From Simple to Complex Skills: The Case of In-Hand Object Reorientation

Haozhi Qi[1,2], Brent Yi[1], Mike Lambeta[2], Yi Ma[1], Roberto Calandra[3,4] and Jitendra Malik[1,2]

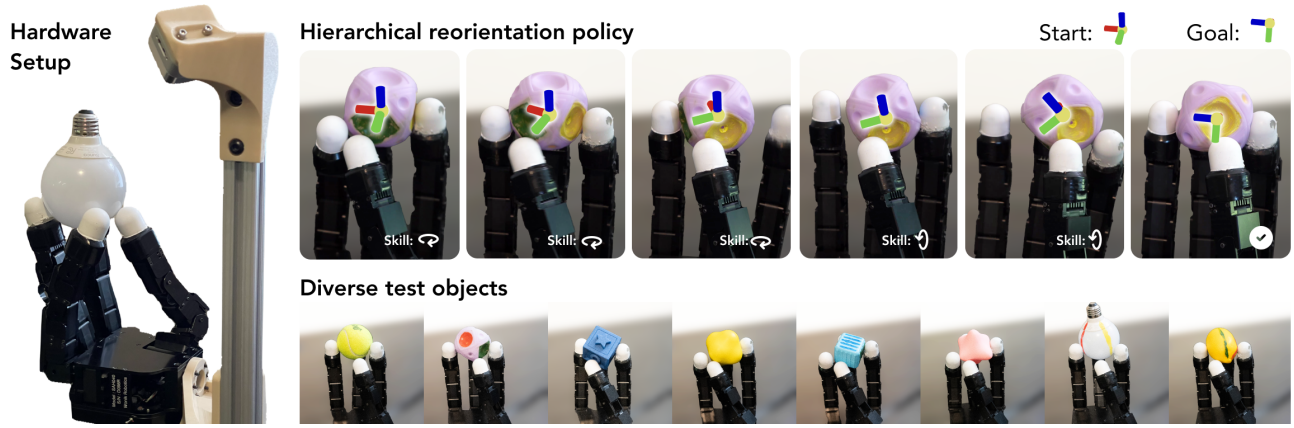


Fig. 1: **Left:** Hardware Setup. We use a multi-fingered robot hand with an RGB-D camera for our system. **Right:** We learn a hierarchical policy for in-hand object reorientation by reusing pre-trained skills (object rotation along single axes). It can manipulate diverse objects with symmetries and with different physical properties.

***Abstract*— Learning policies in simulation and transferring them to the real world has become a promising approach in dexterous manipulation. However, bridging the sim-to-real gap for each new task requires a substantial amount of human effort, such as careful reward engineering, hyperparameter tuning, and system identification. In this work, we present a system that demonstrates how low-level skills can be leveraged to mitigate these challenges for more complex tasks. Specifically, we present a hierarchical policy for in-hand object reorientation based on previously acquired rotation skills. This hierarchical policy learns to select which low-level skill should be executed based on feedback from both the environment as well as from the low-level skill policies themselves. Compared to learning from scratch, the hierarchical policy is robust to out-of-distribution changes and transfers easily from simulated to real-world environments. In addition, we propose a generalizable object pose estimator that takes proprioceptive information, low-level skill predictions, and control errors as inputs, and estimates object pose over time. We show that our system can reorient objects, including symmetrical and texture-less ones, to a desired pose.**

I. Introduction

Dexterous in-hand manipulation has made significant progress in recent years [1], [2], [3]. One promising approach is to train a policy in simulation using reinforcement learning and then transfer it to the real world [4], [5], [6]. These policies are empirically generalizable and robust due to the diverse data available in simulation. However, they are usually trained from scratch for each task, and the reward function and its coefficients must be carefully tuned for each new task. This requires a substantial amount of human effort and is difficult to scale up.

In contrast, humans acquire new skills by building upon existing ones [7]. Consider a beginner in tennis attempting their first serve: tossing a ball into the air, swinging their racket, and directing the serve. Each of these sub-skills is not practiced specifically within the context of tennis, but the individual can draw from past experiences with other balls or rackets. Although the initial execution might be clumsy, it can become smoother with practice.

Motivated by how humans acquire new skills, we argue that, when learning robot behavior for new tasks, we should leverage existing pre-trained skills. In machine learning, the idea of using pre-trained models has led to significant progress in computer vision [8], [9] and natural language processing [10], yet their application in manipulation skill acquisition remains limited. We present a hierarchical policy for in-hand object reorientation using pre-trained object rotation skills, and show its effectiveness compared to training from scratch. We chose this task because in-hand reorientation is an important skill in our daily lives and is also representative of the complexity of dexterous manipulation.

The idea of exploiting hierarchies in robotics has also been widely studied in both task and motion planning (TAMP) [11] and reinforcement learning [12]. However, it often suffers because the low-level skill cannot provide enough feedback to the high-level policy, making it brittle if (inevitably) errors occur in the execution of the low-level skill. We tackle this problem by providing the high-level planner with feedback from the low-level skill and outputting a residual correction term to complement the low-level skills.

Specifically, we use the in-hand object rotation policies [3], [6] as pre-trained skills and build a planner policy on top. The planner policy outputs two commands: 1) a rotation axis to guide the low-level rotation skills, and 2) a residual action to complement the low-level actions. Such a design

offers several advantages. First, it utilizes the structured low-level skills and reduces the exploration space, making training significantly more efficient and effective. Second, the low-level policy can predict an object representation and provide feedback to the high-level policy. This design enables the high-level policy to be aware of low-level skill reactions and to correct errors by outputting residual actions. Additionally, since the low-level skills are transferable to the real world, the human effort required to bridge the gap is significantly reduced.

Another challenge in building an in-hand object reorientation system is the need for a generalizable object state estimator. Pose estimation has been extensively studied in computer vision [13], [14], but it suffers when encountering out-of-distribution objects and heavy occlusions, as in in-hand manipulation tasks. As a result, previous works bypass this problem by building state estimators using either object keypoints [5] or point clouds [4]. However, object keypoints need to be manually designed for specific objects (e.g., a single cube in [5]), and using point clouds for goal specification cannot handle symmetrical objects. For example, even simple spheres become challenging because point clouds look the same from all rotation angles.

