Visual Dexterity: In-hand Dexterous Manipulation from Depth

Tao Chen [1 2] **Megha Tippur** [2] **Siyang Wu** [3] **Vikash Kumar** [4] **Edward Adelson** [2] **Pulkit Agrawal** [1 2]

Abstract

In-hand object reorientation is necessary for performing many dexterous manipulation tasks, such as tool use in unstructured environments that remain beyond the reach of current robots. Prior works built reorientation systems that assume one or many of the following specific circumstances: reorienting only specific objects with simple shapes, limited range of reorientation, slow or quasi-static manipulation, etc. We overcome these limitations and present a general object reorientation controller that is trained in simulation and evaluated in the real world. Our system uses readings from a single commodity depth camera to dynamically reorient complex objects by any amount in real time. The controller generalizes to new objects not used during training. It even demonstrates some capability of reorienting objects in the air held by a downward-facing hand that must counteract gravity during reorientation.

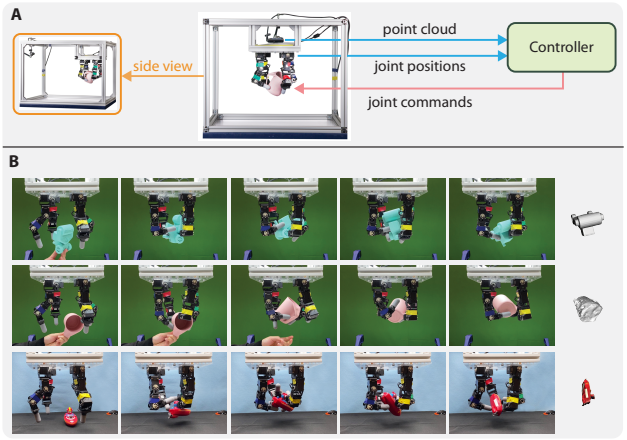

Figure 1. **(A)**: the front and side views of our real-world setup. **(B)**: Visualization of the same controller reorienting three different objects. The rightmost column shows the target orientation. The first two rows are instances of a four-fingered hand reorienting objects in the air. The last row shows reorientation with the help of a supporting surface (i.e., extrinsic dexterity).

Introduction

The dexterity of the human hand is vital to a wide range of daily tasks such as loading dishes in a dishwasher and fastening bolts. Despite a long-standing interest in creating similarly capable robotic systems, current robots are far behind in their versatility, dexterity, and robustness. In-hand object reorientation, illustrated in Figure 1, is a dexterous manipulation problem where the goal is to manipulate a hand-held object from an arbitrary initial orientation to an arbitrary target orientation (Mason et al., 1989; Salisbury & Craig, 1982; Mordatch et al., 2012; Bai & Liu, 2014; Kumar et al., 2014; Chen et al., 2022). Object reorientation occupies a special place in manipulation because it is a precursor to flexible tool use. After picking a tool, the robot must orient the tool in an appropriate configuration to use it.

A reorientation system ready for the real world should satisfy multiple criteria: it should be able to reorient objects into any orientation, generalize to new objects, and operate in real-time using data from commodity sensors. Some seemingly benign setup choices can make the system impractical for real-world deployment. For instance, consider the choice of placing multiple cameras around the workspace to reduce occlusion in viewing the object being manipulated (Andrychowicz et al., 2020; OpenAI et al., 2019). For a mobile manipulator, such camera placements are impractical. Similarly, performing reorientation under the assumption of an upward-facing hand (Andrychowicz et al., 2020; Nagabandi et al., 2020) is much easier. With a downward-facing hand, the hand must manipulate the object while simultaneously counteracting gravity. Small errors in finger motion can result in the object falling down.

Even without real-world setup constraints, object reorientation is challenging because it requires coordinated movement between multiple fingers resulting in a high-dimensional control space. The robot must control the amount of applied force, when to apply it, and where the

[1]Improbable AI Lab, MIT, USA [2]Department of Electrical Engineering and Computer Science, MIT, USA [3]Institute for Interdisciplinary Information Sciences, Tsinghua University, China [4]Meta AI, USA. Correspondence to: tao chen <taochen@mit.edu>.

Workshop on New Frontiers in Learning, Control, and Dynamical Systems at the International Conference on Machine Learning (ICML), Honolulu, Hawaii, USA, 2023.

fingers should make and break contact with the object. A majority of prior works constrain manipulation to simple convex shapes such as polygons or cylinders (Ishihara et al., 2006; Kumar et al., 2014; Calli & Dollar, 2017; Andrychowicz et al., 2020; Van Hoof et al., 2015; Abondance et al., 2020; Bhatt et al., 2021; Khandate et al., 2022). Other simplifying assumptions include designing specific movement patterns of fingers (Morgan et al., 2022; Bhatt et al., 2021), assuming fingers never make break contact with the object (Sundaralingam & Hermans, 2019; Jeong et al., 2020), hand being in upwards facing configuration (Bai & Liu, 2014; Nagabandi et al., 2020; Andrychowicz et al., 2020) or the manipulation being quasi-static (Morgan et al., 2022; Sievers et al., 2022). Such assumptions restrict the applicability of reorientation to a limited set of objects, scenarios, or orientations (e.g., along only a single axis).

