Appendix A Evidence indicating transfer to real-world

Due to the scope of the problem and the ongoing pandemic, we limit our experiments to be in simulation. However, we provided evidence indicating that the learned policies can be transferred to the real world in the future in the paper. We summarize this evidence as follows.

Convex decomposition The objects after the convex decomposition still have geometrically different and complex geometries as shown in Figure B.3. The objects in the EGAD dataset are 3D printable. The YCB objects are available in the real world.

Action space We control the finger joints via relative position control as explained in Section 2.1. This suffers less sim-to-real gap compared to using torque control on the joints directly.

Student policies We designed two student policies and both of them use the observation data that can be readily acquired from the real world. The first student policy only requires the joint positions and the object pose. Object pose can be obtained using a pose estimation system or a motion capture system in the real world. Our second student policy only require the point cloud of the scene and the joint positions. We can get the point cloud in the real world by using RGBD cameras such as Realsense D415, Azure Kinect, etc.

Domain randomization We also trained and tested our policies with domain randomization. We randomized object mass, friction, joint damping, tendon damping, tendon stiffness, etc. Table C.4 lists all the parameters we randomized in our experiments. We also add noise to the state observation and action commands as shown in Table C.4. For the vision experiments, we also added noise (various ways of data augmentation including point position jittering, color jiterring, dropout, etc.) to the point cloud observation in training and testing as explained in Section D.5.

The results in Table D.5, Table D.6, and Table 4 show that even after adding randomization/noise, we can still get good success rates with the trained policies. Even though we cannot replicate the true real-world setups in the simulation, our results with domain randomization indicates a high possibility that our policies can be transferred to the real Shadow hand. Prior works [11] have also shown the domain randomization can effectively reduce the sim-to-real gap.

Torque analysis We also conducted torque analysis as shown in Section D.4. We can see that the peak torque values remain in an reasonable and affordable range for the Shadow hand. This indicates that our learned policies are less likely to cause motor overload on the real Shadow hand.

Appendix B Environment Setup

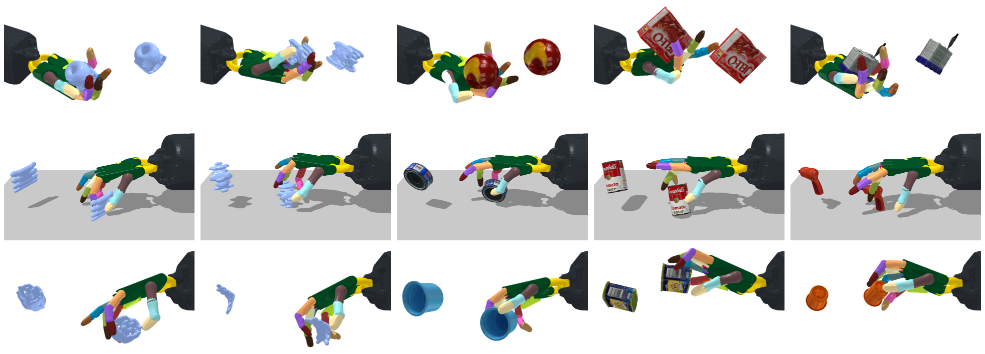


Figure B.1: We learn policies that can reorient many objects in three scenarios respectively: (1) hand faces upward, (2) hand faces downward with a table below the hand, (3) hand faces downward without any table. The extra object in each figure shows the desired orientation.

B.1 State definition

The full state space $\mathbb{S}^{\mathcal{E}}$ includes joint, fingertip, object and goal information detailed in Table B.1. To compute the angle difference between two quaternion orientations α_1 and α_2, we first compute the difference rotation quaternion: $\beta = \alpha_1\alpha_2^{-1}$. Then the angle difference (distance between two rotations) $\Delta\theta$ is computed as the angle of β from the axis-angle representation of β.

Table B.1: Full state $s_t^{\mathcal{E}} \in \mathbb{R}^{134}$ information. Orientations are in the form of quaternions.

Parameter	Description	Parameter	Description	Parameter	Description
$q_t \in \mathbb{R}^{24}$	joint positions	$v_t^f \in \mathbb{R}^{15}$	fingertip linear velocities	$\alpha^g \in \mathbb{R}^4$	object goal orientation
$\dot{q}_t \in \mathbb{R}^{24}$	joint velocities	$w_t^f \in \mathbb{R}^{15}$	fingertip angular velocities	$v_t^o \in \mathbb{R}^3$	object linear velocity
$p_t^f \in \mathbb{R}^{15}$	fingertip positions	$p_t^o \in \mathbb{R}^3$	object position	$w_t^o \in \mathbb{R}^3$	object angular velocity
$\alpha_t^f \in \mathbb{R}^{20}$	fingertip orientation	$\alpha_t^o \in \mathbb{R}^4$	object orientation	$\beta_t \in \mathbb{R}^4$	$\alpha_t^o(\alpha^g)^{-1}$

B.2 Dataset

We use two object datasets (EGAD and YCB) in our paper. To further increase the diversity of the datasets, we create 5 variants for each object mesh by randomly scaling the mesh. The scaling ratios are randomly sampled such that the longest side of the objects' bounding boxes $l_{\max}$ lies in $[0.05, 0.08]$m for EGAD objects, and $l_{\max} \in [0.05, 0.12]$m for YCB objects. The mass of each object is randomly sampled from $[0.05, 0.15]$kg. When we randomly scale YCB objects, some objects become very small and/or thin, making the reorientation task even more challenging. In total, we use 11410 EGAD object meshes and 390 YCB object meshes for training.

Figure B.2 shows examples from the EGAD and YCB dataset. We can see that these objects are geometrically different and have complex shapes. We also use V-HACD [43] to perform an approximate convex decomposition on the object meshes for fast collision detection in the simulator. Figure B.3 shows the object shapes before and after the decomposition. After the decomposition, the objects are still geometrically different.