To address these issues, we propose a generalizable state estimator that takes a series of proprioceptive inputs and low-level skill feedback and outputs the relative rotations over a certain time interval. The state estimator is inspired by [15], but it cannot generalize, so the authors train a separate policy for each different object. We improve upon this by making two key changes: 1) using the modular design of a hierarchical policy instead of jointly learning both the controller and the state estimator, and 2) providing feedback from the low-level policies to the state estimator, enabling it to work with multiple objects. The estimator is trained entirely in simulation but is robust enough for use with novel objects in the real world.

We evaluate our system through comprehensive experiments. First, we demonstrate that our policy converges faster and achieves better performance compared to learning from scratch. Second, we show that we can learn a generalizable state estimator using the data generated in simulation and transfer it to the real world. We further study different design choices of the system through ablation experiments and verify the importance of residual actions and low-level skill feedback. Finally, we show that our learned policy naturally gains the benefits of a transferable low-level policy and can be easily transferred to the real world.

II. Related Work

In-Hand Manipulation. In-hand manipulation has been studied for decades [16], [17], [18], [19], [20], [21], [22], [23], [24], [25] in classic robotics. More recently, learning-based methods have achieved significant progress [1], [3], [4], [26] by learning policies in simulation and transferring them to the real world (sim-to-real). These methods do not require an accurate dynamics model and can more easily leverage diverse available data. However, one of the major challenges that limits these methods from scaling up is the necessity for human effort to bridge the sim-to-real gap. In practice, this usually requires a significant amount of work for reward engineering, hyperparameter tuning, and domain randomization. Our work falls into the sim-to-real category but differs from others because we build a hierarchical policy for the in-hand manipulation system by reusing previously acquired skill policies instead of starting from scratch.

Pose Estimation in Reorientation. One key component of an in-hand object reorientation system is the pose estimation method. It has been extensively studied in both computer vision [27] and robotics [13], [28]. However, pose estimation for in-hand manipulation is still far from being solved due to large occlusions and the requirements for efficiency and generalizability. As a result, previous work usually simplifies this problem either by assuming known object shapes [23], [29] or by choosing to eschew general pose estimation. For example, Dextreme [5] focuses only on a single cube and can thus estimate the pose using manually defined keypoints. It cannot generalize to different objects. Visual Dexterity [4] uses point clouds as a proxy for goal specification. However, this limits the flexibility of goal specification (it cannot reorient a simple sphere because the point clouds look the same from all rotation angles). Recently, proprioceptive feedback has also been used to estimate pose [15]. However, these methods train one policy to fit a single object and cannot generalize to different objects. Our work distinguishes itself by using generalizable low-level skills and utilizing feedback from low-level controllers, resulting in a generalizable pose estimator for multiple objects.

Skill Hierarchy in Robotics. The idea of hierarchy has a long history in classical robotics [30], [11] and hierarchical reinforcement learning [31], [32], [33]. The low-level controller can either be fine-tuned together with the high-level control or kept completely frozen [34], [35], [36], [37]. In the context of dexterous manipulation, Bhatt et al. [38] use manually designed action primitives to achieve open-loop dexterous manipulation. Khandate et al. [39] learn a switch between classic controllers and learning-based policies. Morgan et al.[23] also decompose reorientation into several rotation sequences but do not use a learning-based method. In learning-based skill reuse, Gupta et al. [40] study reset-free reinforcement learning using multiple skills, but the task transitions are defined by the user. Sequential Dexterity [29] learns a transition feasibility function for sub-skill selection. In contrast, our approach does not require learning the transition explicitly; instead, the transition is implicitly encoded in the hierarchical policy.

III. Learning In-Hand Object Reorientation with Hierarchical Skills

Our system overview is shown in Figure 2 (A). It consists of two policies: the planner policy π^{plan} and the skill policy π^{skill}. The planner policy π^{plan} takes an object's state, robot proprioception, feedback from low-level policies, and a goal orientation as input and outputs a rotation axis command a_t^{plan} and a residual action a_t^{res}. This command is sent to the

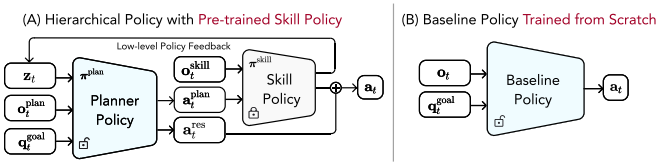


Fig. 2: **Comparison between our hierarchical policy and baseline policy.** Our planner policy takes goal orientation $\boldsymbol{q}_t^{\text{goal}}$, observation $\boldsymbol{o}_t^{\text{plan}}$, and feedback from low-level policies $\boldsymbol{z}_t$ as input and produces a one-hot skill vector $\boldsymbol{a}_t^{\text{skill}}$ along with a residual action $\boldsymbol{a}_t^{\text{res}}$, which is then used to control a low-level in-hand rotation skill.

low-level policy $\boldsymbol{\pi}^{\text{skill}}$, which outputs the raw joint position targets $\boldsymbol{a}_t$ to the robot. To estimate the object's state in the real world, we additionally train a recursive estimator using feedback from sensory and low-level policies.

A. Preliminary

Skill Policy. Our skill policy is based on the in-hand object rotation policies in [6]. We select it because it demonstrates generalization in manipulating a diverse set of objects. We reimplement the framework and learn an axis-condition in-hand object rotation policy for a given axis $\boldsymbol{k}$.