In this paper, we present a system that advances the dexterous manipulation system's capability. Our system learns a controller in simulation for dynamic in-hand object reorientation and transfers to the real world in a zero-shot manner. The systematic choices of identifying the simulated robot's dynamics parameters using real-world data (i.e., system identification on robot dynamics, details in the Method section), domain randomization (Tobin et al., 2017), design of reward function and the hardware considerations including the number of fingers and the fingertip material enabled us to overcome the sim-to-real gap. We conducted experiments in the challenging downward-facing hand configuration. We tested the controller's ability to make use of an external support surface for reorientation (i.e., extrinsic dexterity (Dafle et al., 2014)) and the harder condition when the object is in the air without any supporting surface. The results demonstrate success in designing a *real-time* controller that can *dynamically* reorient *new* objects with *complex shapes* and *diverse materials* by any amount in the *full space of rotations* $\big(SO(3)\big)$ using inputs from just a *single commodity depth camera* and *joint encoders*. Our results provide convincing evidence that *sim-to-real* transfer is possible for challenging dynamic and contact-rich manipulation tasks.

Method

Given a random object in a random initial pose, the robot is tasked to reorient the object to a user-provided target orientation in $SO(3)$ space. We train a single vision-based object reorientation controller (or policy) in simulation (Isaac Gym (Makoviychuk et al., 2021)) to reorient hundreds of objects. The controller trained in simulation is directly deployed in the real world (i.e., zero-shot transfer).

Training the visuomotor policy

We use a *teacher-student* training paradigm for learning the controller. The paradigm has been used to learn object reorientation policy in simulation from visual and proprioceptive observations (Chen et al., 2022). However, a separate policy was trained per object. Secondly, it required more than a week to train the student vision policy for a single object on an NVIDIA V100 GPU. We developed a two-stage student training (Teacher-student^2) framework that substantially speeds up the vision student policy learning. Using this framework, we were able to learn a vision policy that operates across a diverse set of objects and generalizes to objects with different shapes and physical parameters.

Teacher policy: reinforcement learning with privileged information

The teacher policy ($\pi^{\mathcal{E}}$) is trained using reinforcement learning. The observation input includes proprioceptive state information, object state, and target orientation. The policy outputs the relative joint position changes in 12Hz. To smooth the commands sent to the low-level controller, we usethe *exponential moving average* of actions $\bar{\boldsymbol{a}}_t = \alpha \boldsymbol{a}_t + (1-\alpha)\bar{\boldsymbol{a}}_{t-1}$ where $\alpha \in [0, 1]$. The reward function is described in Section A.1.

Student policy - imitation learning from depth observations

The student policy ($\pi^{\mathcal{S}}$) is trained in simulation with the purpose of being deployed in the real world. Since the sim-to-real gap for depth data is less pronounced than RGB data, we only use the depth images provided by the camera along with readings from joint encoders. We represent the depth data as a point cloud in the robot's base link frame. To enable the neural network representing $\pi^{\mathcal{S}}$ to model the spatial relationship between the fingers and the object, we express the robot's current configuration by showing the policy a point cloud representing points sampled on the surface of the fingers. We concatenate the point cloud obtained from the camera along with the generated point cloud of the hand. We denote this scene point cloud as $\boldsymbol{P}_t^s$.

Goal representation Instead of providing the goal orientation as a pose which has generalization issues discussed above, the goal is represented as the object's point cloud in the target orientation $\boldsymbol{P}^g$.

Observation space The input to $\pi^{\mathcal{S}}$ is the point cloud $\boldsymbol{P}_t = \boldsymbol{P}_t^s \cup \boldsymbol{P}^g$.

Architecture The critical requirement for the vision policy is to run at a high enough frequency to enable real-time control. For fast computation, we designed a sparse convolutional neural network with a gated recurrent unit (GRU; (Cho et al., 2014)) to process point cloud ($\boldsymbol{P}_t$) using

the Minkowski Engine (Choy et al., 2019).

Optimization The student policy $\pi^{\mathcal{S}}$ is trained using DAGGER (Ross et al., 2011) to imitate the teacher policy $\pi^{\mathcal{E}}$.

