.

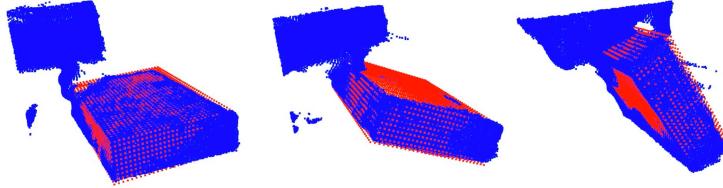


Figure 16: Examples of pose estimation with ICP during an episode. The blue points are the observed point cloud from the camera. The red points are the object template that matches to the observed point cloud using ICP.

D.2 More information on the objects

To emphasize the diversity of the objects and demonstrate the generalization capability of the policy, we include more descriptions on the objects in this section. In Table 4, we highlight the object properties that are out of the ADR training distribution in bold. Box-0 is the default object that we used to calibrate the simulator and to tune the low-level controller.

Table 4: Real robot evaluations with more object information. We highlight the out-of-distribution aspect of the object properties in bold.

| Object-ID | Box? | Surface Material | Bounding Box Dimension (cm) | Weight (g) | Success w/o ADR | Success w/ ADR |
|-------------------|--------|----------------------------|-----------------------------|------------|-----------------|----------------|
| Box-0 | Yes | Cardboard | (15.0, 20.0, 5.0) | 128 | 9/10 | 9/10 |
| Box-0 + 4 erasers | Yes | Cardboard | (15.0, 20.0, 5.0) | 237 | 6/10 | 10/10 |
| Box-0 + 8 erasers | Yes | Cardboard | (15.0, 20.0, 5.0) | 345 | 3/10 | 4/10 |
| Box-1 | Yes | Cardboard | (15.4, 29.2, 5.8) | 130 | 5/10 | 8/10 |
| Box-2 | Yes | Cardboard | (15.3, 22.2, 7.4) | 113 | 2/10 | 9/10 |
| Box-3 | Yes | Cardboard with tape | (16.5, 24.5, 5.2) | 50 | 0/10 | 7/10 |
| Toy Bag | Almost | Silicone | (16.6, 14.5, 7.1) | 203 | 8/10 | 7/10 |
| Bottle | No | Plastic | (16.3, 28.8, 9.0) | 112 | 0/10 | 8/10 |
| Container | No | Plastic | (14.7, 14.7, 8.1) | 61 | 0/10 | 10/10 |
| Container-reverse | No | Plastic | (14.7, 14.7, 8.1) | 61 | 0/10 | 6/10 |
| Average | | | | | 33% | 78% |

The policy trained with ADR can generalize across physical properties such as weight and surface friction. We stress test Box-0 with additional weights by putting four or eight erasers inside of the box. The erasers can move in the box during execution, which is not modeled in simulation. Although we do not have access to the true friction coefficient between the object and the table, the difference in surface friction results in qualitatively different behavior of the object even among the cardboard boxes. For example, Box-3 has tape on its surface which has much higher friction than the others cardboard boxes. It tends to stick to the wall during execution. The toy bag has a similar cross section as the box but the material is very different.

We also evaluate the policies with objects that are not similar to a box shape including a bottle and a container. Due to the difference in shape, both objects result in different dynamics during execution. In addition, with the same container object, starting it from different initial poses will also lead to different object pose distribution. Videos can be found on the website. Nonetheless, the policy trained with ADR shows reasonable generalization across these non-box objects.

D.3 Failure cases

In this section, we include discussions on the failure cases of the evaluation. We categorize the failure cases into the following categories and discuss the potential reasons:

A failure case that happens before the initial contact:

- **Missing initial contact:** The robot is not able to reach the initial contact of the object to rotate it. This is mostly due to the noise in pose estimation and the variations in object dimension.