Formally, the skill policy is defined as $\boldsymbol{a}_t^{\text{skill}}, \boldsymbol{z}_t = \boldsymbol{\pi}^{\text{skill}}(\boldsymbol{o}_t^{\text{skill}}, \boldsymbol{k}_t)$ where $\boldsymbol{o}_t = [\boldsymbol{\theta}_{t-T:t}, \boldsymbol{a}_{t-T-1:t-1}^{\text{skill}}, \boldsymbol{d}_{t-T:t}]$. Among the observations, $\boldsymbol{\theta}_t \in \mathbb{R}^{16}$ represents the robot's joint positions, $\boldsymbol{a}_t^{\text{skill}} \in \mathbb{R}^{16}$ represents the commanded joint targets, and $\boldsymbol{d}_t \in \mathbb{R}^{32}$ represents the embedding of the depth image output by a lightweight ConvNet. We use $T = 30$ in our experiments. The temporal sequence $\boldsymbol{o}_t$ is fed into a transformer and outputs a single vector as the representation. The skill policy also outputs $\boldsymbol{z}_t$, which is the estimation of the object's physical properties and shapes, represented by a feature vector. We use this vector to represent the feedback from the low-level policy.

Object State. We define the object state space as $\boldsymbol{s}_t = [\boldsymbol{p}_t, \boldsymbol{q}_t]$, where $\boldsymbol{p}_t \in \mathbb{R}^3$ denotes the object's 3D position, and $\boldsymbol{q}_t \in \mathbf{S}^3$ denotes the object's orientation, represented as a unit quaternion. We define the relative pose as $\Delta(\boldsymbol{q}_{t_1}, \boldsymbol{q}_{t_2}) = \boldsymbol{q}_{t_2} \cdot \bar{\boldsymbol{q}}_{t_1}$, where $\bar{\boldsymbol{q}}$ denotes the conjugate of $\boldsymbol{q}$.

B. Learning a Hierarchical Policy

Observation and Action. Our planner policy takes in object states, robot proprioception, feedback from low-level policies, and a desired orientation, and outputs the desired rotation axis $\boldsymbol{a}_t^{\text{plan}}$. In order to adapt to variance in object dynamics, we include a short horizon of paired robot state and control actions. Formally, we have $\boldsymbol{o}_t^{\text{plan}} = [\boldsymbol{s}_{t-5:t}, \boldsymbol{\zeta}_{t-5:t}, \boldsymbol{a}_{t-6:t-1}^{\text{plan}}]$ where $\boldsymbol{a}_t^{\text{plan}}$ is the planner action in the previous timestep and $\boldsymbol{\zeta}_t = \Delta(\boldsymbol{q}_t, \boldsymbol{q}_t^{\text{goal}}) \in \mathbb{R}^4$ represents the relative transformation between object and goal orientation at timestep t. In addition, we augment the policy observation with the feedback from the low-level policy $\boldsymbol{z}_t$. Formally, we have $\boldsymbol{a}_t^{\text{plan}} = \boldsymbol{\pi}^{\text{plan}}(\boldsymbol{o}_t^{\text{plan}}, \boldsymbol{q}_t^{\text{goal}}, \boldsymbol{z}_t)$. Note that although the inputs to our policy do not contain object's shape or physical property information, it is implicitly encoded in the low-level policy feedback $\boldsymbol{z}_t$.

In practice, we concatenate the inputs as a vector and pass it through the policy network. The policy network is a simple 3-layer MLP with ELU activation [41]. The network outputs a 7-dimensional categorical distribution, from which we sample a 7-dimensional one-hot action vector denoted as $\boldsymbol{a}_t^{\text{plan}}$. The dimensions correspond to one of the six canonical rotation axes ($\pm x$, $\pm y$, $\pm z$) and an extra `STOP` command. When inputting quaternions to the network, we convert to 6D representations [42].

Residual Actions. Using a good set of low-level skills could accelerate exploration for the new task. However, they can not adapt to the new tasks since they are frozen during training. We augment the planner to output an additional residual action to complement output of skill policy. This design enables additional error correction from the planner policy. In summary, we have $[\boldsymbol{a}_t^{\text{plan}}; \boldsymbol{a}_t^{\text{res}}] = \boldsymbol{\pi}^{\text{plan}}(\boldsymbol{o}_t^{\text{plan}}, \boldsymbol{q}_t^{\text{goal}}, \boldsymbol{z}_t)$ and $\boldsymbol{a}_t = \boldsymbol{a}_t^{\text{res}} + \boldsymbol{a}_t^{\text{skill}}$ will be sent to the robot.

Reward and Policy Optimization. Our reward function is simple (t omitted for simplicity) thanks for the robustness of hierarchical policy training: $r = 1/(d(\boldsymbol{q}_t, \boldsymbol{q}_t^{\text{goal}}) + \epsilon) + \lambda_s \mathbf{1}(\text{Success})$, where λ_s are the coefficients for the reward terms. $1/(d(\boldsymbol{q}_t, \boldsymbol{q}_t^{\text{goal}}) + \epsilon)$ is the rotational distance reward [1], [4], [5]. $\mathbf{1}(\text{Success})$ is the success bonus to encourage the planner to finish the task. Without this bonus, the policy learns to approach the goal but not finish it, aiming to maximize the product of rotation reward and episode length. Compared to previous works [1], [4], [5], our reward function contains only two terms and is much easier to tune. This simplification is possible because the low-level skills have already been tuned to be transferable to the real world. Our central claim is that, when building policies for new tasks, we can avoid the tedious reward and hyperparameter tuning associated with training from scratch.

In our design, the planner policy is learned using the ground-truth object states $\boldsymbol{q}_t$ provided by the simulator. We intentionally separate the perception and control components because this modularized design allows us to benefit from advancements in both reusable low-level skills and generalized perception models in the future.

C. Generalizable State Estimator

The policy described above takes noisy object state information from the simulator as input. To transfer it to the real world, we need a robust pose estimator for our system. Pose estimation has been extensively studied in computer vision. However, generalized pose estimation in uncontrolled environments remains an unsolved problem. Additionally, the in-hand perception problem presents unique challenges. We propose a generalizable state estimator that takes a sequence of proprioceptive inputs and low-level skill feedback and outputs the relative rotations over a certain time interval. During deployment, the estimated relative transformations will be integrated to determine if the desired pose is achieved.

