ARTIFICIAL INTELLIGENCE

Visual dexterity: In-hand reorientation of novel and complex object shapes

Tao Chen[1,2], Megha Tippur[2], Siyang Wu[3], Vikash Kumar[4], Edward Adelson[2], Pulkit Agrawal[1,2,5]*

In-hand object reorientation is necessary for performing many dexterous manipulation tasks, such as tool use in less structured environments, which remain beyond the reach of current robots. Prior works built reorientation systems assuming one or many of the following conditions: reorienting only specific objects with simple shapes, limited range of reorientation, slow or quasi-static manipulation, simulation-only results, the need for specialized and costly sensor suites, and other constraints that make the system infeasible for real-world deployment. We present a general object reorientation controller that does not make these assumptions. It uses readings from a single commodity depth camera to dynamically reorient complex and new object shapes by any rotation in real time, with the median reorientation time being close to 7 seconds. The controller was trained using reinforcement learning in simulation and evaluated in the real world on new object shapes not used for training, including the most challenging scenario of reorienting objects held in the air by a downward-facing hand that must counteract gravity during reorientation. Our hardware platform only used open-source components that cost less than 5000 dollars. Although we demonstrate the ability to overcome assumptions in prior work, there is ample scope for improving absolute performance. For instance, the challenging duck-shaped object not used for training was dropped in 56% of the trials. When it was not dropped, our controller reoriented the object within 0.4 radians (23°) 75% of the time.

INTRODUCTION

The human hand's dexterity is vital to a wide range of daily tasks, such as rearranging objects, loading dishes in a dishwasher, fastening bolts, cutting vegetables, and other forms of tool use both inside and outside households. 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 Fig. 1, is a specific dexterous manipulation problem where the goal is to manipulate a handheld object from an arbitrary initial orientation to an arbitrary target orientation (*1*–*7*). 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. For example, a screwdriver can only be used if its head is aligned with the top of the screw. Object reorientation is, therefore, not only a litmus test for dexterity but also an enabler for many downstream manipulation tasks.

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 (*8*, *9*). For a mobile manipulator, such camera placements are impractical. Similarly, performing reorientation under the assumption that the hand is below the object (upward-facing hand configuration) (*8*–*10*) instead of the hand holding the object from the top (downward-facing hand configuration) 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. The upward-facing hand assumption makes control easier, but it limits the downstream use of the reorientation skill in many tool-use applications.

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 fingers should make and break contact with the object. The combination of continuous and discrete decisions leads to a challenging continuous-discrete optimization problem that is often computationally intractable. For computational feasibility, most prior works constrain manipulation to simple convex shapes, such as polygons or cylinders (*6*, *8*, *11*–*22*). Other simplifying assumptions include designing specific movement patterns of fingers (*18*, *23*), assuming that fingers never make and break contact with the object (*15*, *24*), hand being in an upward-facing configuration (*5*, *8*, *10*), or the manipulation being quasi-static (*23*, *25*). Such assumptions restrict the applicability of reorientation to a limited set of objects, scenarios, or orientations (for example, along only a single axis).

Complementary to the control problem is the issue of measuring the state information the controller requires, such as the object's pose, surface friction, and whether the finger is in contact with the object. Touch sensors provide local contact information but are not widely available as a plug-and-play module. The difficulty in using visual sensing is that fingers occlude the object during reorientation. Recent works used RGBD [red, green, and blue (RGB) and depth] cameras to estimate object pose but required a separate

[1]Improbable AI Laboratory, Massachusetts Institute of Technology, Cambridge, MA 02139, USA. [2]Computer Science and Artificial Intelligence Laboratory (CSAIL), Massachusetts Institute of Technology, Cambridge, MA 02139, USA. [3]Institute for Interdisciplinary Information Sciences, Tsinghua University, Beijing 100084, China. [4]Meta AI, Pittsburgh, PA 15213, USA. [5]Institute of Artificial Intelligence and Advanced Interactions (IAIFI), Massachusetts Institute of Technology, Cambridge, MA 02139, USA.
*Corresponding author. Email: pulkitag@mit.edu

