A Simple Method for Complex In-Hand Manipulation

Anonymous Author(s)
Affiliation
Address
email

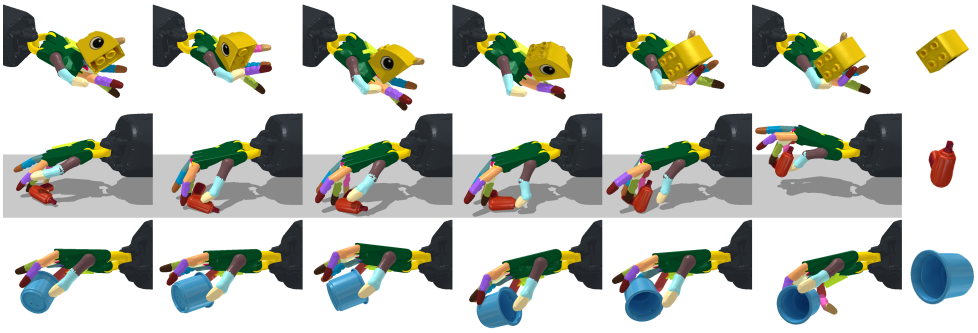


Figure 1: We present a simple framework for learning policies for reorienting a large number of objects in scenarios where the (1) hand faces upward, (2) hand faces downward with a table below the hand and (3) without the support of the table. The object orientation in the rightmost image in each row shows the goal orientation.

Abstract: In-hand object reorientation has been a challenging problem in robotics due to high dimensional actuation space and the frequent change in contact state between the fingers and the objects. We present a simple model-free framework that can learn to reorient objects with both the hand facing upwards and downwards. We demonstrate the capability of reorienting over 2000 geometrically different objects in both cases. The learned policies show strong zero-shot transfer performance on new objects. We provide evidence that these policies are amenable to real-world operation by distilling them to use observations easily available in the real world. The videos of the learned policies are available at: https://sites.google.com/view/in-hand-reorientation.

Keywords: Dexterous manipulation, in-hand manipulation, object reorientation

1 Introduction

A common maneuver in many tasks of daily living is to pick an object, reorient it in hand and then either place it or use it as a tool. Consider three simplified variants of this maneuver in Figure 1. The first task shown in the top row requires an upward-facing multi-finger hand to manipulate an arbitrary object in a random orientation to a goal configuration. The next two rows show the second and third tasks where the hand is facing downward and is required to reorient the object either using the table as a support or without the aid of any support surface respectively. These two tasks are harder because the object is in an intrinsically unstable configuration owing to the downward gravitational force and lack of support from the palm. Additional challenges in making such maneuvers with a multi-finger robotic hand stem from the control space being high-dimensional and reasoning about multiple transitions in the contact state between the finger and the object. Due to its practical utility and several unsolved issues, in-hand object reorientation remains an active area of research.

Past work has tackled the in-hand reorientation problem via several approaches: (i) The use of analytical models with powerful trajectory optimization methods [1, 2, 3]. While these methods demonstrated remarkable performance, the results were largely in simulation with simple object geometries and required detailed knowledge of the object model and physical parameters. As such, it remains unclear how to scale these methods to real-world and generalize to new objects. Another line of work has employed (ii) model-based reinforcement learning [4, 5]; or (iii) model-free reinforcement

Submitted to the 5th Conference on Robot Learning (CoRL 2021). Do not distribute.

learning with [6, 7, 8, 9] and without expert demonstrations [10, 11, 12, 13]. While some of these works demonstrated learned skills on the real robots, they required substantial engineering and additional apparatus (e.g., motion capture system) to infer the object state, and the learned policies did not generalize to diverse objects. Furthermore, nearly all of these methods operate in a simplified setting where the hand is only allowed to face upward (with pick-and-place being an exception, but pick-and-place doesn't incorporate any in-hand manipulation). A detailed discussion of prior research is provided in Section 5.

In this paper, our goal is to study the object reorientation problem with a multi-finger hand in its general form. We desire (a) manipulation with hand facing upward or downward; (b) the skill of using external surfaces to aid manipulation; (c) the ability to reorient objects of novel shapes to arbitrary orientations; (d) operation from sensory data that can be easily obtained in the real world such as RGBD images and joint positions of the hand. While some of these aspects have been individually demonstrated in prior work, we are unaware of any published method that realizes all four. Our main contribution is building a simple framework that can achieve these desiderata. We experimentally test this framework with a simulated Shadow hand. Due to the scope of the problem and the ongoing pandemic, we limit our experiments to be in simulation. However, we provide evidence indicating that the learned policies can be transferred to the real world in the future.