B.3 Camera setup

We placed two RGBD cameras above the hand, as shown in Figure B.4. In ISAAC gym, we set the camera pose by setting its position and focus position. The two cameras' positions are shifted from the Shadow hand's base origin by $[-0.6, -0.39, 0.8]$ and $[0.45, -0.39, 0.8]$ respectively. And their focus points are the points shifted from the Shadow hand's base origin by $[-0.08, -0.39, 0.15]$ and $[0.045, -0.39, 0.15]$ respectively.

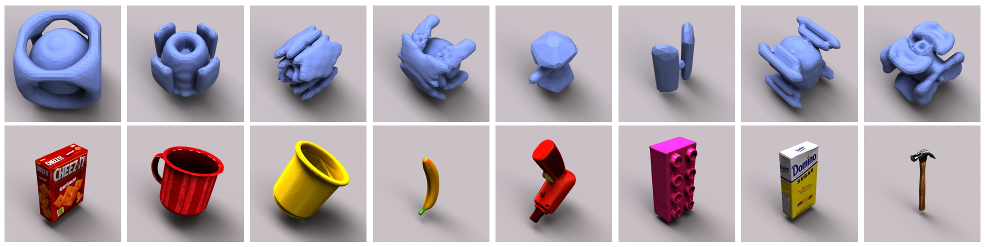


Figure B.2: First row: examples of EGAD objects. Second row: examples of YCB objects.

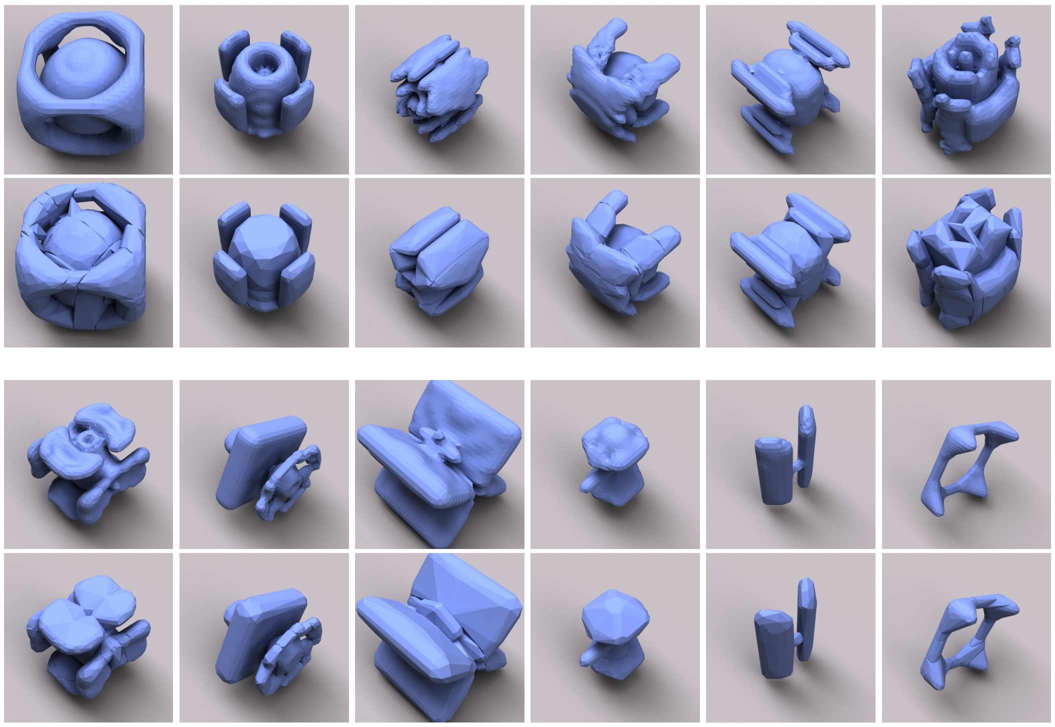


Figure B.3: Examples of EGAD objects. The first and third row shows the visual mesh of the objects. The second and fourth row show the corresponding collision mesh (after V-HACD decomposition).

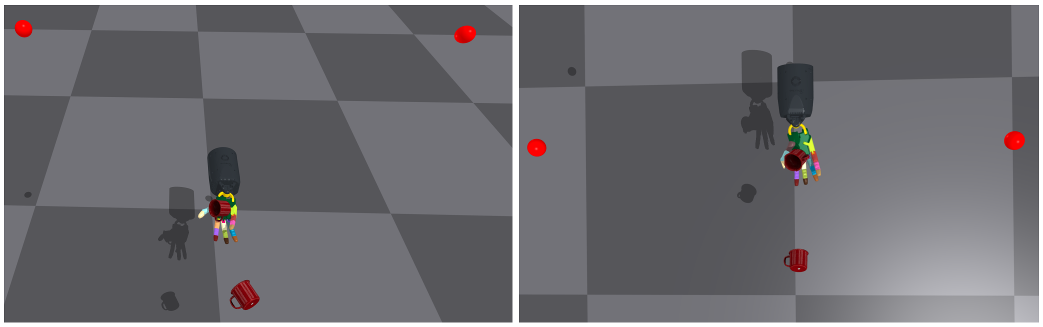


Figure B.4: Camera positions

Appendix C Experiment Setup

C.1 Network architecture

For the non-vision policies, we experimented with two architectures: The MLP policy π_M consists of 3 hidden layers with $512, 256, 256$ neurons respectively. The RNN policy π_R has 3 hidden layers

$(512 - 256 - 256)$, followed by a 256-dim GRU layer and one more 256-dim hidden layer. We use the exponential linear unit (ELU) [44] as the activation function.