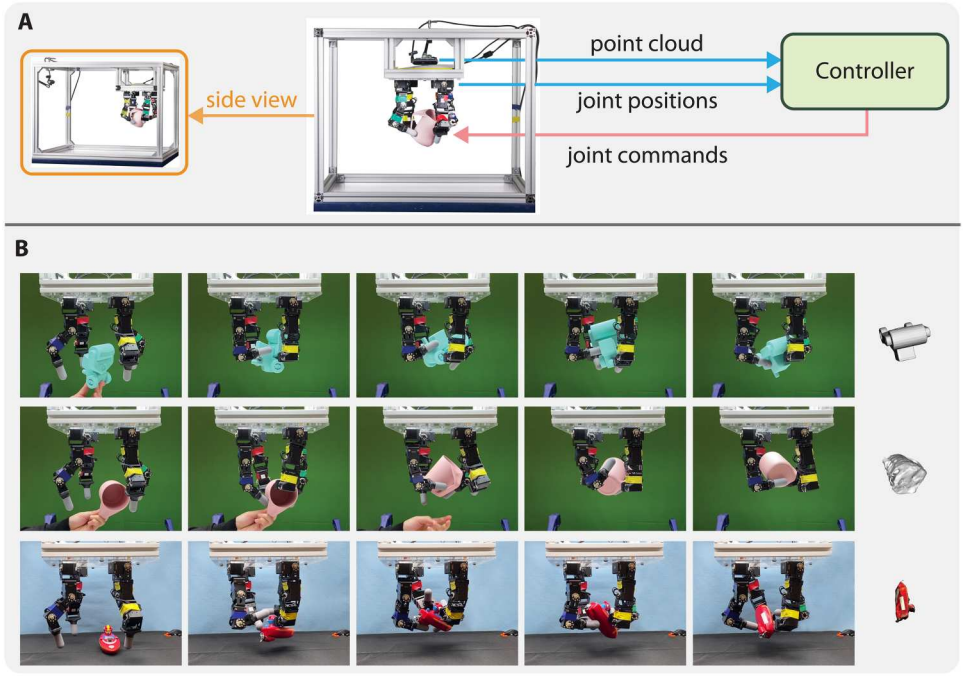


Fig. 1. Illustration of the robot system. (**A**) Front and side views of our real-world setup. The controller is a neural network that uses depth recordings from a single camera along with the joint positions of the manipulator to predict the change in joint positions. (**B**) Visualization of the same 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 (extrinsic dexterity).

pose estimator to be trained per object, which limits their generalization to new object shapes (*8*, *9*, *23*, *26*).

Because of challenges in perception and control, no prior work has demonstrated a real-world–ready reorientation system. Although controlling directly from perception is hard, given the full low-dimensional representation of relevant state information such as the object's position, velocity, pose, and manipulator's proprioceptive state, it is possible to build a controller using deep reinforcement learning (RL) that successfully reorients diverse objects in simulation (*7*). RL effectively leverages large amounts of interaction data to find an approximate solution to the computationally challenging optimization problem of solving for reorientation. However, as a result of requiring large amounts of data and full state information, today, such RL controllers can only be trained in simulation. This leaves at least two open questions: how to train controllers with sensors available in the real world such as visual inputs and whether controllers trained in simulation transfer to the real world (sim-to-real transfer problem).