Need for two-stage student learning We found training a vision policy in simulation to be slow, consuming 20+ days on an NVIDIA V100 GPU. The main reason for slow training is that the simulator performs rendering to generate a point cloud which consumes a substantial amount of time and GPU memory. To reduce training time, we generated *synthetic point clouds* by uniformly sampling points on the object and robot meshes used by the simulator. The *synthetic point cloud* is also complete (i.e., no occlusions), which makes training easier. The vision policy ($\pi_1^{\mathcal{S}}$) can be trained with *synthetic point cloud* in less than three days, i.e., 7× speedup (**stage 1**). However, the policy, $\pi_1^{\mathcal{S}}$, cannot be deployed in the real world because it operates on an idealized point cloud (i.e., no occlusions). Therefore, once the student reaches high performance, we initiate **stage 2**, where the policy is finetuned with the rendered point cloud. Such finetuning is quick in wall-clock time (around one day), and the resulting policy ($\pi_2^{\mathcal{S}}$) performs better than training from scratch with rendered point clouds. An additional benefit of the two-stage student policy training is that $\pi_1^{\mathcal{S}}$ is agnostic to the camera pose. Therefore a policy from a new viewpoint ($\pi_2^{\mathcal{S}}$) can be quickly obtained by finetuning using rendered point clouds from that camera pose. Training the vision policy from scratch is not necessary. Details on how to compute the point cloud are in Section A.2.

Overcoming the simulation to reality gap

There are two main sources of gap between simulation and reality. The first one is *dynamics* gap that arises from differences in the robot dynamics, the approximation in the simulator's contact model, and differences in object dynamics that depends on material properties such as friction, etc. The other source is *perception* gap caused by differences in statistics of sensor readings and/or noise. One way to overcome these gaps is to train a single policy across many different settings of the simulation parameters (i.e., domain randomization (Tobin et al., 2017)). The success of domain randomization hinges on the hope that the real world can be well approximated by one of the many simulation parameter settings used during training. The chances that one of the parameter settings being used in simulation being close to the real world increases by randomizing parameters over a larger range. However, excessive randomization may result in an overly conservative policy with low performance. Therefore, we make design choices that reduce the need for domain randomization and use it only when needed.

The *perception* gap is reduced by using only depth readings from the camera which is represented as a point cloud and

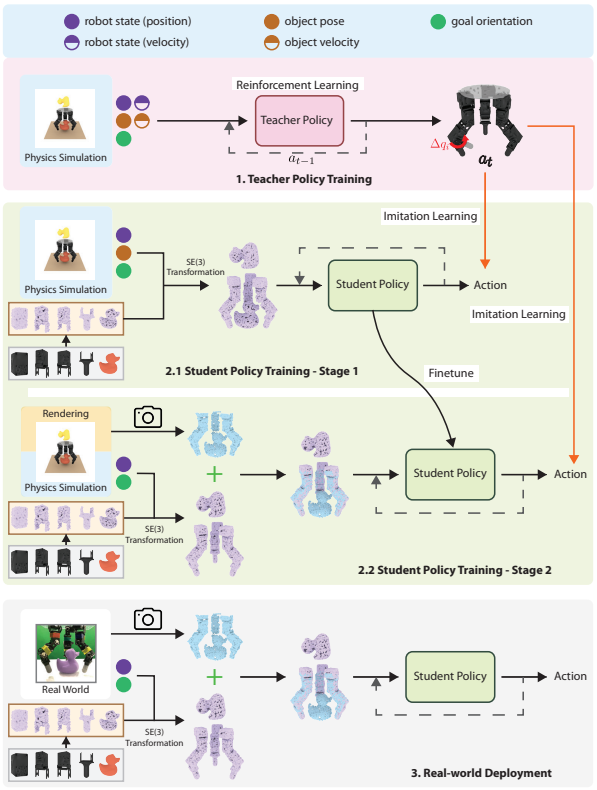

Figure 2. Teacher-student^2 framework

is sufficient for reorienting objects with different shapes. To account for noise in depth readings, we add noise to the simulated point cloud The *dynamics* gap can be reduced by identifying simulation parameters that are closest to the real world. While such identification is possible for the robotic manipulator, it is infeasible to do so for object dynamics that vary in material, mass distribution, etc. Therefore, we perform system identification on the robot dynamics and use only small randomization to account for unmodeled errors. We use a larger range of domain randomization (Section S4.2.2) on the object and environment dynamics. To make the policy more robust to unmodeled real-world physics, we apply random forces on the object during training which pressures the policy to reorient objects while being robust to external disturbance. Lastly, to increase compliance and friction between the object and the manipulator, we use soft fingertips. Such a choice makes the system more tolerant to errors in control commands. Empirically we noticed that soft fingertips make the robot less aggressive and result in fewer overshoots.