Recursive State Estimator. Our state estimator is a neural network ϕ that takes proprioception, action, control errors, low-level skill feedback, and previously estimated object state sequences as input and outputs the object state at the next timestep. We define the object's pose at the first frame as the canonical frame $\boldsymbol{q}_0$.

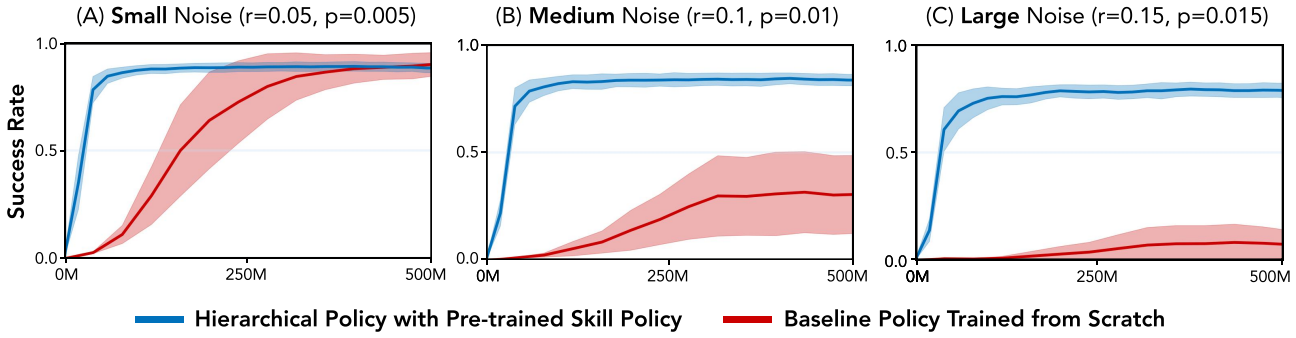


Fig. 3: **Policy training for different levels of object state noises.** We plot the training progress as the success rate with respect to agent steps. The shaded region covers one standard deviation from the mean success rate. In the easy case (A), both our method and the baseline achieve similar performance, but our method converges 8× faster than the baseline. When we gradually add input noise, the baseline method becomes unstable and has high variance across random seeds. In the large noise case (C), the baseline method does not converge, while our method still achieves decent performance and low variance across seeds.

State Estimation with Generalizable Low-level Skills. What makes our approach different from previous methods is the usage of generalizable low-level skills and the feedback provided from these skills. Previous work requires training one policy for each object because it needs to simultaneously learn manipulation skills and object pose estimation [15]. In contrast, since our low-level skills are generalizable, the pose estimator addresses an easier task, allowing our policy to manipulate multiple objects. The skill policy not only makes the reorientation of diverse objects feasible but also provides its own estimate $\boldsymbol{z}_t$ about the object's properties and shapes. For example, as shown in [6], it contains information about the object's geometry. This information is critical for achieving generalized object pose estimation.

We implement the state estimator using a transformer. We concatenate a temporal history of proprioception, action, control errors, and the predicted extrinsics as the input $\boldsymbol{f}_t = [\boldsymbol{q}_t, \boldsymbol{a}_{t-1}, \boldsymbol{q}_t - \boldsymbol{a}_{t-1}, \hat{\boldsymbol{s}}_{t-1}, \boldsymbol{z}_t]$. Then we feed a sequence of features $\boldsymbol{f}_T = \{\boldsymbol{f}_{t-k}, \ldots, \boldsymbol{f}_{t-1}, \boldsymbol{f}_t\}$ as input to the transformer. The transformer outputs $\hat{\boldsymbol{s}}_t$, which is used as the input to our policy $\boldsymbol{\pi}^{\text{plan}}$.

To train this predictor, we rollout the planner policy $\boldsymbol{\pi}^{\text{plan}}$ with the *predicted object state* in simulation. Meanwhile, we also save the ground-truth object state and construct a training set $\mathcal{B} = \{(\boldsymbol{f}_T^{(i)}, \boldsymbol{s}_t^{(i)}, \hat{\boldsymbol{s}}_t^{(i)})\}_{i=1}^N$. Then we optimize ϕ by minimizing the ℓ_2 distance between $\boldsymbol{s}_t$ and $\hat{\boldsymbol{s}}_t$ using Adam [43]. Following [15], we reset the episode when the rotational distance between the predicted quaternion and the ground-truth is larger than 0.8 radians, or the predicted object location is off by 3 cm. This is crucial for estimator training, and we find the network does not converge without it.

IV. Experiments

This section is organized as follows: We first introduce the experiment setup in Section IV-A. Then, we study the advantages of building a hierarchical policy by reusing in-hand object rotation skills compared to learning from scratch. We use the object state from the simulator as input and empirically analyze the training performance and robustness to out-of-distribution scenarios in Section IV-B. Next, we examine the performance when using the learned state estimator and the factors affecting the performance of the state estimator in Section IV-C. We also present ablation experiments on different design choices in Section IV-D, specifically showing the effectiveness of residual actions and low-level skill feedback. Finally, we demonstrate the sim-to-real results in the real world in Section IV-E.

A. Experiment Setup

Simulation Setup. We use the IsaacGym simulator [44] to train our skill policy, planner policy, and state estimator. The simulation frequency is 120Hz, and the control frequency is 20Hz. We follow the standard setting [3], [26] where the episode starts from a stable grasp sampled from a grasp set, and the target goal is randomly sampled from SO(3). For training, we use four different objects (cylinders, tennis balls, apples, and piggy banks) with randomized physics and sizes. Note that although the number of objects is not large, they still represent a class of different shapes. With randomized physical parameters, they provide enough variations to develop a robust policy [3].

Fast Depth Rendering for In-Hand Objects. One bottleneck in many previous in-hand manipulation works is the slow vision rendering speed. Previous studies have shown that this is due to IsaacGym not performing parallel rendering [45]. Therefore, we implemented our own pseudo-z-buffer depth rendering algorithm and increased the depth rendering speed from 800 FPS to 10,000 FPS.