The difficulty in training RL controllers from visual inputs stems from the learner's need to simultaneously solve the problem of inferring the relevant state information (feature learning) and determining the optimal actions. If the optimal actions were known in advance, it would be simpler to train a model that predicts these actions from visual inputs (supervised learning). Such a two-stage teacher-student training paradigm, where first a control policy is trained via RL with full state information (teacher) and then a second student policy is trained via supervised learning to mimic the teacher, has been successfully used for several applications (*7*, *27*–*30*). We found that the major roadblock in learning a visual policy that works across diverse objects is the slow speed of rendering in simulation, which resulted in training times of more than 20 days with our compute resources. Such slow training makes experimentation infeasible. We devised a two-stage approach for training the vision policy that first uses a synthetic point cloud without the need for rendering and is then fine-tuned with a rendered point cloud to reduce the sim-to-real gap. Our pipeline makes training five times faster. The second consideration was the use of a sparse convolution neural network to represent the policy to process point clouds at the speed required for real-time feedback control (12 Hz in our case). By directly predicting actions from point clouds, our approach bypasses the problem of consistently defining pose/key points across different objects, allowing for generalization to new shapes.

The next challenge is in overcoming the sim-to-real gap. In dynamic in-hand object reorientation, both the robot and the object move quickly. Achieving precise control in a system with fast-changing dynamics is challenging. It becomes even more challenging when using a downward-facing hand because control failures are irreversible. Therefore, dynamic in-hand object reorientation poses a substantial sim-to-real transfer challenge. Some reasons for the sim-to-real gap are differences in motor/object dynamics, perception noise, and modeling approximations made by the simulator. For instance, contact models in fast simulators tend to be a crude approximation of reality, especially for nonconvex objects (*31*). Whether sim-to-real transfer of reorientation controller is even possible for these complex object shapes remained unclear.

The systematic choices of identifying the manipulator dynamics (details in Materials and Methods); domain randomization (*32*); the design of reward function; and the hardware considerations,

including the number of fingers and the fingertip material, reduced 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 [extrinsic dexterity (*33*)] and the harder condition when the object is in the air without any supporting surface. The results show progress toward developing a real-time controller capable of dynamically reorienting new objects with complex shapes and diverse materials by any amount in the full space of rotations [SO(3), special orthogonal group in three dimensions] using inputs from just a single commodity depth camera and joint encoders. Although there is substantial room for improvement, especially in achieving precise reorientation, our results provide evidence that sim-to-real transfer is possible for challenging tasks involving dynamic and contact-rich manipulation in less-structured settings than previously demonstrated.

Last, many prior efforts used custom or expensive manipulators [such as the Shadow Hand (*8*, *9*, *10*) costing more than $100,000] and often relied on sophisticated sensing equipment such as a motion capture system. Such a hardware stack is hard to replicate because of its cost and complexity. In contrast, our hardware setup costs less than $5000 and uses only open-source components, making it easier to replicate. Furthermore, our platform is not specific to object reorientation and can be used for other dexterous manipulation tasks. Because of the low barrier to entry and the evidence that such a system can tackle a challenging manipulation task, our platform can democratize research in dexterous manipulation.

RESULTS

We trained a single controller to reorient 150 objects from an arbitrary initial to a target configuration in simulation. The learned controllers were deployed in the real world on the open-source three-fingered D′Claw manipulator (*34*) and a modified four-fingered version with 9 and 12 degrees of freedom, respectively. The robot's observation is a depth image captured from a single Intel RealSense camera and the proprioceptive state of the fingers. The goal is provided as the point cloud of the object in a target configuration in the SO(3) space. The initial configuration of the object is a random transformation in SE(3) (special Euclidean group in three dimensions) space within the range of the robot's fingers—The object is either set on a table or handed over by a human to the robot.

We experimented with the hand in the downward-facing configuration in two settings: with and without a supporting table. Our system runs in real time at a control frequency of 12 Hz using a commodity workstation. Figure 1 shows the intermediate steps of manipulating three objects to target orientations depicted in the rightmost column. The proposed controller reorients a diverse set of new objects with complex geometries not used for training. Movie 1 provides a short summary of our results with audio. Movie S1 shows our system reorienting many objects and provides a more detailed summary of our major findings. Movie S2 visualizes the setting where the robot is tasked with a sequence of target orientations. In such a scenario, it has to stop when it reaches the current target orientation and then restart to achieve the next target.