For our vision policies, we design a sparse convolutional network architecture (*Sparse3D-IMPALA-Net*). As shown in Figure C.5, the point cloud W_t is processed by a series of sparse CNN residual modules and projected into an embedding vector. q_t and a_{t-1} are concatenated together and projected into an embedding vector via an MLP. Both embedding vectors from W_t and (q_t, a_{t-1}) are concatenated and passed through a recurrent network to output the action a_t.

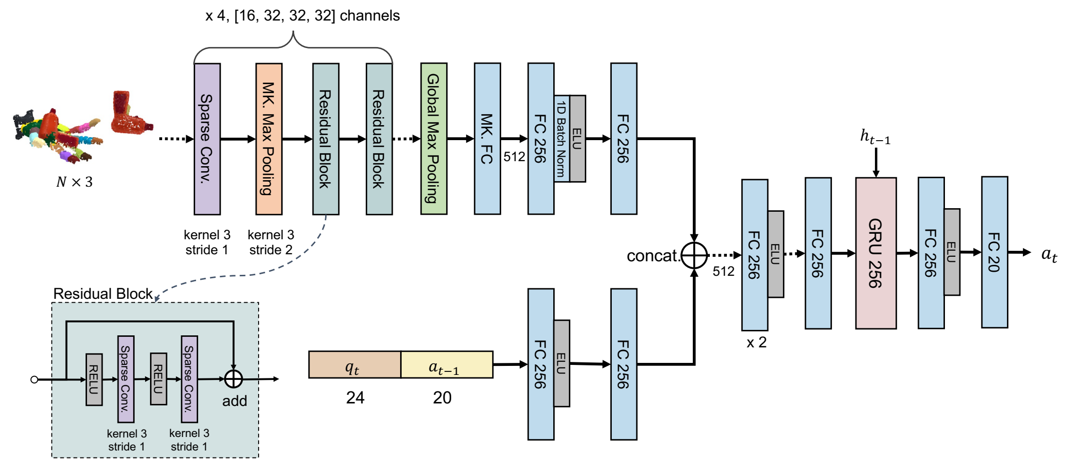


Figure C.5: Visual policy architecture. MK stands for Minkowski Engine. q_t is the joint positions and a_t is the action at time step t.

C.2 Training details

All the experiments in the paper were run on at most 2 GPUs with a 32GB memory. We use PPO [17] to learn $\pi^{\mathcal{E}}$. Table C.2 lists the hyperparameters for the experiments. We use 40K parallel environments for data collection. We update the policy with the rollout data for 8 epochs after every 8 rollout steps for the MLP policies and 50 rollout steps for the RNN policies. A rollout episode is terminated (reset) if the object is reoriented to the goal orientation successfully, or the object falls, or the maximum episode length is reached. To learn the student policies $\pi^{\mathcal{S}}$, we use Dagger[19]. While Dagger typically keep all the state-action pairs for training the policy, we do Dagger in an online fashion where $\pi^{\mathcal{S}}$ only learns from the latest rollout data.

For the vision experiments, the number of parallel environments is 360 and we update policy after every 50 rollout steps from all the parallel environments. The batch size is 30. We sample 15000 points from the reconstructed point cloud of the scene from 2 cameras for the scene point cloud W_t^s and sample 5000 points from the object CAD mesh model for the goal point cloud W^g.

We use Horovod [45] for distributed training and Adam [46] optimizer for neural network optimization.

Reward function for reorientation: For training $\pi^{\mathcal{E}}$ for the reorientation task, we modified the reward function proposed in [24] to be:

$$r(s_t, a_t) = c_{\theta_1}\frac{1}{|\Delta\theta_t| + \epsilon_\theta} + c_{\theta_2}\mathbb{1}(|\Delta\theta_t| < \bar{\theta}) + c_3\left\|a_t\right\|_2^2 \tag{1}$$

where $c_{\theta_1} > 0$, $c_{\theta_2} > 0$ and $c_3 < 0$ are the coefficients, $\Delta\theta_t$ is the difference between the current object orientation and the target orientation, ϵ_θ is a constant, $\mathbb{1}$ is an indicator function that identifies whether the object is in the target orientation. The first two reward terms encourage the policy to reorient the object to the desired orientation while the last term suppresses large action commands.

Reward function for object lifting: To train the lifting policy, we use the following reward function:

$$r(s_t, a_t) = c_{h_1}\frac{1}{|\Delta h_t| + \epsilon_h} + c_{h_2}\mathbb{1}(|\Delta h_t| < \bar{h}) + c_3\left\|a_t\right\|_2^2 \tag{2}$$

where $\Delta h_t = \max(p_t^{b,z} - p_t^{o,z}, 0)$ and $p_t^{b,z}$ is the height (z coordinate) of the Shadow Hand base frame, $p_t^{o,z}$ is the height of the object, $\bar{h}$ is the threshold of the height difference. The objects have randomly initialized poses and are dropped onto the table.

Goal specification for vision policies: We obtain W^g by sampling 5000 points from the object's CAD mesh using the Barycentric coordinate, rotating the points by the desired orientation, and translating them so that these points are next to the hand. Note that one can also put the object in the desired orientation right next to the hand in the simulator and render the entire scene altogether to remove the need for CAD models. We use CAD models for W^g just to save the computational cost of rendering another object while we still use RGBD cameras to get W_t^s.

Table C.2: Hyperparameter Setup

Hyperparameter	Value	Hyperparameter	Value	Hyperparameter	Value
Num. batches	5	Entropy coeff.	0.	Num. pts sampled from W_t^s	15000
Actor learning rate	0.0003	GAE parameter	0.95	Num. pts sampled from W^g	5000
Critic learning rate	0.001	Discount factor	0.99	Num. envs	40000
Num. epochs	8	Episode length	300	Num. rollout steps per policy update (MLP/RNN)	8/50
Value loss coeff.	0.0005	PPO clip range	0.1	c_{θ_1}	1
c_{θ_2}	800	c_3	0.1	ϵ_θ	0.1
$\bar{\theta}$	0.1rad	c_{h_1}	0.05	ϵ_h	0.02
$\bar{h}$	0.04	c_{h_2}	800		

C.3 Dynamics randomization

Table C.4 list all the randomized parameters as well the state observation noise and action command noise.