Hardware Setup. We use an Allegro Hand from Wonik Robotics for our experiments. The Allegro Hand is a multifingered robot hand with four fingers and four degrees of freedom per finger. It is mounted on a specially designed hand frame connected to the camera (see Figure 1). The low-level policy receives vision input. We use MobileSAM [46] to detect the object in the first frame and use Cutie [47] for tracking over time. We use the Lambda workstation with an RTX4090 GPU, which is sufficiently fast for processing vision input and sending commands at 20 Hz.

Baseline Methods. We compare with two baselines. First, we compare a baseline policy trained from scratch, as shown in Figure 2 (B). The baseline policy is also trained with PPO using similar reward and penalty terms. We comprehensively evaluate different observation choices, reward terms,

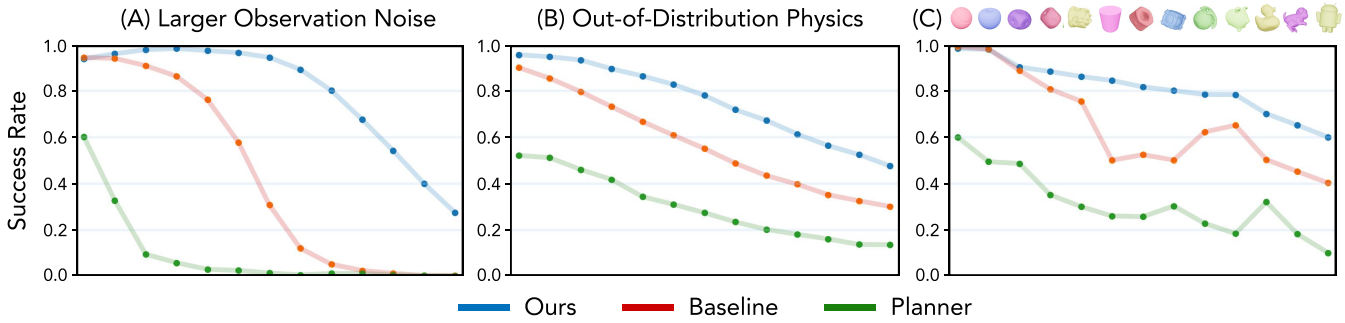


Fig. 4: **Robustness to different out-of-distribution scenarios.** We study the performance under larger orientation observation noises, physical randomizations, and shape variations. The perturbation becomes more challenging towards the right of each figure. Our policy and the baseline achieve similar success rates in easy cases but demonstrate more robustness in more challenging cases.

Method	Stage 1 → Stage 2	Torque ↓	Work ↓	DofAcc ↓	DofVel ↓	LinVel ↓
Baseline	$87.34_{\pm 12.13}$ → $52.32_{\pm 4.89}$	$0.43_{\pm 0.24}$	$0.78_{\pm 0.55}$	$0.55_{\pm 0.23}$	$0.79_{\pm 0.30}$	$0.47_{\pm 0.10}$
Ours	$88.25_{\pm 1.28}$ → $75.24_{\pm 1.27}$	$0.15_{\pm 0.01}$	$0.22_{\pm 0.01}$	$0.44_{\pm 0.02}$	$0.59_{\pm 0.02}$	$0.37_{\pm 0.01}$

TABLE I: **Stage 2 and Policy smoothness evaluation.** We show the gap between stage 1 and stage 2 and evaluate several metrics including energy metrics such as torque and work, robot smoothness such as joint acceleration (DofAcc) and joint velocity (DofVel), and object stability (LinVel). Our method is much stable than the baseline because we can leverage smooth transferrable skills. Work and LinVel are scaled by 10x.

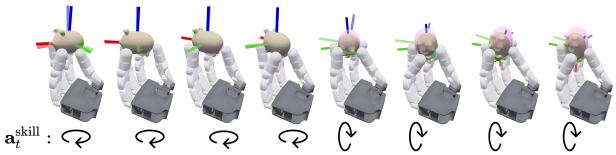


Fig. 5: **State estimation visualization across a rolled out trajectory.** We visualize poses from our learned estimator with the pink mesh and ground-truth poses with the green mesh. We observe generally robust estimation performance. Note that the sudden error in the middle is due to slipping during axes transition. Even in this challenging case, our state estimator can still predict relatively accurate state and guide the planner to finish the task.

and hyperparameters of the baseline and use the best set of hyperparameters here. For observation, we use a short window of proprioception, depth encoding, robot actions, and the noisy object state as input. Second, we include a heuristic planner as a baseline. This planner receives noisy pose data and heuristically selects a rotation axis to send to the low-level policy.

B. Policy Hierarchy with Pre-trained Skills

Policy Learning Performance. We first study the sample efficiency and training performance of our method compared to the baseline. The results are shown in Figure 3. We examine three different levels of object state noise as inputs. Specifically, we add noise sampled from a normal distribution, with the standard deviation annotated in the figure title. $r = 0.05$ represents a standard deviation of 0.05 radius, and $p = 0.005$ represents a standard deviation of 0.005 meter.

In the small noise case (A), both our policy and the baseline achieve an 85% success rate, but our policy converges $8\times$ faster than the baseline. As we increase the noise level, we find that the baseline policy becomes unstable and exhibits very high variance across different seeds (Figure 3 B), while our method remains stable. When we further increase the standard deviation of orientation noise from $0.05\,\text{rad}$ to $0.15\,\text{rad}$ and position noise from $0.5\,\text{cm}$ to $1.5\,\text{cm}$, our policy remains stable while the baseline policy fails to converge (Figure 3 C). The benefits of fast convergence and training stability come from the use of a pre-trained model. This pre-trained model provides a structured exploration space and thus avoids many meaningless random actions.

Note that, although the low-level skill policy has experienced additional samples compared to the baseline, simply increasing the training time of the baseline policy does not help. We have tried to train the baseline policy with $20\times$ more samples, but the conclusion remains the same. This experiment shows that the structural skill space of the low-level policy is more important than the number of samples it has seen during training.