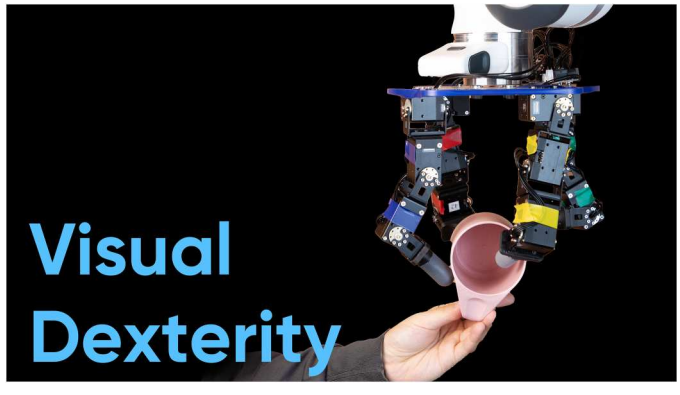
For quantitative evaluation, we used seven objects from the training dataset ($\mathbb{B}$), which we refer to as in distribution, and five objects from the held-out test dataset ($\mathbb{S}$), which we refer to as out of distribution (OOD). Objects are shown in Fig. 2A. We tested each object 20 times with random initial and goal orientation in each testing condition. We three-dimensionally (3D) printed these objects to ensure that the shape of objects in simulation and the real world was identical, which is helpful in evaluating the extent of sim-to-real transfer. Whereas the shape of these seven objects is included in the training set, the surface properties, such as friction, of the real-world objects may not correspond to any object used for training in simulation. Evaluation on five OOD objects tested generalization to shapes. To further showcase generalization to shapes and different material properties, we also present results on some rigid objects from daily life. The orientation errors were measured using an OptiTrack motion capture system that tracks object pose. We defined error as the distance between the goal and the object's orientation when the controller predicts that it has reached the goal and stops. The motion capture was only used for evaluation and was not required by our controller otherwise.

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. Using an external surface to aid reorientation has been referred to as extrinsic dexterity (*33*) and is necessary in many real-world use cases. Visualization of the proposed controller reorienting a diverse set of objects is provided in Fig. 3. To demonstrate the versatility of our system, we present results of the robot manipulating objects of different shapes, materials, and surfaces and using different fingertip materials and varying numbers of fingers.

Reorientation using a three-fingered manipulator with rigid and soft fingertips

With table support, we found three fingers to be 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 Fig. 2E), is shown in Fig. 2B using a violin plot (*35*). Although the overall error distribution is more informative, for ease of comparison, in Table 1, following the success threshold used in previous work (*8*), we report summary statistics of success rate measured as the percentage of tests with error within 0.4 or 0.8 rad. The seven trained objects could be reoriented within an error of 0.4



Movie 1. A dexterous hand facing downward reorients various objects in mid-air.