2 Method

To our surprise, simple extensions to existing techniques in robot learning can be used to construct a system for general object reorientation. First, to avoid explicit modeling of the highly non-linear and frequent changes in the contact state between the object and the hand, we use model-free reinforcement learning (RL). An added advantage is that model-free RL is amenable to directly operating from raw RGBD point cloud observations. Second, we make use of *privileged* state information such as the positions/orientations/velocities of the object/fingertips that is easily available in the simulator but not in the real world. Such state information can simplify policy learning and increase data efficiency. We train an RL policy with privileged state information to reorient more than two thousand objects of diverse shapes (see Section 2.1). However such a policy cannot be transferred to the real robot due to the difficulty of getting the priviledged information in real world. We address this issue via teacher-student learning.

The idea of first learning an expert or *teacher* policy ($\pi^{\mathcal{E}}$) from privileged information and then using it to guide the learning of a *student* policy ($\pi^{\mathcal{S}}$) that only uses sensory inputs available in the real world is known as *teacher-student training* [14, 15]. We leverage this idea to demonstrate the possibility of transferring the learned policies to the real world in the future. Let the state space corresponding to $\pi^{\mathcal{E}}$ and $\pi^{\mathcal{S}}$ be $\mathbb{S}^{\mathcal{E}}$ and $\mathbb{S}^{\mathcal{S}}$ respectively. In general, $\mathbb{S}^{\mathcal{E}} \neq \mathbb{S}^{\mathcal{S}}$. We first detail the method for distilling $\pi^{\mathcal{E}}$ to a student policy using a reduced state space consisting of only the joint positions of the hand, the object position, and the difference in orientation from the goal configuration (see Section 2.2.1). However, in the real world, the object position and relative orientation must be inferred from sensory observation. Not only does this process require substantial engineering effort (e.g., a motion capture or a pose estimation system), but also inferring the pose of a symmetric object is prone to errors. This is because a symmetric object at multiple orientations looks exactly the same in sensory space such as RGBD images. To mitigate these issues, we further distill $\pi^{\mathcal{E}}$ to operate directly from the RGBD point cloud and position of all the hand joints (see Section 2.2.2). We propose a simple modification that generalizes an existing 2D CNN to make this possible.

While the procedure described above works well for manipulation with the hand facing upwards and downwards with a table as a support, we find that in the case of the hand facing downward without a table, a good initialization of object pose such as the one obtained by an object lifting policy is key for successful reorientation, and the proposed gravity curriculum substantially improves performance.

2.1 Learning the teacher policy

We use model-free RL to learn the teacher policy ($\pi^{\mathcal{E}}$) for reorienting an object ($\{O_i | i = 1, ..., N\}$) from an initial orientation α_0^o to a target orientation α^g ($\alpha_0^o \neq \alpha^g$). At every time step t, the agent observes the state s_t, executes the action a_t sampled from the policy $\pi^{\mathcal{E}}$, and receives a reward r_t. $\pi^{\mathcal{E}}$ is optimized to maximize the expected discounted return: $\pi = \arg\max_{\pi^{\mathcal{E}}} \mathbb{E}\big[\sum_{t=0}^{T-1} \gamma^t r_t\big]$, where γ is the discount factor. The task is successful if the angle difference $\Delta\theta$ between the object's current (α_t^o) and the goal orientation (α^g) is smaller than the threshold value $\bar{\theta}$, *i.e.*, $\Delta\theta \leq \bar{\theta}$.

To encourage the policy to be smooth, the previous action is appended to the inputs to the policy (*i.e.*, $a_t = \pi^{\mathcal{E}}(s_t, a_{t-1})$) and large actions are penalized in the reward function. We experiment with two architectures for $\pi^{\mathcal{E}}$: (1) an MLP policy π_M, (2) an RNN policy π_R. We use PPO [16] to optimize $\pi^{\mathcal{E}}$. All the experiments in the paper were run on at most 2 GPUs with a 32GB memory. More details about the network architecture and training are in Section C.1 and Section C.2 in the appendix.

Observation and action space: We define $\mathbb{S}^{\mathcal{E}}$ to include joint, fingertip, object, and goal information as detailed in Table B.1 in the appendix. Note that $\mathbb{S}^{\mathcal{E}}$ does not include object shape or information about friction, damping, contact states between the fingers and the object, etc. We control the joint movements by commanding the relative change in the target joint angle (q_t^{target}) on each actuated joint (action $a_t \in \mathbb{R}^{20}$): $q_{t+1}^{target} = q_t^{target} + a_t \times \Delta t$, where Δt is the control time step. We clamp the action command if necessary to make sure $|\Delta q_t^{target}| \leq 0.33\,\text{rad}$. The control frequency is $60\,\text{Hz}$.

Reward function: For training $\pi^{\mathcal{E}}$, the reward function is defined as:

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

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 is a regularizer to suppress large action commands. In our experiments, we use $c_{\theta_1} = 1, c_{\theta_2} = 800, c_3 = 0.1, \epsilon_\theta = 0.1, \bar{\theta} = 0.1\text{rad}$.