Out-of-Distribution Robustness. We then study the out-of-distribution robustness of observation noise, physical randomizations, and shapes of a *trained model*. In this experiment, we use the models trained under small noise from the previous section because the baseline model tends to perform best in this training setting. The results are shown in Figure 4. The general trend across all three evaluations is that, although baseline performance is similar to our method in the easiest case (left data point in each plot), it drops rapidly when the test setting becomes more out-of-distribution.

Specifically, in the larger observation noise test, our policy can still maintain an 80% success rate when the baseline completely fails. When we increase the physical randomization, our method also consistently outperforms the baseline. We also test shape generalization. On the tennis ball and apple, the baseline slightly outperforms our method. However, it does not generalize well to other objects.

Another interesting observation is that the heuristic planner performs much worse than both methods. Even in the easiest setting, it is significantly inferior. This is because our low-level policy is far from perfect, and a heuristic planner with pre-defined rules cannot account for this. As we show in the ablation experiments, the design of the observation space for the planner is also very important.

C. Generalizable State Estimation

In the above section, the object states $\boldsymbol{p}_t$ and $\boldsymbol{q}_t$ are directly obtained from the simulator. However, to transfer the learned policy to the real world, we also need a general state estimator system for a diverse set of objects. In this section, we study the task accuracy using our proposed state estimator and how the policy's performance changes with predicted object states. Similar to the robustness test, we use the model trained with small noise and multiple objects.

Comparison to Baseline. We first study the policy performance using the predicted object state as the input. The results are shown in Table I. We compare the performance gap from stage 1 (using the noisy object states as inputs) to stage 2 (using the predicted states as input). Ideally, the performance gap would be small if the state estimation is accurate. We find that, although the baseline and our method achieve similar performance in stage 1, performance drops significantly in stage 2.

Policy Smoothness. To understand why the baseline policy has a larger performance drop, we quantitatively study various smoothness and energy metrics for our policy and the baseline. The energy metric is a critical aspect of successful sim-to-real transfer [3]. We measure energy metrics such as torque and work, robot smoothness metrics such as joint acceleration (DofAcc) and joint velocity (DofVel), and object stability (LinVel). We find that our method is significantly more stable than the baseline method. Because of this, the object movement is smooth and easy to predict given sensory feedback. Note that we tune the baseline with all of these penalties and select the best-performing one without affecting the success metric.

Our policy is significantly more stable than the baseline because of the stability of the low-level policy. We argue that when the task becomes increasingly difficult, balancing task performance with the smoothness and stability required for sim-to-real transfer becomes very challenging. Therefore, it is necessary to use a structured low-level policy, and we hope our analysis provides solid evidence for this argument.

Visualize Rotation Sequence and Predicted Rotations. We show a sequence of object reorientation, predicted orientations, and corresponding commanded actions in Figure 5. This is a challenging rollout as the object experiences slight slipping during transition. Even in this scenario, our recursive estimator could still produce predictions accurate enough to guide the planner to complete the task. Additional video results are available in our supplementary material.

D. Ablation Experiments

In addition to the high-level design of reusing pre-trained skills, we also make several critical design choices within our architecture. In this section, all experiments are conducted with the small noise setting.

Residual Actions and Low-Level Skill Feedback. Our planner takes low-level skill feedback $\boldsymbol{z}_t$ as input and outputs a residual action on top of the skill policy's output. We show their effect in Figure 6 (Left). First, without these components, our method achieves 76.37% accuracy in stage 1 (using noisy object states) and 58.12% accuracy in stage 2 (using predicted object states). This performance is worse than training from scratch. With this vanilla design, adding residual actions improves performance to 83.63% (stage 1) and 68.84% (stage 2). This experiment shows that even though the low-level policy is already tuned and achieves good performance on its tasks, we can still improve it by adding small error corrections. When we further incorporate skill feedback into the policy, performance improves to 88.25% and 75.24%, indicating the importance of feedback. Note that although our planner policy does not use vision information as input, the skill policy does take vision as input and predicts the object shape information encoded in $\boldsymbol{z}_t$.

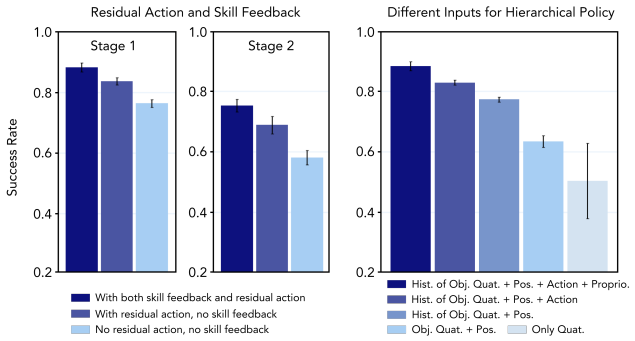


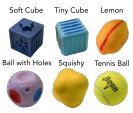
Fig. 6: **Left:** Ablation experiments on residual actions and skill feedback from low-level skill policies. **Right:** The effect of different input formats to the planner policy.

Inputs to the policy. Our planner incorporates several different types of observations (Section III-B), and we study the effect of each of them. The results are shown in Figure 6 (Right). The most basic version (using only quaternion difference as the input) achieves 50.37% accuracy and is similar to the heuristic planner baseline. Adding object position and including a history of observations both help increase performance, as the policy can infer object dynamics from this information. Adding planner actions further improves performance, as the planner also accounts for the effects of its decisions. Finally, incorporating proprioception yields the best performance, highlighting the importance of being aware of the low-level robot state. This result underscores the significance of closed-loop feedback between the planner and the low-level skill policies.