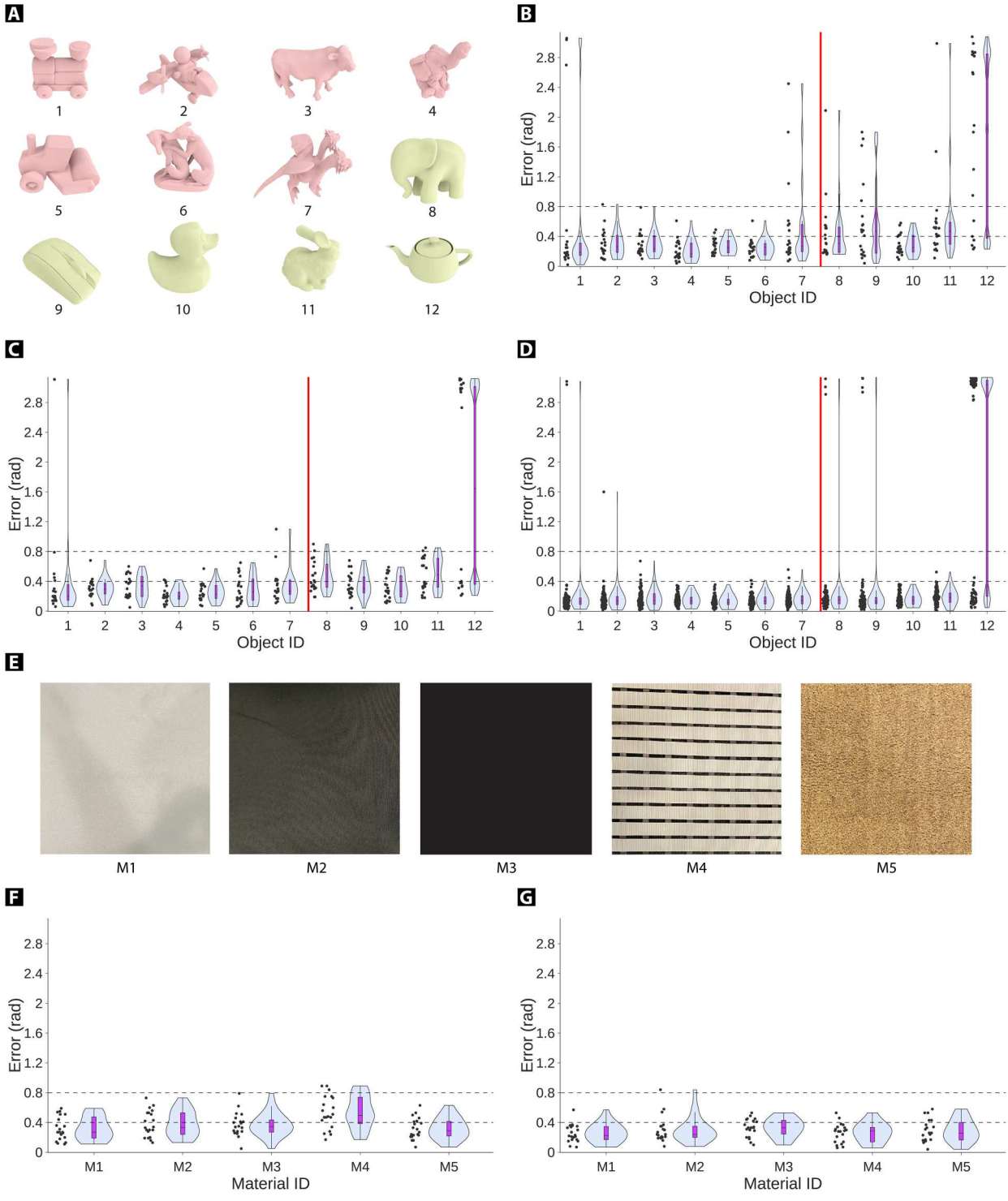


Fig. 2. Experimental results of reorientation. (**A**) Twelve objects with their IDs. The first seven objects are from the training dataset $\mathbb{B}$, and the last five are from the testing dataset $\mathbb{S}$. (**B** and **C**) Real-world error distribution when using rigid and soft fingertips, respectively, on material M1. (**D**) Error distribution in simulation for each object as a violin plot (*35*). The violet rectangle shows the errors within (25 and 75%) percentile, and the horizontal bar in the rectangle depicts the median error. Trained objects can mostly be reoriented within an error of 0.4 rad, with similar performance for rigid and soft fingertips. The error on test objects is higher, and soft fingertips exhibit better generalization. (**E**) Five table materials. (**F** and **G**) Error distributions of different materials for objects #5 and #10, respectively.