Failure cases that happen during the rotation:

- **Object drops during rotation:** The object drops to the table during rotation. One potential reason for this failure case is that the finger slips on the object during rotation. Another potential reason for this failure case is the insufficient rotation of the low-level controller due to the sim2real gap (See Section D - Sim2Real gap of the low-level controller). In the “dropping” strategy, the policy is supposed to rotate object and then let it drop on the bottom finger. Before the dropping happens, the gripper needs to be rotated until the bottom finger is below the object. Otherwise, the bottom finger will not be able to catch the object and the object directly drops to the table.
- **Repeated rotation:** The robot repeatedly rotates and drops the object. This is different from the previous failure case because the robot moves down with the object at the same time. Our hypothesis for this failure case is that the policy gets stuck in a loop in the MDP.
- **Joint limit:** The robot hits a joint limit and the policy gets stuck at the joint limit.

Failure cases that happen after the rotation:

- **Unexpected object dynamics:** When the robot rotates the object, the object might move in unexpected ways. This mostly happens for the non-box objects.

- **Stop reaching:** Following the “standing” strategy, the policy successfully rotates the object to a stable pose on the side of the object. However, it cannot reach the final grasping pose. The gripper tries to move down to reach the pose but it collides with the object due to the unexpected object dimension.
- **Timeout:** Since we use a fixed episode length during evaluation, sometimes the policy does not have enough time to finish the task although it is very close to a success. This happens when the policy spends time to recover from some failed attempts at the beginning of the episode.

Videos on these failure cases can be found on the website. We summarize the counts of the failure cases in Table 5 and Table 6. The most common failure case for the policy trained with ADR is the repeated rotation. For the policy trained without ADR, the most common failure case is missing the initial contact. Comparing the percentage of the failure cases between Table 5 and Table 6, we observe that percentage of missing the initial contact and the percentage of stopping the reaching motion drops drastically when the policy is trained with ADR because the policy is more robust to variations in shape and dimensions.

Table 5: Failure cases for **Policy w/ ADR** during real robot evaluation. The most common failures include dropping the object during rotation, repeated rotation, and unexpected object dynamics.

| | Initial contact | Object drops | Repeated rotation | Joint limit | Unexpected dynamics | Stop reaching | Timeout |
|--------------------------|-----------------|--------------|-------------------|-------------|---------------------|---------------|-------------|
| Box-0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| Box-0 + 4 erasers | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Box-0 + 8 erasers | 0 | 3 | 3 | 0 | 0 | 0 | 0 |
| Box-1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 |
| Box-2 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| Box-3 | 0 | 0 | 2 | 0 | 0 | 0 | 0 |
| Toy Bag | 0 | 2 | 1 | 0 | 0 | 0 | 0 |
| Bottle | 1 | 0 | 0 | 0 | 1 | 0 | 0 |
| Container | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Container-reverse | 0 | 0 | 0 | 0 | 4 | 0 | 0 |
| Total | 1 | 6 | 8 | 1 | 5 | 0 | 0 |
| Percentage | 4.8% | 28.5% | 38.1% | 4.8% | 23.8% | 0.0% | 0.0% |

Table 6: Failure cases for **Policy w/o ADR** during real robot evaluation. The most common failures include missing the initial contact, repeated rotation and unexpected object dynamics.

| | Initial contact | Object drops | Repeated rotation | Joint limit | Unexpected dynamics | Stop reaching | Timeout |
|--------------------------|-----------------|--------------|-------------------|-------------|---------------------|---------------|-------------|
| Box-0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| Box-0 + 4 erasers | 1 | 0 | 2 | 0 | 0 | 0 | 1 |
| Box-0 + 8 erasers | 0 | 1 | 3 | 0 | 0 | 2 | 1 |
| Box-1 | 2 | 0 | 2 | 0 | 1 | 0 | 0 |
| Box-2 | 1 | 0 | 1 | 0 | 0 | 6 | 0 |
| Box-3 | 10 | 0 | 0 | 0 | 0 | 0 | 0 |
| Toy Bag | 0 | 2 | 0 | 0 | 0 | 0 | 1 |
| Bottle | 6 | 0 | 0 | 0 | 4 | 0 | 0 |
| Container | 0 | 0 | 2 | 0 | 8 | 0 | 0 |
| Container-reverse | 1 | 1 | 6 | 0 | 1 | 0 | 1 |
| Total | 21 | 4 | 16 | 0 | 14 | 8 | 5 |
| Percentage | 30.9% | 5.9% | 23.5% | 0.0% | 20.6% | 11.8% | 7.3% |