E. Real-World Experiments

Finally, we transfer the learned policies to the real world. We test our policies on six objects (Figure 7, left), using proprioception and segmented depth as inputs. Note that there is no cube in our training set, so most of the real-world objects are out-of-distribution. We evaluate two settings: *Single:* We set the target to rotate along one of the $x/y/z$-axes by $\pi/2$ or π radians. This setting does not require switching between different skills but tests the accuracy of pose estimation. We evaluate 30 trials per object. *Multi:* We set the target so that the policy needs to rotate over two axes by $\pi/2$. We evaluate 20 trials per object.

The results are shown in Figure 7. We find that our policy performs well in the real world for most of the objects.



Method	Soft Cube		Tiny Cube		Lemon		Ball w. Holes		Squishy		Tennis Ball	
	Single	Multi	Single	Multi	Single	Multi	Single	Multi	Single	Multi	Single	Multi
Ours	73.3	60.0	70.0	37.5	86.7	85.0	76.7	65.0	80.0	75.0	93.3	90.0

Fig. 7: **Real-world performance for our policy.** We evaluate six different objects and each is evaluated in two settings and show the success rate. We find that our policy performs well in the real world for most of the objects.

The most challenging one is the tiny cube, as it is small compared to the size of the hand, making it difficult to manipulate. The successful sim-to-real transfer results from two factors: 1) the use of transferable low-level skills and 2) a well-designed planner structure, as demonstrated in the ablation experiments IV-D. We also provide qualitative videos showcasing the reorientation in the real world in our supplementary material.

V. Conclusions and Limitations

In this work, we present a system for in-hand object reorientation by building a hierarchical policy with pre-trained low-level skills. To achieve robust and generalizable state estimation in the real world, we also learn a state estimator. We demonstrate successful deployment on multiple symmetric and textureless objects. Our work highlights the potential of moving away from robot policies trained from scratch: for a given task, we can dramatically improve training efficiency, robustness, and generalizability by leveraging pre-trained models for lower-level skills.

Limitations and Future Work. Our approach relies on the effectiveness of generalizable low-level policies. Importantly, it requires that there is no slipping between the finger and the object. This issue can be mitigated by incorporating tactile sensing into our framework. In our current setting, estimated pose errors accumulate over time. Integrating vision and touch for accurate and long-term pose tracking is a promising direction for future work.

Acknowledgement

This research was supported as a BAIR Open Research Common Project with Meta. In their academic roles at UC Berkeley, Haozhi Qi and Jitendra Malik are supported in part by DARPA Machine Common Sense (MCS) and ONR MURI N00014-21-1-2801, Brent Yi is supported by the NSF Graduate Research Fellowship Program under Grant DGE 2146752, and Haozhi Qi, Brent Yi, and Yi Ma are partially supported by ONR N00014-22-1-2102 and the InnoHK HKCRC grant. Roberto Calandra is funded by the German Research Foundation (DFG, Deutsche Forschungsgemeinschaft) as part of Germany's Excellence Strategy – EXC 2050/1 – Project ID 390696704 – Cluster of Excellence "Centre for Tactile Internet with Human-in-the-Loop" (CeTI) of Technische Universität Dresden. We thank Jiayuan Mao and Qiyang Li for their feedback on related literature. We thank Xinru Yang for her help on real-world videos.

References

[1] OpenAI, M. Andrychowicz, B. Baker, M. Chociej, R. Józefowicz, B. McGrew, J. Pachocki, A. Petron, M. Plappert, G. Powell, A. Ray, J. Schneider, S. Sidor, J. Tobin, P. Welinder, L. Weng, and W. Zaremba, "Learning dexterous in-hand manipulation," *IJRR*, 2019.

[2] OpenAI, I. Akkaya, M. Andrychowicz, M. Chociej, M. Litwin, B. McGrew, A. Petron, A. Paino, M. Plappert, G. Powell, R. Ribas, J. Schneider, N. Tezak, J. Tworek, P. Welinder, L. Weng, Q. Yuan, W. Zaremba, and L. Zhang, "Solving rubik's cube with a robot hand," *arXiv:1910.07113*, 2019.

[3] H. Qi, A. Kumar, R. Calandra, Y. Ma, and J. Malik, "In-hand object rotation via rapid motor adaptation," in *CoRL*, 2022.

[4] T. Chen, M. Tippur, S. Wu, V. Kumar, E. Adelson, and P. Agrawal, "Visual dexterity: In-hand reorientation of novel and complex object shapes," *Science Robotics*, 2023.

[5] A. Handa, A. Allshire, V. Makoviychuk, A. Petrenko, R. Singh, J. Liu, D. Makoviichuk, K. Van Wyk, A. Zhurkevich, B. Sundaralingam, Y. Narang, J.-F. Lafleche, D. Fox, and G. State, "Dextreme: Transfer of agile in-hand manipulation from simulation to reality," in *ICRA*, 2023.

[6] H. Qi, B. Yi, S. Suresh, M. Lambeta, Y. Ma, R. Calandra, and J. Malik, "General in-hand object rotation with vision and touch," in *CoRL*, 2023.

[7] J. Diedrichsen and K. Kornysheva, "Motor skill learning between selection and execution," *Trends in cognitive sciences*, 2015.

[8] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *CVPR*, 2014.

[9] K. He, G. Gkioxari, P. Dollár, and R. Girshick, "Mask r-cnn," in *ICCV*, 2017.

[10] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," in *ACL*, 2019.

[11] L. P. Kaelbling and T. Lozano-Pérez, "Hierarchical task and motion planning in the now," in *ICRA*, 2011.

[12] A. G. Barto and S. Mahadevan, "Recent advances in hierarchical reinforcement learning," *Discrete event dynamic systems*, 2003.

[13] Y. Labbé, L. Manuelli, A. Mousavian, S. Tyree, S. Birchfield, J. Tremblay, J. Carpentier, M. Aubry, D. Fox, and J. Sivic, "Megapose: 6d pose estimation of novel objects via render & compare," in *CoRL*, 2022.

[14] B. Wen and K. Bekris, "Bundletrack: 6d pose tracking for novel objects without instance or category-level 3d models," in *IROS*, 2021.