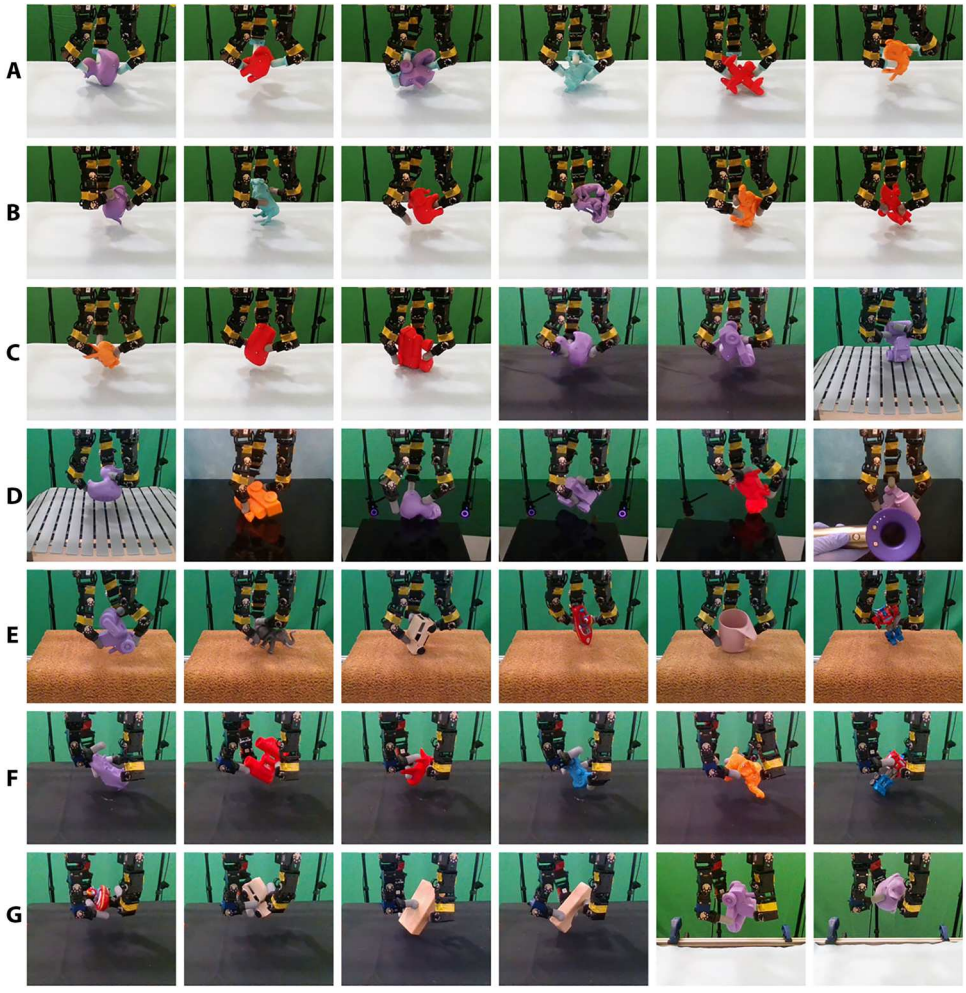


Fig. 3. Different testing scenarios. We tested our controller on objects with diverse shapes and reorientation conditions such as using different supporting surfaces, including a tablecloth, an uneven door mat, a slippery acrylic sheet, and a perforated bath mat. We also evaluated performance using fingertips with different softness: rigid 3D-printed (**A**) and soft elastomer fingertips (**B** to **G**). (A) to (E) use a three-fingered robot hand. (F) and (G) use a four-fingered robot hand. Our policy can reorient real household objects (E and G) and can operate without the need for a supporting surface (in the air) as shown in (G).

rad 81% of the time. For the five OOD test objects, the success rate was lower, at 45%. As expected, the performance is better, with a relaxed error threshold of 0.8 rad and worse at stricter thresholds.

Qualitatively observing the robot behavior revealed that some causes of failure were the object overshooting the target orientation or the finger slipping across the object, especially for OOD objects. One explanation is that rigid hemispherical fingertips contact the object in a very small area (close to making a point contact), which makes small errors in the action commands more pronounced. Further, we found that the fingertip material had low friction, resulting in slips that made manipulation harder. To mitigate these issues, we designed and fabricated soft fingertips that cover the rigid 3D-printed skeleton with a soft elastomer (see fig. S2C). Soft fingertips provide higher friction and deform when contact happens (compliance), increasing the contact area between the finger and the object. The error distribution in Fig. 2C shows that using soft fingers does not affect performance on trained objects but improves generalization to OOD objects. Results in Table 1 confirm the findings: Success rate on OOD objects increased from 45 to 55% when switching from rigid to soft fingertips. Qualitatively, we noticed that soft fingertips behave less aggressively than rigid fingertips, resulting in smoother object motion. We therefore used soft fingertips in the rest of the experiments. Although the controller was trained using a rigid-body simulator, its performance did not degrade when applied to soft fingertips.