Dynamics randomization: Even though we do not test our policies on a real robot, we train and evaluate policies with domain randomization [17] to provide evidence that our work has the potential to be transferred to a real robotics system in the future. We randomize the object mass, friction coefficient, joint damping and add noise to the state observation s_t and the action a_t. More details about domain randomization are provided in Table C.2 in the appendix.

2.2 Learning the student policy

We distill the teacher $\pi^{\mathcal{E}}$ into the student policy $\pi^{\mathcal{S}}$ using Dagger [18], a learning-from-demonstration method that overcomes the covariate shift problem. We optimize $\pi^{\mathcal{S}}$ by minimizing the KL-divergence between $\pi^{\mathcal{S}}$ and $\pi^{\mathcal{E}}$: $\pi^{\mathcal{S}} = \arg\min_{\pi^{\mathcal{S}}} D_{KL}\left(\pi^{\mathcal{E}}(s_t^{\mathcal{E}}, a_{t-1}) || \pi^{\mathcal{S}}(s_t^{\mathcal{S}}, a_{t-1})\right)$. Based on observation data available in real-world settings, we investigate two different choices of $\mathbb{S}^{\mathcal{S}}$.

2.2.1 Training student policy from low-dimensional state

In the first case, we consider a non-vision student policy $\pi^{\mathcal{S}}(s_t^{\mathcal{S}}, a_{t-1})$. $s_t^{\mathcal{S}} \in \mathbb{R}^{31}$ includes the joint positions $q_t \in \mathbb{R}^{24}$, object position $p_t^o \in \mathbb{R}^3$, quaternion difference between the object's current and target orientation $\beta_t \in \mathbb{R}^4$. In this case, $\mathbb{S}^{\mathcal{S}} \subset \mathbb{S}^{\mathcal{E}}$, and we assume the availability of object pose information, but do not require velocity information. We use the same MLP and RNN network architectures used for $\pi^{\mathcal{E}}$ on $\pi^{\mathcal{S}}$ except the input dimension changes as the state dimension is different.

2.2.2 Training student policy from vision

In the second case, $\mathbb{S}^{\mathcal{S}}$ only uses direct observations from RGBD cameras and the joint position (q_t) of the robotic hand. We convert the RGB and Depth data into a colored point cloud using a pinhole camera model [19]. Our vision policy takes as input the voxelized point cloud of the scene W_t, q_t, and previous action command a_{t-1}, and outputs the action a_t, *i.e.*, $a_t = \pi^{\mathcal{S}}(W_t, q_t, a_{t-1})$.

Goal specification: To avoid the dependency on any manually defined coordinate frame for specifying the goal quaternion, we only provide the object's point cloud rotated to the desired orientation W^g to the policy, *i.e.*, we only show the policy how the object should look like in the end (see the top left of Figure C.5). The input to $\pi^{\mathcal{S}}$ is the point cloud $W_t = W_t^s \cup W^g$. 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.

Sparse3D-IMPALA-Net: To convert a voxelized point cloud into a lower-dimensional feature representation, we use a sparse convolutional neural network. We extend the IMPALA policy architecture [20] for processing RGB images to process colored point cloud data using 3D convolution. Since many voxels are unoccupied, the use of regular 3D convolution wastes substantial computation time. Hence, we use Minkowski Engine [21], a PyTorch library for sparse tensors, to design a 3D version of IMPALA-Net with sparse convolutions (*Sparse3D-IMPALA-Net*)[1]. The point cloud W_t is processed by a series of 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. The detailed architecture is illustrated in Figure C.5 in the appendix.

Mitigating the object symmetry issue: $\pi^{\mathcal{E}}$ is trained with the the ground-truth state information $s_t^{\mathcal{E}}$ including the object orientation α_t^o and goal orientation α^g. The vision policy does not take any orientation information as input. If an object is symmetric, the two different orientations of the object may correspond to the same point cloud observation. This makes it problematic to use the difference in orientation angles ($\Delta\theta \leq \bar{\theta}$) as the stopping and success criterion. To mitigate this issue, we use Chamfer distance [22] to compute the distance between the object point cloud in α_t^o and the goal point cloud (i.e., the object rotated by α^g) as the evaluation criterion. The Chamfer distance is computed as $d_C = \sum_{a \in W_t^o} \min_{b \in W^g} \|a - b\|_2^2 + \sum_{b \in W^g} \min_{a \in W_t^o} \|a - b\|_2^2$, where W_t^o is the object point cloud in its current orientation. Both W_t^o and W^g are scaled to fit in a unit sphere for computing d_C. We check Chamfer distance in each rollout step. If $d_C \leq \bar{d}_C$ ($\bar{d}_C$ is a threshold value for d_C), we consider the episode to be successful. Hence, the success criterion is $(\Delta\theta \leq \bar{\theta}) \vee (d_C \leq \bar{d}_C)$. In training, if the success criterion is satisfied, the episode is terminated and used for updating $\pi^{\mathcal{S}}$.