System identification of robot dynamics

We build the Unified Robot Description Format (URDF) models for the hands using their CAD models. While accurate kinematics parameters can be obtained from the CAD

model, the dynamics parameters, such as joint damping and stiffness, must be estimated. One way of identifying dynamics parameters is to write down the equations of motion (or the dynamics model) and solve for the unknown variables using the collected robot's motion trajectories. The Isaac Gym simulator we use already has a built-in dynamics model. But because the simulator's code is not open-source, we do not have access to the precise dynamics model nor any access to gradients of dynamics parameters. We, therefore, propose to use a black-box approach that leverages the ability of Isaac Gym to perform massively parallel simulations. We spawn a large number of simulations with different dynamics parameters and use the one that has the closest match to the real robot's motion.

Let $\boldsymbol{\lambda}_i \in \boldsymbol{\Lambda}$ denote the dynamics parameter of the i^{th} simulated robot ($C^{\boldsymbol{\lambda}_i}$), where $\boldsymbol{\Lambda}$ denotes the entire set of dynamics parameter values over which search is performed. To evaluate the similarity between the motion of $C^{\boldsymbol{\lambda}_i}$ and the real robot (C^{real}), we compute the score: $h(\boldsymbol{q}_A^{C^{real}}(\cdot), \boldsymbol{q}_A^{C^{\boldsymbol{\lambda}_i}}(\cdot)) = -\left\|\boldsymbol{q}_A^{C^{real}}(\cdot) - \boldsymbol{q}_A^{C^{\boldsymbol{\lambda}_i}}(\cdot)\right\|_2^2$, where $\boldsymbol{q}_A^C(\cdot)$ represents the joint position trajectories of a robot C given action commands $\boldsymbol{A}(\cdot)$. The closer the motion of the simulated robot is to that of the real world, the higher the score will be. We use the black-box optimization method of CMAES (Hansen et al., 2003), an instance of evolutionary search algorithms, to determine the optimal dynamics parameter: $\boldsymbol{\lambda}^* = \arg\max_{\boldsymbol{\lambda}\in\boldsymbol{\Lambda}} h(\boldsymbol{q}_A^{C^{real}}(\cdot), \boldsymbol{q}_A^{C^{\boldsymbol{\lambda}}}(\cdot))$. Note that it might be impossible to find a simulated robot that exactly matches the real robot due to the approximate parameterization of real-world dynamics in simulation and the stochasticity in the real-world resulting from actuation/sensing noise.

Results

We experiment with the hand in the downward-facing configuration in two settings: with and without a supporting table. To quantitatively measure the orientation error, we use an OptiTrack motion capture system to track object pose. We define error as the distance between the goal orientation and the object orientation when the controller predicts it has reached the target configuration and stops. We use seven objects from the training dataset (150 objects in total), and five objects from the held-out test dataset. Each object was tested 20 times in each testing setup. We 3D print these objects to ensure the shape of objects in simulation and the real world is identical, which is helpful in evaluating the extent of *sim-to-real* transfer.

Extrinsic dexterity: object reorientation with a supporting surface

We first report results on the easier problem of reorienting objects when the table is present below the hand to support the object. First, we experimented with a three-fingered hand. With table support, three fingers are sufficient for the reorientation task. The error distribution for different objects, when tested on a table surface covered with a white cloth (material M1 in Figure 3). The seven train objects can be reoriented successfully within an error of 0.4 radians 81% of the time. Performance on the five OOD test objects is lower with a success rate of 45%. Qualitatively observing the robot behavior revealed that one cause of failures was object overshooting the target orientation or the finger slipping across the object, especially for OOD objects. We found that using *soft* fingertips helps improve the system's performance (see Figure 3).

Object reorientation on different supporting materials Changing the table surface results changes the dynamics of object motion. We tested if our controller is robust to a diverse set of materials that have different surface structures, roughness, and friction, leading to different system dynamics. We evaluate with one in-distribution object (object $\#5$) and one out-of-distribution object (object $\#10$). Figure 3 show that our controller performs well on different supporting materials.

Towards object reorientation in air

While both three and four-fingered hands can reorient objects on a supporting surface (Figure 4), only the four-fingered hand is successful at in-air reorientation . We hypothesize this to be the case because, with four fingers, there are more ways of reorienting the object, making it easier for policy optimization to find one solution.

$SO(3)$ OBJECT REORIENTATION IN AIR

Simulation analysis reveals that object dropping is the most significant source of errors. Dropping rates vary substantially across objects. Real-world results follow the same trend. The dropping rate of a truck-shaped object ($\#5$) was 23%, significantly lower than the dropping rate of 56% for an out-of-distribution duck-shaped object ($\#10$). The dropping rate for the same object shape in the simulation was around 20% showing a sim-to-real gap. However, it remains unclear if the difference in performance can be attributed to the simulator being an approximate model of the real world or whether the object in the real world is much harder to manipulate.