Comparing the Column 1 and Column 2 in Table D.5, we can see that if we directly deploy the policy trained without domain randomization into an environment with different dynamics, the performance drops significantly. If we train policies with domain randomization (Column 3), the policies are more robust and the performance only drops slightly compared to Column 1 in most cases. The exceptions are on C3 and H3. In these two cases, the π_M^S policies collapsed in training during the policy distillation along with the randomized dynamics.

Table C.3: Mesh Parameters

Parameter	Range
longest side of the bounding box of EGAD objects	[0.05, 0.08]m
longest side of the bounding box of YCB objects	[0.05, 0.12]m
mass of each object	[0.05, 0.15]kg
No. of EGAD meshes	2282
No. of YCB meshes	78
No. of variants per mesh	5
Voxelization resolution	0.003 m

Table C.4: Dynamics Randomization and Noise

Parameter	Range	Parameter	Range	Parameter	Range
state observation	$+\mathcal{U}(-0.001, 0.001)$	action	$+\mathcal{N}(0, 0.01)$	joint stiffness	$\times\mathcal{E}(0.75, 1.5)$
object mass	$\times\mathcal{U}(0.5, 1.5)$	joint lower range	$+\mathcal{N}(0, 0.01)$	tendon damping	$\times\mathcal{E}(0.3, 3.0)$
object static friction	$\times\mathcal{U}(0.7, 1.3)$	joint upper range	$+\mathcal{N}(0, 0.01)$	tendon stiffness	$\times\mathcal{E}(0.75, 1.5)$
finger static friction	$\times\mathcal{U}(0.7, 1.3)$	joint damping	$\times\mathcal{E}(0.3, 3.0)$		

$\mathcal{N}(\mu, \sigma)$: Gaussian distribution with mean μ and standard deviation σ.
$\mathcal{U}(a, b)$: uniform distribution between a and b. $\mathcal{E}(a, b) = \exp^{\mathcal{U}(\log(a), \log(b))}$.
$+$: the sampled value is added to the original value of the variable. $\times$: the original value is scaled by the sampled value.

C.4 Gravity curriculum

We found building a curriculum on gravity helps improve the policy learning for YCB objects when the hand faces downward. Algorithm 1 illustrates the process of building the gravity curriculum. In our experiments, we only test on the training objects once (one random initial and goal orientation) to get the surrogate average success rate w on all the objects during training. $\bar{w} = 0.8, g_0 = 1\,\mathrm{m/s^2}, \Delta g = -0.5\,\mathrm{m/s^2}, K = 3, L = 20, \Delta T_{\min} = 40$.

Algorithm 1 Gravity Curriculum

```
 1: Initialize an empty FIFO queue $Q$ of size $K$, $\Delta T = 0, g = g_0$
 2: for $i \leftarrow 1$ to $M$ do
 3:     $\tau$ = rollout_policy($\pi_\theta$)                              ▷ get rollout trajectory
 4:     $\pi_\theta$ = optimize_policy($\pi_\theta$, $\tau$)                        ▷ update policy
 5:     $\Delta T = \Delta T + 1$
 6:     if $i \mod L = 0$ then
 7:         $w$ = evaluate_policy($\pi_\theta$)     ▷ evaluate the policy, get the success rate $w$
 8:         append $w$ to the queue $Q$
 9:         if avg($Q$) $> \bar{w}$ and $\Delta T > \Delta T_{\min}$ then
10:             $g = \max(g - \Delta g, -9.81)$
11:             $\Delta T = 0$
12:         end if
13:     end if
14: end for
```

Appendix D Supplementary Results

D.1 Hand faces upward

Learning curves Figure D.6 shows the learning curve of the RNN and MLP policies on the EGAD and YCB datasets. Both policies learn well on the EGAD and YCB datasets. The YCB dataset requires much more environment interactions for the policies to learn. We can also see that using the full-state information can speed up the learning and give a higher final performance.

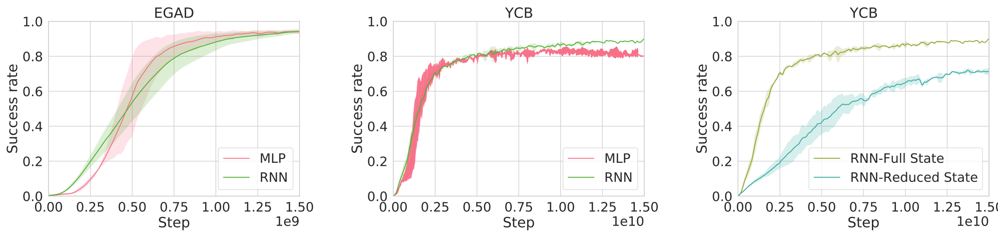


Figure D.6: Learning curves of the MLP and RNN policies on the EGAD (**Left**) and YCB datasets (**Middle**). The **Right** plot shows that using the full state information speeds up the policy learning compared to only using the reduced state information.

Testing performance - Teacher The testing results in Table D.5 show that both the MLP and RNN policies are able to achieve a success rate greater than 90% on the EGAD dataset (entries A1, B1) and greater than 70% on the YCB dataset (entries F1, G1) without any explicit knowledge of the object shape. This result is surprising because intuitively, one would assume that information about the object shape is important for in-hand reorientation.