3 Experimental Setup

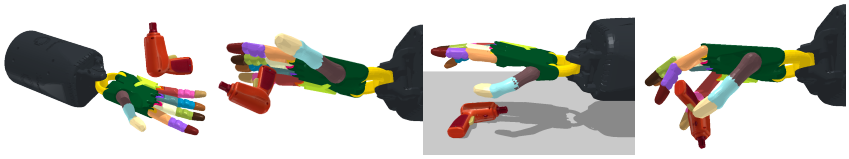
We use the simulated Shadow Hand [23] in NVIDIA Isaac Gym [24]. Shadow Hand is an anthropomorphic robotic hand with 24 degrees of freedom (DoF). In our experiments, we assume the base of the hand to be fixed. Twenty joints are actuated by agonist–antagonist tendons and the remaining four joints are under-actuated.

Object datasets: We use the EGAD dataset [25] and YCB dataset [26] that contain objects with diverse shapes (see Figure B.2) for in-hand manipulation experiments. EGAD contains 2282 geometrically diverse textureless object meshes, while the YCB dataset includes textured meshes for objects of daily life with different shapes and textures. We use the 78 YCB object models collected with the Google scanner. Since most YCB objects are too big for in-hand manipulation, we proportionally scale down the YCB meshes. 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. Mass is randomly sampled from $[0.05, 0.15]$kg for every object. When we randomly scale YCB objects, some objects become too small or too thin, making the reorientation task even more challenging. In total, we use 11410 EGAD object meshes and 390 YCB object meshes for training. More details on the dataset are available in Section B.2 in the appendix.

Setup for visual observations: For the vision experiments, we experiment with the scenario where the hand faces upward. We place two RGBD cameras above the hand (Figure B.4). The data from these cameras is combined to create a point cloud observation of the scene [19]. For downstream computation, the point cloud is voxelized with a resolution of $0.003\,\mathrm{m}$.

Setup with the upward facing hand: We first consider the case where the Shadow Hand faces upward and is required to reorient objects placed in the hand (see Figure 2a). We use the coordinate system where the z-axis points vertically upward and the xy-plane denotes the horizontal plane. The object is initialized as following: xy position of the object's center of mass (COM) $p_{0,xy}^o$ is randomly sampled from a square region of size $0.09\,\mathrm{m} \times 0.09\,\mathrm{m}$. The center of this square is approximately located on the intersection of the middle finger and the palm so that the sampling region covers both the fingers and the palm. The z position of the object is fixed to $0.13\,\mathrm{m}$ above the base of the hand to ensure that the object does not collide with the hand at initialization. The initial and goal orientations are randomly sampled from the full $SO(3)$ space.

[1] We also experimented with a 3D sparse convolutional network based on ResNet18, and found that 3D IMPALA-Net works better.



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

Figure 2: Examples of initial poses of the hand and object. **(a)**: hand faces upward. **(b)**, **(c)**, **(d)**: hand faces downward. **(b)**: both the hand and the object are initialized with random poses . **c**: there is a table below the hand. **(d)**: the hand and the object are initialized from the lifted poses.

Setup with the downward facing hand: Next, we consider the cases where the hand faces downward. We experiment with two scenarios: with and without a table below the hand. In the first case, we place a table with the tabletop being $0.12\,\mathrm{m}$ below the hand base. We place objects in a random pose between the hand and the table so that the objects will fall onto the table. We will describe the setup for the second case (without a table) in Section 4.2.2.

Evaluation criterion: For non-vision experiments, a policy rollout is considered a success if $\theta \leq \bar{\theta}$. $\bar{\theta} = 0.1\,\mathrm{rad}$. For vision experiments, we also check $d_C \leq \bar{d}_C$ as another criterion and $\bar{\theta} = 0.2\,\mathrm{rad}, \bar{d}_C = 0.01$. The initial and goal orientation are randomly sampled from $SO(3)$ space in all the experiments. We report performance as the percentage of successful episodes when the agent is tasked to reorients all the training objects 100 times from arbitrary start to goal orientation. We report the mean and standard deviation of success rate from 3 seeds.

4 Results

4.1 Reorient objects with the hand facing upward

Train a teacher policy with full-state information We train our teacher MLP and RNN policies using the full state information on the EGAD and YCB datasets separately. The success rate as a function of environment steps during training is shown in Figure D.6 in the appendix. Figure D.6 also shows that using the full-state information can speed up policy learning significantly. The testing results in Table 1 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.