Our controller performs dynamic reorientation, and the median time for successful manipulation across objects and randomly sampled orientation distances in the full $SO(3)$

space is less than 7s, which makes it the fastest in-air reorientation controller operating in the full $SO(3)$ space. The reorientation times in the real world are longer than in simulation, which we believe results from contact dynamics in the real world being different from simulation.

References

Abondance, S., Teeple, C. B., and Wood, R. J. A dexterous soft robotic hand for delicate in-hand manipulation. *IEEE Robotics and Automation Letters*, 5(4):5502–5509, 2020. 2

Andrychowicz, O. M., Baker, B., Chociej, M., Józefowicz, R., McGrew, B., Pachocki, J., Petron, A., Plappert, M., Powell, G., Ray, A., Schneider, J., Sidor, S., Tobin, J., Welinder, P., Weng, L., and Zaremba, W. Learning dexterous in-hand manipulation. *The International Journal of Robotics Research*, 39(1):3–20, 2020. 1, 2

Bai, Y. and Liu, C. K. Dexterous manipulation using both palm and fingers. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1560–1565. IEEE, 2014. 1, 2

Bhatt, A., Sieler, A., Puhlmann, S., and Brock, O. Surprisingly robust in-hand manipulation: An empirical study. *Robotics: Science and Systems (RSS)*, 2021. 2

Calli, B. and Dollar, A. M. Vision-based model predictive control for within-hand precision manipulation with underactuated grippers. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2839–2845. IEEE, 2017. 2

Chen, T., Xu, J., and Agrawal, P. A system for general in-hand object re-orientation. In *Conference on Robot Learning*, pp. 297–307. PMLR, 2022. 1, 2

Cho, K., van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., and Bengio, Y. Learning phrase representations using RNN encoder–decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 1724–1734, October 2014. 2

Choy, C., Gwak, J., and Savarese, S. 4d spatio-temporal convnets: Minkowski convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 3075–3084, 2019. 3

Dafle, N. C., Rodriguez, A., Paolini, R., Tang, B., Srinivasa, S. S., Erdmann, M., Mason, M. T., Lundberg, I., Staab, H., and Fuhlbrigge, T. Extrinsic dexterity: In-hand manipulation with external forces. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1578–1585. IEEE, 2014. 2

Hansen, N., Müller, S. D., and Koumoutsakos, P. Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (cma-es). *Evolutionary computation*, 11(1):1–18, 2003. 4

Ishihara, T., Namiki, A., Ishikawa, M., and Shimojo, M. Dynamic pen spinning using a high-speed multifingered hand with high-speed tactile sensor. In *6th IEEE-RAS International Conference on Humanoid Robots*, pp. 258–263. IEEE, 2006. 2

Jeong, R., Springenberg, J. T., Kay, J., Zheng, D., Zhou, Y., Galashov, A., Heess, N., and Nori, F. Learning dexterous manipulation from suboptimal experts. In *Conference on Robot Learning*, pp. 915–934. PMLR, 2020. 2

Khandate, G., Haas-Heger, M., and Ciocarlie, M. On the feasibility of learning finger-gaiting in-hand manipulation with intrinsic sensing. In *2022 International Conference on Robotics and Automation (ICRA)*, pp. 2752–2758. IEEE, 2022. 2

Kumar, V., Tassa, Y., Erez, T., and Todorov, E. Real-time behaviour synthesis for dynamic hand-manipulation. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 6808–6815. IEEE, 2014. 1, 2

Makoviychuk, V., Wawrzyniak, L., Guo, Y., Lu, M., Storey, K., Macklin, M., Hoeller, D., Rudin, N., Allshire, A., Handa, A., and State, G. Isaac gym: High performance GPU based physics simulation for robot learning. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2021. 2

Mason, M. T., Salisbury, J. K., and Parker, J. K. *Robot hands and the mechanics of manipulation*. The MIT Press, 1989. 1

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

Morgan, A. S., Hang, K., Wen, B., Bekris, K., and Dollar, A. M. Complex in-hand manipulation via compliance-enabled finger gaiting and multi-modal planning. *IEEE Robotics and Automation Letters*, 7(2):4821–4828, 2022. 2

Nagabandi, A., Konolige, K., Levine, S., and Kumar, V. Deep dynamics models for learning dexterous manipulation. In *Conference on Robot Learning*, pp. 1101–1112. PMLR, 2020. 1, 2

OpenAI, Akkaya, I., Andrychowicz, M., Chociej, M., Litwin, M., McGrew, B., Petron, A., Paino, A., Plappert, M., Powell, G., Ribas, R., et al. Solving rubik's cube with a robot hand. *arXiv preprint arXiv:1910.07113*, 2019. 1

Ross, S., Gordon, G., and Bagnell, D. 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*, pp.