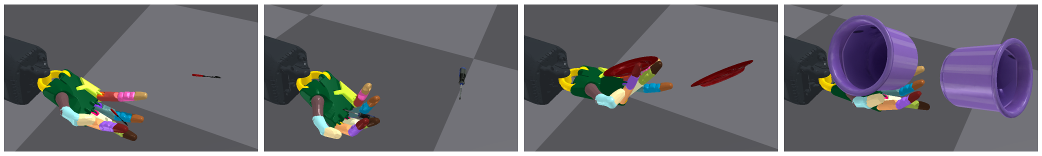
Testing performance - Student We experimented with the following three combinations: (1) distill $\pi_M^{\mathcal{E}}$ into $\pi_M^{\mathcal{S}}$, (2) distill $\pi_M^{\mathcal{E}}$ into $\pi_R^{\mathcal{S}}$, (3) distill $\pi_R^{\mathcal{E}}$ into $\pi_R^{\mathcal{S}}$. The student policy state is $s_t^{\mathcal{S}} \in \mathbb{R}^{31}$. In Table D.5 (entries C1-E1, H1-J1), we can see that $\pi_R^{\mathcal{E}} \rightarrow \pi_R^{\mathcal{S}}$ gives the highest success rate on $\pi^{\mathcal{S}}$, while $\pi_M^{\mathcal{E}} \rightarrow \pi_M^{\mathcal{S}}$ leads to much worse performance (36% drop of success rate in EGAD, and 47%

Table D.5: Success rates (%) of policies tested on different dynamics distribution. $\bar{\theta} = 0.1\text{rad}$. DR stands for domain randomization and observation/action noise. X→Y: distill policy X into policy Y.

				1	2	3
Exp. ID	Dataset	State	Policy	Train without DR		Train with DR
				Test without DR	Test with DR	Test with DR
A	EGAD	Full state	MLP	92.55 ± 1.3	78.24 ± 2.4	**91.92** ± **0.4**
B			RNN	**95.95** ± **0.8**	**84.27** ± **1.0**	88.04 ± 0.6
C		Reduced state	MLP→MLP	55.55 ± 0.2	25.09 ± 3.0	23.77 ± 1.8
D			MLP→RNN	85.32 ± 1.2	68.31 ± 2.6	**81.05** ± **1.2**
E			RNN→RNN	**91.96** ± **1.5**	**78.30** ± **1.2**	80.29 ± 0.9
F	YCB	Full state	MLP	73.40 ± 0.2	54.57 ± 0.6	66.00 ± 2.7
G			RNN	**80.40** ± **1.6**	**65.16** ± **1.0**	**72.34** ± **0.9**
H		Reduced state	MLP→MLP	34.08 ± 0.9	12.08 ± 0.4	5.41 ± 0.3
I			MLP→RNN	69.30 ± 0.8	47.38 ± 0.6	53.07 ± 0.9
J			RNN→RNN	**81.04** ± **0.5**	**64.93** ± **0.2**	**65.86** ± **0.7**

drop in YCB). This shows that π^S requires temporal information due to reduced state space. The last two columns in Table D.5 also show that the policy is more robust to dynamics variations and observation/action noise after being trained with domain randomization.

Failure cases Figure D.7 shows some example failure cases. If the objects are too small, thin, or big, the objects are likely to fall. If objects are initialized close to the hand border, it is much more difficult for the hand to catch the objects. Another failure mode is that the objects are reoriented close to the goal orientation but not close enough to satisfy $\Delta\theta \leq \bar{\theta}$.



(a) (b) (c) (d)

Figure D.7: Examples of failure cases. (a) and (b): objects are too small. (c): the object is reoriented close to the target orientation, but not close enough. (d): the object is too big and initialized around the palm border.

D.2 Hand faces downward (in the air)

Testing performance For the case of reorienting objects in the air with the hand facing downward Table D.6 lists the success rates of different policies trained with/without domain randomization, and tested with/without domain randomization.

Table D.6: Success rates (%) of policies trained with hand facing downward and to reorient objects in the air. Due to the large number of environment steps required in this setup, we fine-tune the model trained without DR with randomized dynamics instead of training models with DR from scratch.

				1	2	3
Exp. ID	Dataset	State	Policy	Train without DR		Finetune with DR
				Test without DR	Test with DR	Test with DR
K	EGAD	Full state	MLP	**84.29** ± **0.9**	**38.42** ± **1.5**	**71.44** ± **1.3**
L			RNN	82.27 ± 3.3	36.55 ± 1.4	67.44 ± 2.1
M		Reduced state	MLP→RNN	**77.05** ± **1.6**	29.22 ± 2.6	59.23 ± 2.3
N			RNN→RNN	74.10 ± 2.3	**37.01** ± **1.5**	**62.64** ± **2.9**
O	YCB	Full state	MLP	58.95 ± 2.0	26.04 ± 1.9	44.84 ± 1.3
P			RNN	52.81 ± 1.7	**26.22** ± **1.0**	40.44 ± 1.5
Q			RNN + g-curr	**74.74** ± **1.2**	25.56 ± 2.9	**54.24** ± **1.4**
R		Reduced state	MLP→RNN	46.76 ± 2.5	**25.49** ± **1.4**	34.14 ± 1.3
S			RNN→RNN	45.22 ± 2.1	24.45 ± 1.2	31.63 ± 1.6
T			RNN + g-curr→ RNN	**67.33** ± **1.9**	19.77 ± 2.8	**48.58** ± **2.3**

Example visualization We show an example of reorienting a cup in Figure D.8 and an example of reorienting a sponge in Figure D.9. More examples are available at `https://taochenshh.github.io/projects/in-hand-reorientation`.