[15] L. Rostel, J. Pitz, L. Sievers, and B. Bauml, "Estimator-coupled reinforcement learning for robust purely tactile in-hand manipulation," in *Humanoids*, 2023.

[16] L. Han and J. C. Trinkle, "Dextrous manipulation by rolling and finger gaiting," in *ICRA*, 1998.

[17] J.-P. Saut, A. Sahbani, S. El-Khoury, and V. Perdereau, "Dexterous manipulation planning using probabilistic roadmaps in continuous grasp subspaces," in *IROS*, 2007.

[18] D. Rus, "In-hand dexterous manipulation of piecewise-smooth 3-d objects," *IJRR*, 1999.

[19] Y. Bai and C. K. Liu, "Dexterous manipulation using both palm and fingers," in *ICRA*, 2014.

[20] I. Mordatch, Z. Popović, and E. Todorov, "Contact-invariant optimization for hand manipulation," in *Eurographics*, 2012.

[21] R. Fearing, "Implementing a force strategy for object re-orientation," in *ICRA*, 1986.

[22] C. Teeple, B. Aktas, M. C.-S. Yuen, G. Kim, R. D. Howe, and R. Wood, "Controlling palm-object interactions via friction for enhanced in-hand manipulation," *RA-L*, 2022.

[23] A. S. Morgan, K. Hang, B. Wen, K. Bekris, and A. M. Dollar, "Complex in-hand manipulation via compliance-enabled finger gaiting and multi-modal planning," *RA-L*, 2022.

[24] S. Patidar, A. Sieler, and O. Brock, "In-hand cube reconfiguration: Simplified," in *IROS*, 2023.

[25] A. Sieler and O. Brock, "Dexterous soft hands linearize feedback-control for in-hand manipulation," in *IROS*, 2023.

[26] G. Khandate, S. Shang, E. T. Chang, T. L. Saidi, J. Adams, and M. Ciocarlie, "Sampling-based exploration for reinforcement learning of dexterous manipulation," in *RSS*, 2023.

[27] B. Wen, J. Tremblay, V. Blukis, S. Tyree, T. Müller, A. Evans, D. Fox, J. Kautz, and S. Birchfield, "Bundlesdf: Neural 6-dof tracking and 3d reconstruction of unknown objects," in *CVPR*, 2023.

[28] S. Suresh, H. Qi, T. Wu, T. Fan, L. Pineda, M. Lambeta, J. Malik, M. Kalakrishnan, R. Calandra, M. Kaess, *et al.*, "Neural feels with neural fields: Visuo-tactile perception for in-hand manipulation," *arXiv:2312.13469*, 2023.

[29] Y. Chen, C. Wang, L. Fei-Fei, and C. K. Liu, "Sequential dexterity: Chaining dexterous policies for long-horizon manipulation," in *CoRL*, 2023.

[30] M. T. Mason, *Mechanics of robotic manipulation*. MIT press, 2001.

[31] K. Pertsch, Y. Lee, and J. Lim, "Accelerating reinforcement learning with learned skill priors," in *CoRL*, 2021.

[32] R. S. Sutton, D. Precup, and S. Singh, "Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning," *Artificial Intelligence*, 1999.

[33] P.-L. Bacon, J. Harb, and D. Precup, "The option-critic architecture," in *AAAI*, 2017.

[34] A. C. Li, C. Florensa, I. Clavera, and P. Abbeel, "Sub-policy adaptation for hierarchical reinforcement learning," in *ICLR*, 2020.

[35] P. Ranchod, B. Rosman, and G. Konidaris, "Nonparametric bayesian reward segmentation for skill discovery using inverse reinforcement learning," in *IROS*, 2015.

[36] A. Sharma, M. Sharma, N. Rhinehart, and K. M. Kitani, "Directed-info gail: Learning hierarchical policies from unsegmented demonstrations using directed information," in *ICLR*, 2019.

[37] S. Park, D. Ghosh, B. Eysenbach, and S. Levine, "Hiql: Offline goal-conditioned rl with latent states as actions," *arXiv:2307.11949*, 2023.

[38] A. Bhatt, A. Sieler, S. Puhlmann, and O. Brock, "Surprisingly robust in-hand manipulation: An empirical study," in *RSS*, 2022.

[39] G. Khandate, C. Mehlman, X. Wei, and M. Ciocarlie, "Dexterous in-hand manipulation by guiding exploration with simple sub-skill controllers," *arXiv:2303.03533*, 2023.

[40] A. Gupta, J. Yu, T. Z. Zhao, V. Kumar, A. Rovinsky, K. Xu, T. Devlin, and S. Levine, "Reset-free reinforcement learning via multi-task learning: Learning dexterous manipulation behaviors without human intervention," in *ICRA*, 2021.

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

[42] Y. Zhou, C. Barnes, J. Lu, J. Yang, and H. Li, "On the continuity of rotation representations in neural networks," in *CVPR*, 2019.

[43] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *ICLR*, 2015.

[44] V. Makoviychuk, L. Wawrzyniak, Y. Guo, M. Lu, K. Storey, M. Macklin, D. Hoeller, N. Rudin, A. Allshire, A. Handa, and G. State, "Isaac gym: High performance gpu-based physics simulation for robot learning," in *NeurIPS Datasets and Benchmarks*, 2021.

[45] J. Gu, F. Xiang, X. Li, Z. Ling, X. Liu, T. Mu, Y. Tang, S. Tao, X. Wei, Y. Yao, X. Yuan, P. Xie, Z. Huang, R. Chen, and H. Su, "Maniskill2: A unified benchmark for generalizable manipulation skills," in *ICLR*, 2023.

[46] C. Zhang, D. Han, Y. Qiao, J. U. Kim, S.-H. Bae, S. Lee, and C. S. Hong, "Faster segment anything: Towards lightweight sam for mobile applications," *arXiv:2306.14289*, 2023.

[47] H. K. Cheng, S. W. Oh, B. Price, J.-Y. Lee, and A. Schwing, "Putting the object back into video object segmentation," in *CVPR*, 2024.