627–635. JMLR Workshop and Conference Proceedings, 2011. 3

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

Sievers, L., Pitz, J., and Bäuml, B. Learning purely tactile in-hand manipulation with a torque-controlled hand. In *2022 International Conference on Robotics and Automation (ICRA)*, pp. 2745–2751, 2022. doi: 10.1109/ICRA46639.2022.9812093. 2

Sundaralingam, B. and Hermans, T. Relaxed-rigidity constraints: kinematic trajectory optimization and collision avoidance for in-grasp manipulation. *Autonomous Robots*, 43(2):469–483, 2019. 2

Tobin, J., Fong, R., Ray, A., Schneider, J., Zaremba, W., and Abbeel, P. 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)*, pp. 23–30. IEEE, 2017. 2, 3

Van Hoof, H., Hermans, T., Neumann, G., and Peters, J. Learning robot in-hand manipulation with tactile features. In *2015 IEEE-RAS 15th International Conference on Humanoid Robots (Humanoids)*, pp. 121–127. IEEE, 2015. 2

A. Training details

A.1. Reward function for teacher policy learning

Reward Our reward function consists of the following terms: a sparse and a dense reward term for minimizing the orientation distance (Equation 1 and 2), a penalty term when fingertips move away from the object (Equation 3), a penalty term for using much energy (Equation 4), a penalty for pushing objects away (Equation 5), a penalty for the contact between the object and table (Equation 6) and a penalty for using the penultimate joint instead of the fingertip for reorientation (Equation 7).

$$
\begin{aligned}
r_t =& c_1 \mathbb{1}(\text{Task successful}) && (1)\\
&+ c_2 \frac{1}{|\Delta\theta_t| + \epsilon_\theta} && (2)\\
&+ c_3 \sum_{i=1}^{G} \left\| \boldsymbol{p}_t^{f_i} - \boldsymbol{p}_t^{o} \right\|_2^2 && (3)\\
&+ c_4 |\dot{\boldsymbol{q}}_t|^T |\boldsymbol{\tau}_t| && (4)\\
&+ c_5 \mathbb{1}(\|\boldsymbol{p}_t^{o}\|_2^2 > \bar{p}) && (5)\\
&+ c_6 \mathbb{1}(\text{object contacts with the table}) && (6)\\
&+ c_7 \sum_{i=1}^{N} \mathbb{1}(p_{t,z}^{f_i} > \bar{p}_z) && (7)
\end{aligned}
$$

where $c_1, c_2 > 0$, and $c_3, c_4, c_5 < 0, c_6, c_7 < 0$ are coefficients, $\mathbb{1}$ is an indicator function, ϵ_θ and $\bar{p}$ are constants, $\boldsymbol{p}_t^{f_i}$ is the fingertip position of i^{th} finger, $\boldsymbol{p}_t^{o}$ is the object center position, $\boldsymbol{\tau}_t$ is the vector of the joint torques.

A.2. Point cloud in the two-stage student training

Stage 1: details of synthetic point cloud In stage 1, the simulation is not used for rendering but only for physics simulation. We generate the point cloud for each link on the manipulator and object by sampling K points on their meshes in the following way: let the point cloud of link l_j in the local coordinate frame of the link be denoted as $\boldsymbol{P}^{l_j} \in \mathbb{R}^{K\times 3}$. Given link orientation ($\boldsymbol{R}_t^{l_j} \in \mathbb{R}^{3\times 3}$) and position ($\boldsymbol{p}_t^{l_j} \in \mathbb{R}^{3\times 1}$) at time step t, the point cloud can be computed in the global frame, $\boldsymbol{P}_t^{l_j} = \boldsymbol{P}^{l_j}(\boldsymbol{R}_t^{l_j})^T + (\boldsymbol{p}_t^{l_j})^T$. The point cloud representation of the entire scene is the union of point clouds of all the links, the object being manipulated, and the object in the goal orientation: $\boldsymbol{P}_t^{s} = \bigcup_{j=1}^{j=M} \boldsymbol{P}_t^{l_j}$ where M is the total number of links (bodies) in the environment. The point cloud $\boldsymbol{P}_t^{s}$ can be efficiently generated using matrix multiplication.