The visualization of the policy rollout reveals that the agent is able to reorient different objects without knowing their shapes by finding a clever strategy that is invariant to object geometry. The agent throws the object in the air with a spin and catches it at the precise time when the object's orientation matches the goal orientation. Throwing the object with a spin is a dexterous spin that automatically emerges! One possibility for the emergence of this skill is that we used very light objects. This is not true because we trained with objects in the range of 50-150g which spans many hand-held objects used by humans (e.g., an egg weighs about 50g, a glass cup weighs around 100g, iPhone 12 weighs 162g, etc.). To further probe this concern, we evaluated zero-shot performance on objects weighing up to 500g[2] and found that the learned policy can successfully reorient them. We provide further analysis in the appendix showing that forces applied by the hand for such manipulation are realistic. While there is still room for the possibility that the learned policy is exploiting the simulator to reorient objects by throwing them in the air, our analysis so far indicates otherwise.

Next, to understand the failure modes, we collected one hundred failed trajectories on YCB dataset and manually documented the reason for failure. It turns out that the primary failure is with long, small, or thin objects, which accounts for 60% of the failures. In such cases, the objects slip through the fingers and fall down, or get stuck on the palm. Results in Figure D.9 confirm 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. Another cause of failures (19%) is that objects are reoriented close to the goal orientation but not close enough to satisfy $\Delta\theta < \bar{\theta}$. Finally, we note that the performance on YCB is lower than EGAD because objects in the YCB dataset have greater diversity in the aspect ratios. Scaling these objects by constraining $l_{\max} \in [0.05, 0.12]$m (see Section 3) makes some of the objects either too small, too big, or too thin and consequently results in failure (see Figure D.11).

[2]We change the mass of the YCB objects to be in the range of $[0.3, 0.5]$kg, and test $\pi_R^{\mathcal{E}}$ from the YCB dataset on these new objects. The success rate is around 75%.

Table 1: 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

Train a student policy with a reduced state space 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 1 (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% drop in YCB). This shows that $\pi^{\mathcal{S}}$ requires temporal information due to reduced state space. The last two columns in Table 1 also show that the policy is more robust to dynamics variations and observation/action noise after being trained with domain randomization.

4.2 Reorient objects with the hand facing downward

The results above demonstrate that when the hand faces upwards, RL can be used to train policies for reorienting a diverse set of geometrically different objects. A natural question to ask is then does this still hold true when the hand is flipped upside down? Intuitively, this task is much more challenging because the objects will fall immediately without proper control on the fingers, which means the exploration becomes much harder for the RL agent. To see if this is the case, we experimented with the following settings: We first flip the hand upside down, placed the objects underneath the hand (see Figure 2b), and use the same reward function (Equation (1)) to train the reorientation policy. Unsurprisingly, the agent is unable to learn and results in 0% success. This is because the policy needs to learn to both stabilize the object to prevent it from falling under the effect of gravity and simultaneously reorient it. We found that the application of the method developed until now leads to object falling down, showing that this is a hard-exploration problem.

4.2.1 Reorient objects on a table

To tackle the problem of reorienting objects with the hand facing downward, we start with a simpler version of the task where there is a table below the hand (see Figure 2c). The presence of the table mitigates the exploration issue because it prevents the objects from falling. We train $\pi_M^{\mathcal{E}}$ using the same reward function Equation (1) on objects sampled from the EGAD and YCB datasets. The success rate for the EGAD dataset with an MLP policy using full state information is $95.3\% \pm 0.9\%$. The success rate for the YCB dataset is $81.59\% \pm 0.7\%$. Using external support for in-hand manipulation has been a challenging problem in robotics. Recent works approach this problem by building analytical models and constructing motion cones [27]. However, building the analytical models is especially difficult for objects with complex geometry. Our experiments show that model-free RL is able to learn to use the support effectively to reorient many objects with the hand facing downward.

4.2.2 Reorient objects in air with hand facing downwards

While providing a table can help RL learn a good reorientation policy, we want the skill to be more general. We would like the policy to reorient objects with the hand facing downwards and without any external support. In this setup, one might hypothesize that object shape information (e.g., from vision) is critical because finding the strategy in Section 4.1 is not easy when the hand needs to overcome gravity and stabilize the object while reorienting it. We experimentally verify that even in this case, the policies achieve a reasonably high success rate without any knowledge of object shape.

A good pose initialization is what you need: The difficulty of directly training the RL policies when the hand faces downward is mainly because of the hard-exploration issue in learning to catch the

Table 2: Zero-shot RNN policy transfer success rates (%) across datasets. **U.** (**D.**) means hand faces upward (downward). **FS** (**RS**) means using full-state (reduced-state) information.