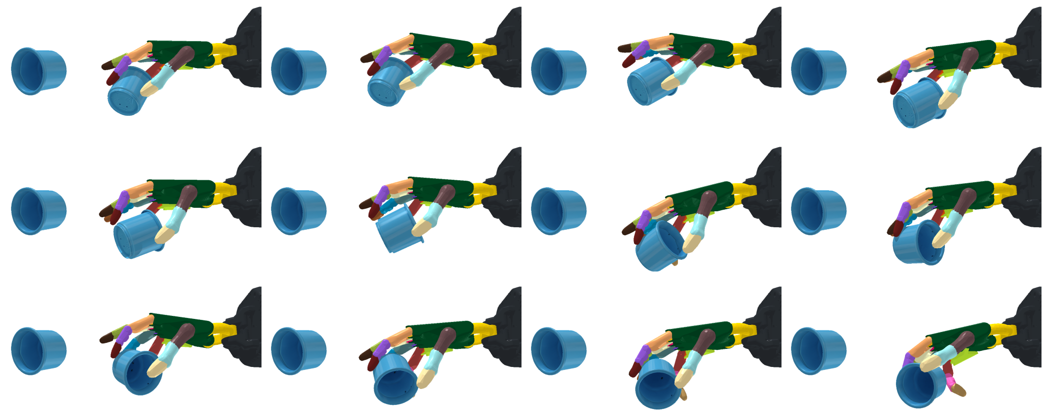


Figure D.8: An example of reorienting a cup with the hand facing downward. From left to right, top to bottom, we show the some moments in an episode.

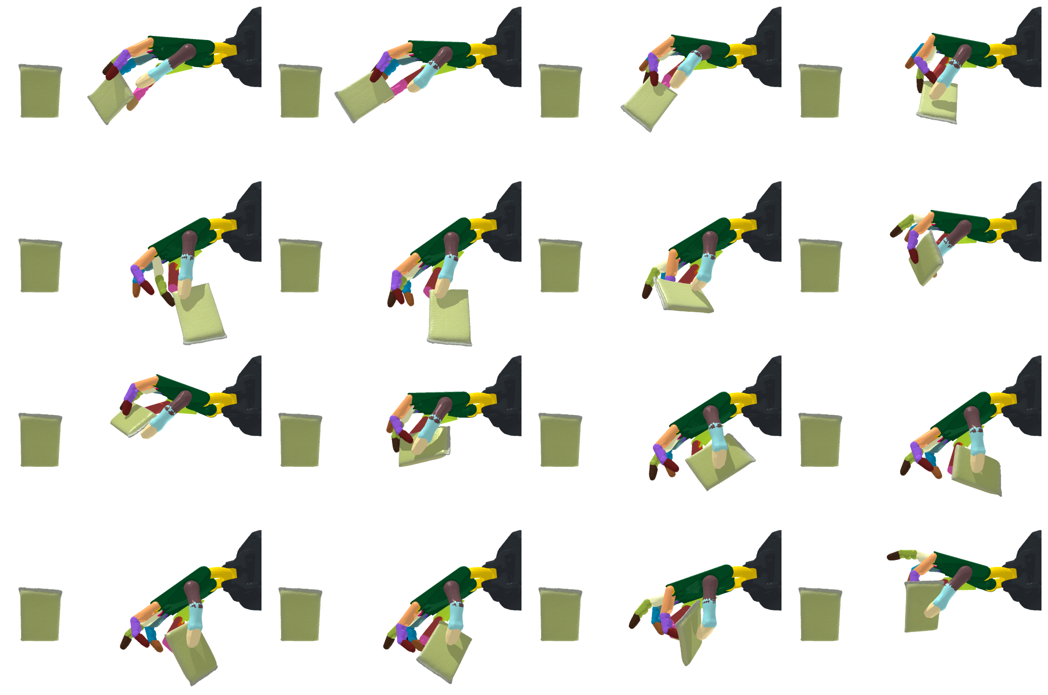


Figure D.9: An example of reorienting a sponge with the hand facing downward. From left to right, top to bottom, we show the some moments in an episode.

D.3 Success rate on each type of YCB objects

We also analyzed the success rates on each object type in the YCB dataset. Using the same evaluation procedure described in Section 3, we get the success rates of each object using $\pi_R^{\mathcal{E}}$. Figure D.10 shows the distribution of the success rates on YCB objects with the hand facing upward while Figure D.11 corresponds to the case of reorienting the objects in the air with the hand facing downward. We can see that sphere-like objects such as tennis balls and orange are easiest to reorient. Long or thin objects such as knives and forks are the hardest ones to manipulate.

D.4 Torque analysis

We randomly sampled 100 objects from the YCB dataset, and use our RNN policy trained without domain randomization with the hand facing downward to reorient each of these objects 200 times. We record the joint torque values for each finger joint at each time step. Let the joint torque value of i^{th} joint at time step j in k^{th} episode be J_{ij}^k. We plot the distribution of $\{\max_{i=[\![1,24]\!]} |J_{ij}^k| \mid j \in [\![1,T]\!], k = [\![1,20000]\!]\}$, where $[\![a,b]\!]$ represents $\{x \mid x \in [a,b], x \in \mathbb{Z}\}$. Figure D.12 shows that the majority of the maximum torque magnitude is around $0.2\,\mathrm{N\,m}$.

D.5 Vision experiments with noise

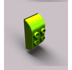
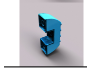
We also trained our vision policies with noise added to the point cloud input. We added the following transformations to the point cloud input.

We applied four types of transformations on the point cloud:

- *RandomTrans*: Translate the point cloud by $[\Delta x, \Delta y, \Delta z]$ where $\Delta x, \Delta y, \Delta z$ are all uniformly sampled from $[-0.04, 0.04]$.
- *JitterPoints*: Randomly sample 10% of the points. For each sampled point i, jitter its coordinate by $[\Delta x_i, \Delta y_i, \Delta z_i]$ where $\Delta x_i, \Delta y_i, \Delta z_i$ are all sampled from a Normal distribution $\mathcal{N}(0, 0.01)$ (truncated at -0.015m and 0.015m).
- *RandomDropout*: Randomly dropout points with a dropout ratio uniformly sampled from $[0, 0.4]$.
- *JitterColor*: Jitter the color of points with the following 3 transformations: (1) jitter the brightness and rgb values, (2) convert the color of 30% of the points into gray, (3) jitter the color contrast. Each of this transformation can be applied independently with a probability of 30% if *JitterColor* is applied.