Stage 2: details of rendered point cloud In stage 2, we acquire depth images from the simulator at each time step and convert the depth images to point clouds (which we call *exteroceptive point cloud*) using the camera's intrinsic and extrinsic matrices. Note that such a point cloud is not complete due to occlusions from a single camera. We also convert the joint angle information into link poses via forward kinematics and then generate the complete point cloud of the robot (which we call *proprioceptive point cloud*). Note that such a proprioceptive point cloud of a robot can be easily obtained in the real world in real-time as well since we can get the joint positions on the real robot. The policy input consists of the union of the exteroceptive point cloud and the proprioceptive point cloud.

B. Quantitative Results

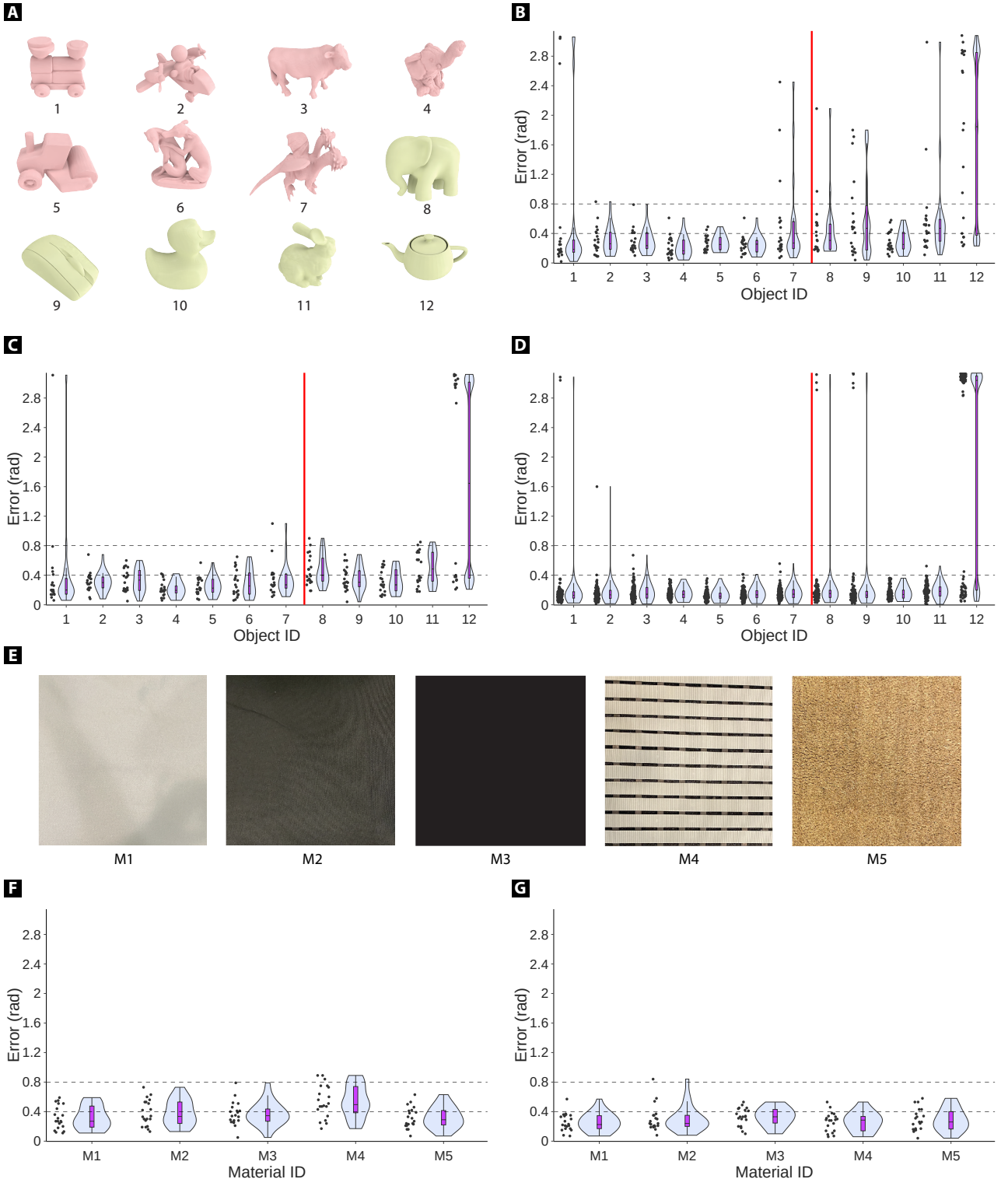

Figure 3. **(A)**: twelve objects with their IDs. The first seven objects are from the training dataset, and the last five are from the testing dataset. **(B)**, **(C)** show the *real-world* error distribution when using rigid and soft fingertips, respectively, on material M1. **(D)** shows the error distribution *in simulation* for each object as a *violin* plot. The error on test objects is higher, and soft fingertips exhibit better generalization. **(E)**: five table materials. **(F)** and **(G)** show the error distribution on different materials for object #5 and #10.