	EGAD $\rightarrow$ YCB	YCB $\rightarrow$ EGAD
U.FS	68.82 ± 1.7	96.41 ± 1.2
U.RS	59.64 ± 1.8	96.38 ± 1.3
D.FS	62.73 ± 2.2	85.45 ± 2.9
D.RS	55.30 ± 1.3	77.91 ± 2.1

Table 3: Vision policy success rate ($\bar{\theta} = 0.2\,\text{rad}, \bar{d}_C = 0.01$)

Object	Success rate (%)
025_mug	34.64 ± 2.9
065-d_cups	79.17 ± 2.3
072-b_toy_airplane	90.25 ± 3.7
073-a_lego_duplo	62.16 ± 3.7
073-c_lego_duplo	58.21 ± 3.9
073-e_lego_duplo	76.57 ± 3.6

objects that are moving downward. However, catching is not necessary for the reorientation. Even for human, we only reorient the object after we grasp it. More specifically, we first train an object-lifting policy to lift objects from the table, collect the ending state (joint positions q_T, object position p_T^o and orientation α_T^o) in each successful lifting rollout episode, and reset the hand and objects to these states (velocities are all 0) for the pose initialization in training the reorientation policy. To train the lifting policy, we use the following reward function: $r(s_t, a_t) = c_{h_1}\frac{1}{|\Delta h_t|+\epsilon_h} + c_{h_2}\mathbb{1}(|\Delta h_t| < \bar{h}) + c_3\left\|a_t\right\|_2^2$ 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. In our experiments, $c_{h_1} = 0.05, \epsilon_h = 0.02, \bar{h} = 0.04, c_{h_2} = 800, c_3 = 0.1$. The objects have randomly initialized poses and are dropped onto the table. After we train the lifting policy, we collect about 250 ending states for each object respectively from the successful lifting episodes. In every reset during the reorientation policy training, ending states are randomly sampled and used as the initial pose of the fingers and objects. With a good pose initialization, policies are able to learn to reorient objects with high success rates. $\pi_R^{\mathcal{E}}$ trained on EGAD dataset gets a success rate more than 80% while $\pi_R^{\mathcal{E}}$ trained on YCB dataset gets a success rate greater than 50%. More results on the different policies with and without domain randomization are available in Table D.3 in the appendix. This setup is challenging because if at any time step in an episode the fingers take a bad action, the object will fall.

Improving performance using gravity curriculum: Since the difficulty of training the reorientation policy with the hand facing downward is due to the gravity, we propose to build a gravity curriculum to learn the policy $\pi^{\mathcal{E}}$. Since $\pi^{\mathcal{E}}$ already performs very well on EGAD objects, we apply gravity curriculum to train $\pi^{\mathcal{E}}$ on YCB objects. Our gravity curriculum is constructed as follows: we start the training with $g = 1\,\text{m/s}^2$, then we gradually decrease g in a step-wise fashion if the evaluation success rate (w) is above a threshold value ($\bar{w}$) until $g = -9.81\,\text{m/s}^2$. More details about gravity curriculum are in Section C.4. In Table D.3 (Exp Q and T) in the appendix, we can see that adding gravity curriculum (g-curr) significantly boost the success rates on the YCB dataset.

4.3 Zero-shot policy transfer across datasets

We have shown the testing performance on the same training dataset so far. How would the policies work on a different dataset? To answer this, we test our policies across datasets: policies trained with EGAD objects are now tested with YCB objects and vice versa. We used the RNN policies trained with full-state information and reduced-state information respectively (without domain randomization) and tested them on the other dataset with the hand facing upward and downward. In the case of the hand facing downward, we tested the RNN policy trained with gravity curriculum. Table 2 shows that policies still perform well on the untrained dataset.

4.4 Reorient objects with RGBD sensors

In this section, we investigate whether we can train a vision policy to reorient objects with the hand facing upward. As vision-based experiments require much more compute resources, we train one vision policy for each object individually on six objects shown in Table 3. We leave training a single vision policy for all objects to future work. We use the expert MLP policy trained in Section 4.1 to supervise the vision policy. We can see from Table 3 that reorienting the toy has a high success rate while reorienting the mug has a much lower success rate. For the mug, the policy reorients the mug to a pose where the body of the mug match how the mug should look like in the goal orientation, but the cup handle does not match. This is likely due to the difficulty of the network capturing the small mug handle structure from the entire point cloud. Using d_C as an episode termination criterion, we

can train a vision policy for the symmetric objects (cup and legos) individually with high success rates as shown in Table 3.