Each of these four transformations is applied independently with a probability of 40% for each point cloud at every time step. Table D.7 shows the success rates of the vision policies trained with the aforementioned data augmentations until policy convergence and tested with the same data augmentations. We found that adding the data augmentation in training actually helps improve the data efficiency of the vision policy learning even though the final performance might be a bit lower. As a reference, we show the policy performance trained and tested without data augmentation in Table D.7. For the mug object, adding data augmentation in training improves the final testing performance significantly. Without data augmentation, the learned policy reorients the mug to a pose where the body of the mug matches how the mug should look in the goal orientation, but the cup handle does not match. Adding the data augmentation helps the policy to get out of this local optimum.

Table D.7: Vision policy success rates when the policy is trained and tested with/without data augmentation ($\bar{\theta} = 0.2\,\mathrm{rad}, \bar{d}_C = 0.01$)

	Object	Without data augmentation (noise)	With data augmentation (noise)
	025_mug	36.51 ± 2.8	89.67 ± 1.2
	065-d_cups	79.17 ± 2.3	68.32 ± 1.9
	072-b_toy_airplane	90.25 ± 3.7	84.52 ± 1.4
	073-a_lego_duplo	62.16 ± 3.7	58.16 ± 3.1
	073-c_lego_duplo	58.21 ± 3.9	50.21 ± 3.7
	073-e_lego_duplo	76.57 ± 3.6	66.57 ± 3.1

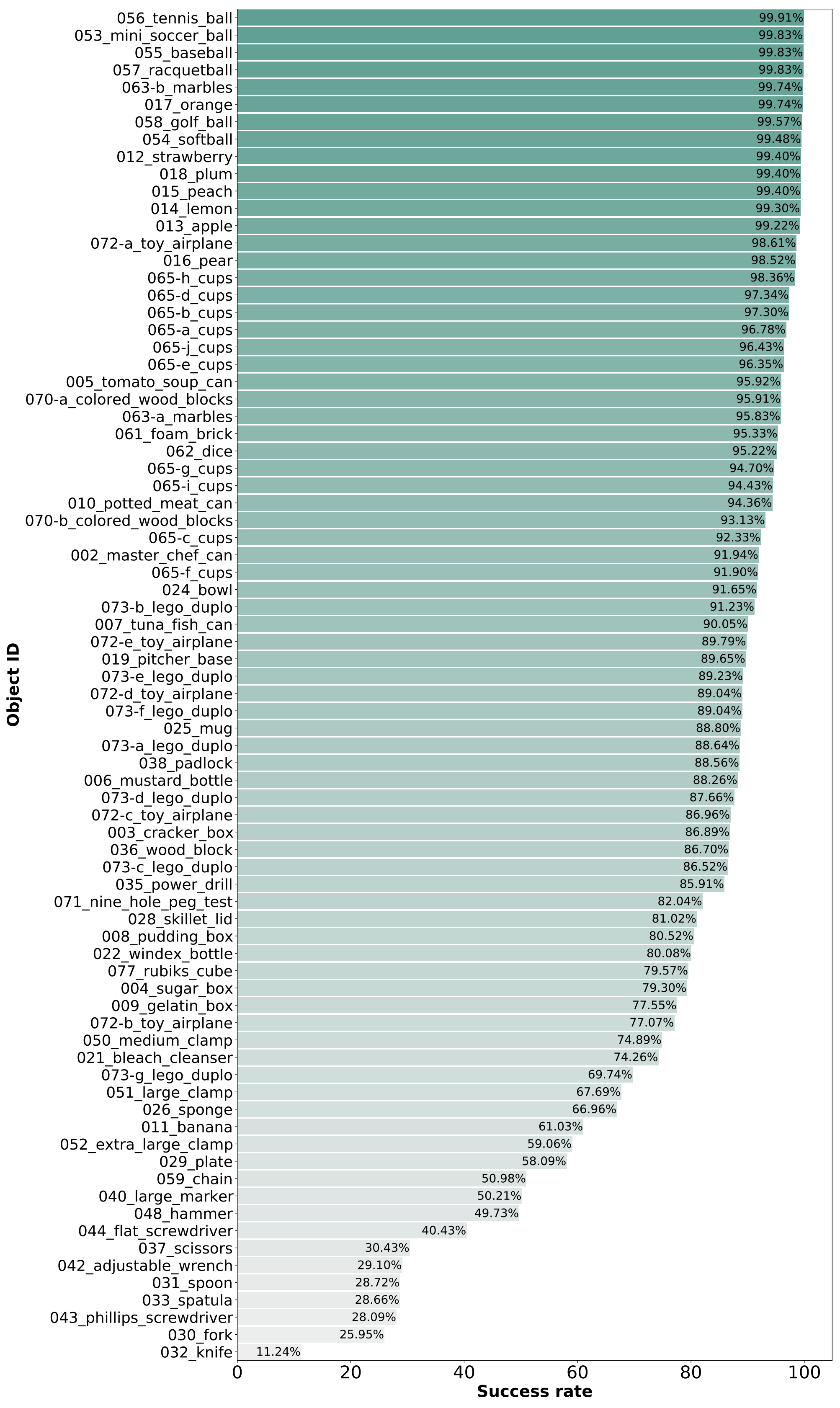


Figure D.10: Reorientation success rates for each object in the YCB dataset when the hand faces upward.