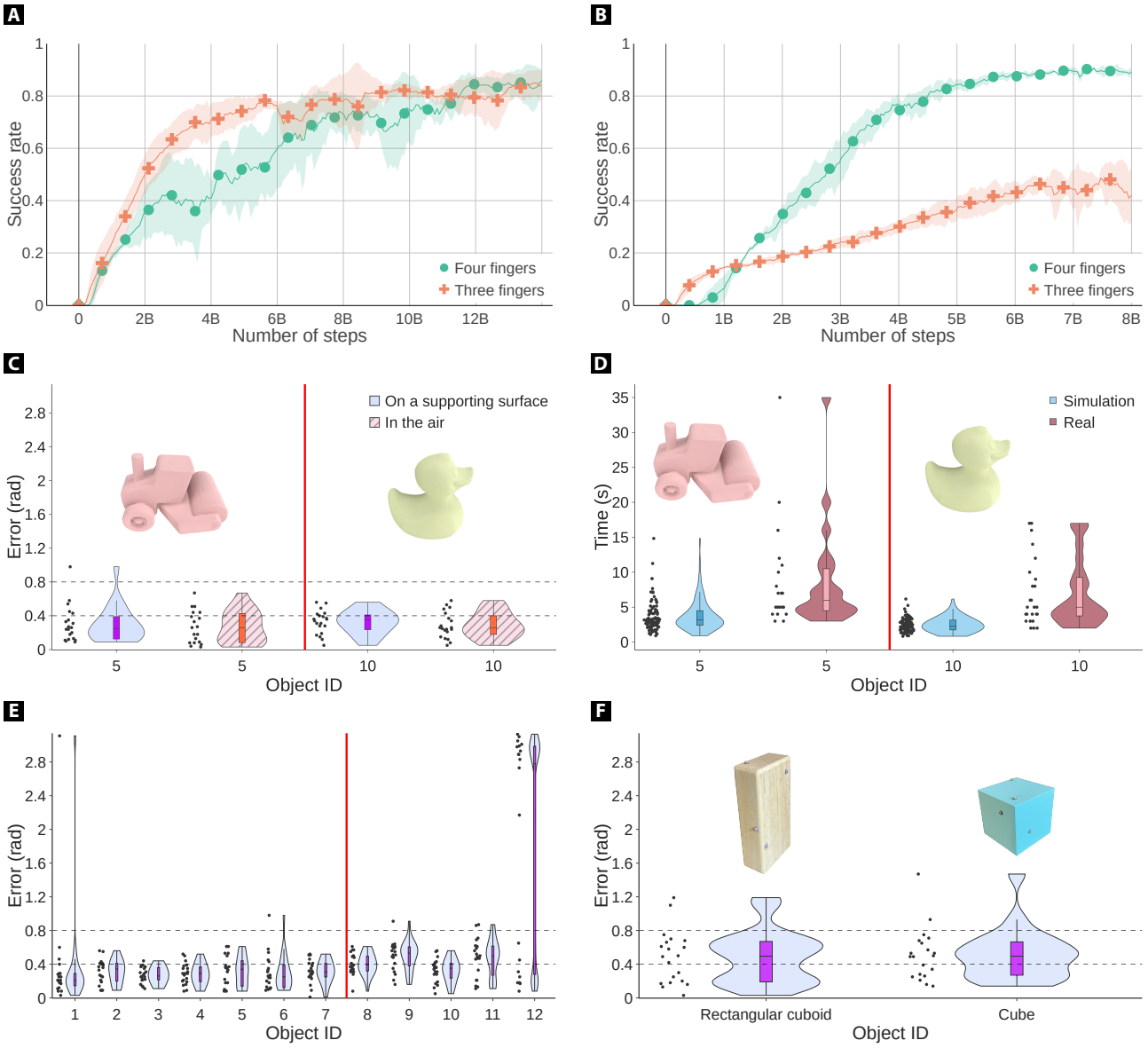

Figure 4. **(A)**: When training a controller to reorient objects with a supporting surface, the three-fingered and four-fingered hands achieve similar learning performance. **(B)**: However, when we incentivize the hands to lift the object during reorientation, the four-fingered hand outperforms the three-fingered hand substantially. **(C)**: We tested the controller performance with a four-fingered hand in the air. We collected 20 non-dropping testing cases for one in-distribution object and one out-of-distribution object. The error distribution is similar to that in the case of table-top reorientation. **(D)** shows the distribution of the episode time both in simulation and the real world. **(E)**: We show the same controller's performance on twelve objects with a supporting surface. **(F)**: We tested the controller on symmetric objects with a supporting surface. The controller behaves reasonably well even though it was never trained with symmetric objects.