5 Related Work

Dexterous manipulation has been studied for decades, dating back to [28, 29]. In contrast to parallel-jaw grasping, pushing, or pick-and-place, dexterous manipulation typically involves continuously controlling force to the object through the fingertips of a robotic hand [30]. Some prior works used analytical kinematics and dynamics models of the hand and object, and used trajectory optimization to output control policies [1, 2] or employed kinodynamic planning to find a feasible motion plan [31]. However, due to the large number of active contacts on the hand and the objects, model simplifications such as simple finger and object geometries are usually necessary to make the optimization or planning tractable. Furukawa et al. [32] achieved a high-speed dynamic regrasping motion on a cylinder using a high-speed robotic hand and a high-speed vision system. Recent works have explored the application of reinforcement learning to dexterous manipulation. Model-based RL works learned a linear [4, 7] or deep neural network [5] dynamics model from the rollout data, and used online optimal control to rotate a pen or Baoding balls on a Shadow hand. However, when the system is unstable, collecting informative trajectories for training a good dynamics model that generalizes to different objects remains challenging. Another line of works uses model-free RL algorithms to learn a dexterous manipulation policy. For example, Andrychowicz et al. [11] and Akkaya et al. [12] learned a controller to reorient a block or a Rubik's cube. To reduce the sample complexity of model-free learning, [9, 13, 8, 33, 6] combined imitation learning with RL to learn to reorient a pen, open a door, assemble LEGO blocks, etc. However, collecting expert demonstration data from humans is expensive, time-consuming, and even incredibly difficult for contact-rick tasks [8]. Our method belongs to the category of model-free learning. We use the teacher-student learning paradigm to speed up the deployment policy learning. Our learned policies also generalize to new shapes and show strong zero-shot transfer performance. To the best of our knowledge, our system is the first work that demonstrates the capabilities of reorienting a diverse set of objects that have complex geometries with both the hand facing upward and downward.

6 Discussion

Our results show that model-free RL with simple deep learning architectures can be used to train policies to reorient a large set of geometrically diverse objects. Further for learning with hand facing downwards, we found that a good pose initialization obtained from a lifting policy was necessary, and the gravity curriculum substantially improved performance. The agent also learns to use an external surface (i.e., the table). The most surprising observation is that information about shape is not required despite the fact that we train a single policy to manipulate multiple objects. Perhaps in hindsight, it is not as surprising – after all, humans can close their eyes and easily manipulate novel objects into a specific orientation. However, even in this scenario, shape can be implicitly inferred using touch signals. As such, our work can serve a strong baseline for future in-hand object reorientation works that incorporate object shape information as part of the observation space to improve performance.

It should be noted that successful training of the teacher policy critically relied on shape information being expendable but required privileged information – a major obstacle in the real-world deployment. While we were not able to train reorientation policies from scratch using visual inputs, we show that we can distill this policy to operate from RGBD point clouds and joint positions that are readily available in the real world. While we only present results in simulation, we also provide evidence that our policies can be transferred to the real world. The experiments with domain randomization indicate that learned policies can work with noisy inputs. Analysis of peak torques during manipulation (see Figure D.12 in the appendix) also reveals that generated torque commands are feasible to actuate on an actual robotic hand.

Finally, Figure D.9 and Figure D.10 in the appendix show that the success rate varies substantially with object shape. This suggests that in the future, a training curriculum based on object shapes can improve performance. Another future work is to directly train one vision policy for a diverse set of objects. A major bottleneck in vision-based experiments is the demand for much larger GPU memory. Learning visual representations of point cloud data that can ease the computational bottleneck is, therefore, an important avenue for future research.

References

[1] I. Mordatch, Z. Popović, and E. Todorov. Contact-invariant optimization for hand manipulation. In *Proceedings of the ACM SIGGRAPH/Eurographics symposium on computer animation*, pages 137–144, 2012.

[2] Y. Bai and C. K. Liu. Dexterous manipulation using both palm and fingers. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1560–1565. IEEE, 2014.

[3] V. Kumar, Y. Tassa, T. Erez, and E. Todorov. Real-time behaviour synthesis for dynamic hand-manipulation. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6808–6815. IEEE, 2014.

[4] V. Kumar, E. Todorov, and S. Levine. Optimal control with learned local models: Application to dexterous manipulation. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 378–383. IEEE, 2016.

[5] A. Nagabandi, K. Konolige, S. Levine, and V. Kumar. Deep dynamics models for learning dexterous manipulation. In *Conference on Robot Learning*, pages 1101–1112. PMLR, 2020.

[6] A. Gupta, C. Eppner, S. Levine, and P. Abbeel. Learning dexterous manipulation for a soft robotic hand from human demonstrations. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3786–3793. IEEE, 2016.

[7] V. Kumar, A. Gupta, E. Todorov, and S. Levine. Learning dexterous manipulation policies from experience and imitation. *arXiv preprint arXiv:1611.05095*, 2016.

[8] A. Rajeswaran, V. Kumar, A. Gupta, G. Vezzani, J. Schulman, E. Todorov, and S. Levine. Learning complex dexterous manipulation with deep reinforcement learning and demonstrations. *arXiv preprint arXiv:1709.10087*, 2017.

[9] I. Radosavovic, X. Wang, L. Pinto, and J. Malik. State-only imitation learning for dexterous manipulation. *arXiv preprint arXiv:2004.04650*, 2020.

[10] M. Mudigonda, P. Agrawal, M. Deweese, and J. Malik. Investigating deep reinforcement learning for grasping objects with an anthropomorphic hand. 2018.

[11] O. M. Andrychowicz, B. Baker, M. Chociej, R. Jozefowicz, B. McGrew, J. Pachocki, A. Petron, M. Plappert, G. Powell, A. Ray, et al. Learning dexterous in-hand manipulation. *The International Journal of Robotics Research*, 39(1):3–20, 2020.