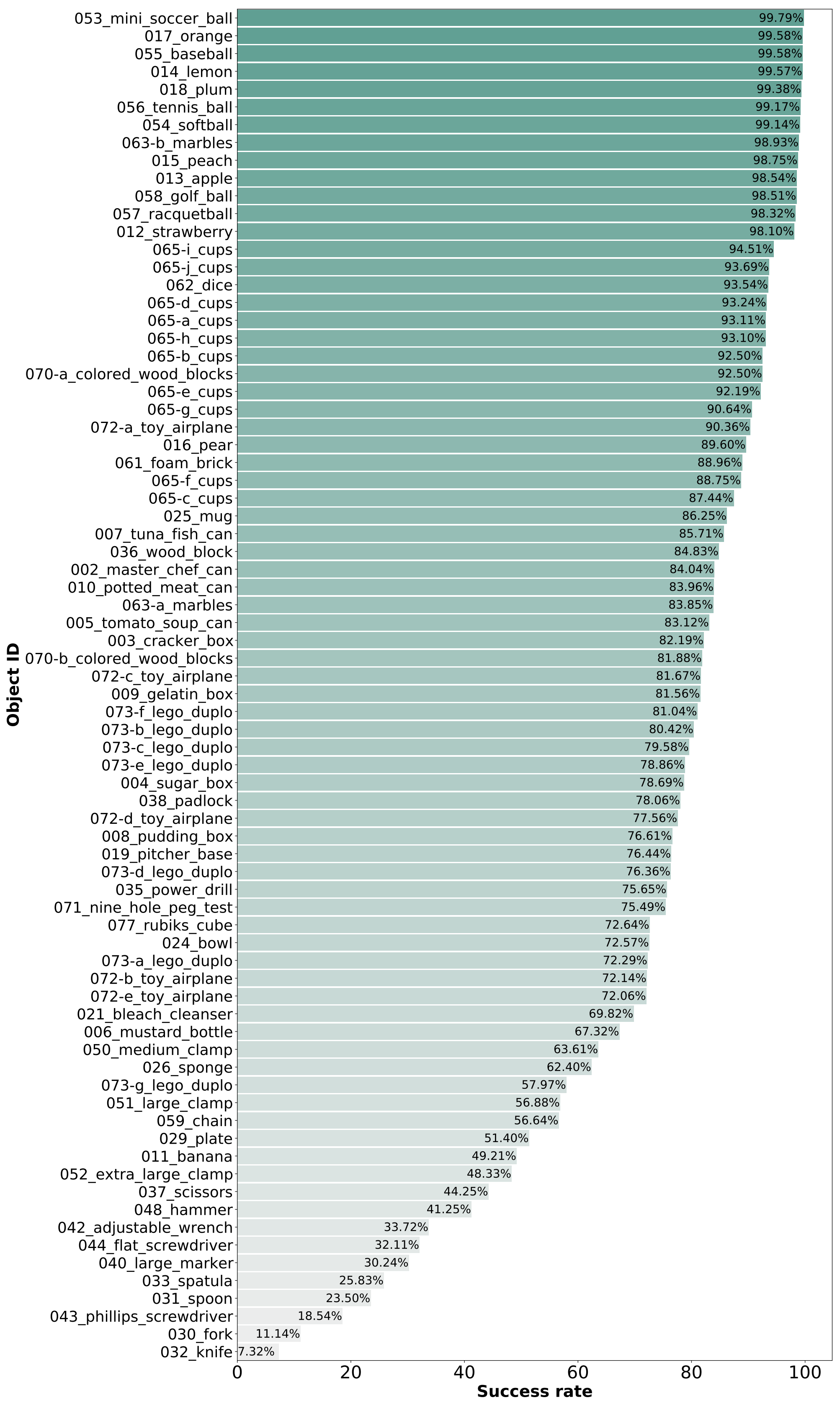


Figure D.11: Reorientation success rates for each object in the YCB dataset when the hand faces downward without a table.

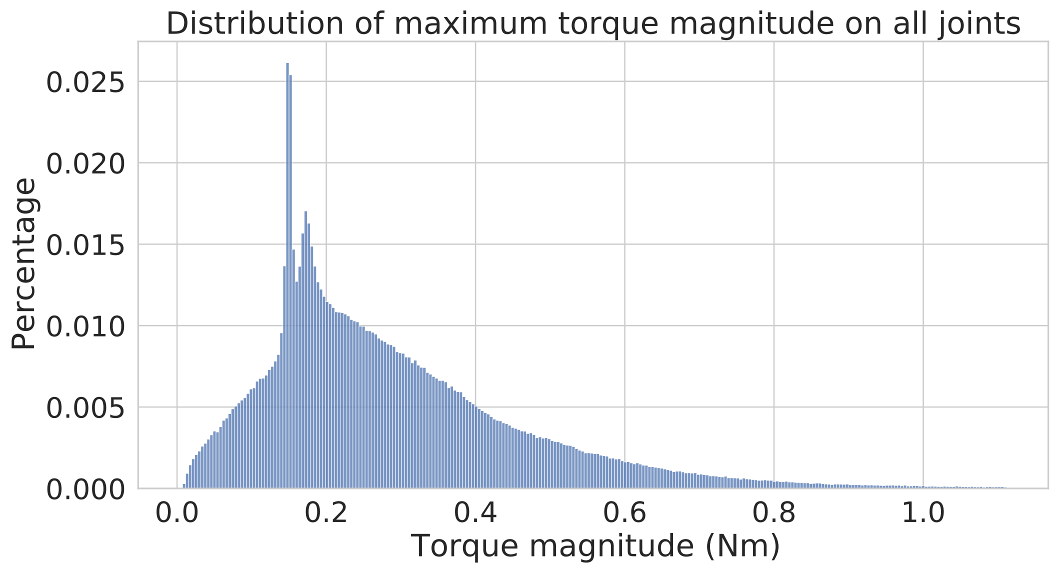


Figure D.12: Distribution of the *maximum* absolute joint torque values on all joints for all the time steps. We exclude a few outliers in the plot, i.e., we only plot the data distributions up to 99% quantile.