[12] I. Akkaya, M. Andrychowicz, M. Chociej, M. Litwin, B. McGrew, A. Petron, A. Paino, M. Plappert, G. Powell, R. Ribas, et al. Solving rubik's cube with a robot hand. *arXiv preprint arXiv:1910.07113*, 2019.

[13] H. Zhu, A. Gupta, A. Rajeswaran, S. Levine, and V. Kumar. Dexterous manipulation with deep reinforcement learning: Efficient, general, and low-cost. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 3651–3657. IEEE, 2019.

[14] J. Lee, J. Hwangbo, L. Wellhausen, V. Koltun, and M. Hutter. Learning quadrupedal locomotion over challenging terrain. *Science robotics*, 5(47), 2020.

[15] D. Chen, B. Zhou, V. Koltun, and P. Krähenbühl. Learning by cheating. In *Conference on Robot Learning*, pages 66–75. PMLR, 2020.

[16] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[17] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. In *2017 IEEE/RSJ international conference on intelligent robots and systems (IROS)*, pages 23–30. IEEE, 2017.

[18] S. Ross, G. Gordon, and D. Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 627–635. JMLR Workshop and Conference Proceedings, 2011.

[19] A. M. Andrew. Multiple view geometry in computer vision. *Kybernetes*, 2001.

[20] L. Espeholt, H. Soyer, R. Munos, K. Simonyan, V. Mnih, T. Ward, Y. Doron, V. Firoiu, T. Harley, I. Dunning, et al. Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures. In *International Conference on Machine Learning*, pages 1407–1416. PMLR, 2018.

[21] C. Choy, J. Gwak, and S. Savarese. 4d spatio-temporal convnets: Minkowski convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3075–3084, 2019.

[22] H. Fan, H. Su, and L. J. Guibas. A point set generation network for 3d object reconstruction from a single image. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 605–613, 2017.

[23] ShadowRobot. ShadowRobot Dexterous Hand, 2005. URL https://www.shadowrobot.com/dexterous-hand-series/.

[24] J. Liang, V. Makoviychuk, A. Handa, N. Chentanez, M. Macklin, and D. Fox. Gpu-accelerated robotic simulation for distributed reinforcement learning. In *Conference on Robot Learning*, pages 270–282. PMLR, 2018.

[25] D. Morrison, P. Corke, and J. Leitner. Egad! an evolved grasping analysis dataset for diversity and reproducibility in robotic manipulation. *IEEE Robotics and Automation Letters*, 5(3): 4368–4375, 2020.

[26] B. Calli, A. Singh, A. Walsman, S. Srinivasa, P. Abbeel, and A. M. Dollar. The ycb object and model set: Towards common benchmarks for manipulation research. In *2015 international conference on advanced robotics (ICAR)*, pages 510–517. IEEE, 2015.

[27] N. Chavan-Dafle, R. Holladay, and A. Rodriguez. In-hand manipulation via motion cones. *arXiv preprint arXiv:1810.00219*, 2018.

[28] J. K. Salisbury and J. J. Craig. Articulated hands: Force control and kinematic issues. *The International journal of Robotics research*, 1(1):4–17, 1982.

[29] M. T. Mason, J. K. Salisbury, and J. K. Parker. Robot hands and the mechanics of manipulation. 1989.

[30] N. C. Dafle, A. Rodriguez, R. Paolini, B. Tang, S. S. Srinivasa, M. Erdmann, M. T. Mason, I. Lundberg, H. Staab, and T. Fuhlbrigge. Extrinsic dexterity: In-hand manipulation with external forces. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1578–1585. IEEE, 2014.

[31] D. Rus. In-hand dexterous manipulation of piecewise-smooth 3-d objects. *The International Journal of Robotics Research*, 18(4):355–381, 1999.

[32] N. Furukawa, A. Namiki, S. Taku, and M. Ishikawa. Dynamic regrasping using a high-speed multifingered hand and a high-speed vision system. In *Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006.*, pages 181–187. IEEE, 2006.

[33] R. Jeong, J. T. Springenberg, J. Kay, D. Zheng, Y. Zhou, A. Galashov, N. Heess, and F. Nori. Learning dexterous manipulation from suboptimal experts. *arXiv preprint arXiv:2010.08587*, 2020.

[34] K. Mamou, E. Lengyel, and A. Peters. Volumetric hierarchical approximate convex decomposition. In *Game Engine Gems 3*, pages 141–158. AK Peters, 2016.

[35] D.-A. Clevert, T. Unterthiner, and S. Hochreiter. Fast and accurate deep network learning by exponential linear units (elus). *arXiv preprint arXiv:1511.07289*, 2015.

[36] A. Sergeev and M. D. Balso. Horovod: fast and easy distributed deep learning in TensorFlow. *arXiv preprint arXiv:1802.05799*, 2018.

[37] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.