The reorientation error can result from imperfect training, sim-to-real gap, generalization gap, or failures at detecting whether the object is at the target orientation, which triggers the controller to stop. In Fig. 2D, we report the error distribution in simulation. Although the trained controller was not perfect in simulation, the errors in simulation followed the same trend as in the real world (Fig. 2C) but were lower, indicating some sim-to-real gap. As shown in Table 1, the performance gap between the simulation and the real world was smaller when using a relaxed error threshold of 0.8 rad compared with a threshold of 0.4 rad. This illustrates the challenge of achieving precise reorientation. For some objects (#1 and #12), the error distribution was bimodal both in simulation and the real world. The test runs with high errors largely resulted

Table 1. Statistics of the orientation error when the hand reorients objects on a table. CI stands for bias-corrected and accelerated (BCa) bootstrap confidence interval. Train stands for testing on the seven objects (Fig. 2A) from the training dataset $\mathbb{B}$. Test stands for testing on the five objects from the testing dataset $\mathbb{S}$.

	With rigid fingertips (real)		With soft fingertips (real)		In simulation	
	Train	Test	Train	Test	Train	Test
<0.4 rad (22.9°)	81%	45%	79%	55%	96%	85%
95% CI	(73%, 90%)	(32%, 58%)	(71%, 86%)	(44%, 62%)	(94%, 97%)	(82%, 88%)
<0.8 rad (45.8°)	95%	75%	98%	86%	98%	87%
95% CI	(88%, 98%)	(46%, 91%)	(96%, 99%)	(58%, 96%)	(97%, 99%)	(84%, 90%)
95% CI of the median of orientation errors (rad)	(0.20, 0.27)	(0.29, 0.46)	(0.21, 0.28)	(0.33, 0.42)	(0.12, 0.13)	(0.15, 0.18)

from incorrect detection of when to stop. For instance, object #12 appeared nearly symmetric in the point cloud representation, which often leads to errors close to 180°. Although it is hard to quantitatively disentangle errors originating from incorrect action prediction and the stopping criterion, on the basis of our experience with the system, we hypothesize that the latter contributes more, which is supported by the analysis in Supplementary Discussion (see the "Discussion on precise manipulation" section).

Object reorientation on different supporting materials

Changing the table surface changes the dynamics of object motion. We tested whether our controller is robust to a diverse set of materials: a rough cloth (M1), a smooth cloth (M2), a slippery acrylic sheet (M3), a bathtub mat with perforations resulting in nonstationary object dynamics depending on the object's position on the mat (M4), and a door mat with uneven texture (M5). The materials have different surface structures, roughness, and friction, leading to different system dynamics. We evaluated with one in-distribution object (object #5) and one OOD object (object #10). Figure 2 (F and G) shows that our controller performed similarly on different supporting materials, demonstrating its robustness.

Toward object reorientation in air

Because the controllers discussed above were trained with a supporting surface, when the supporting surface was removed, the manipulator consistently dropped the object, resulting in failures. Prior work used a specialized training procedure of configuring the object in a good pose at the start of each training episode and a manually designed gravity curriculum (*7*) to learn in-air (without supporting surface) reorientation controllers. Consequently, it was necessary to train separate controllers for reorientation with a supporting surface and in the air. It is preferable to have a single controller capable of in-air reorientation and use the supporting surface, if available, to recover from any dropping failures. We achieved this desideratum by using a four-fingered hand and designing a reward function that penalizes contact between the object and the supporting surface to discourage the controller from using external support for reorientation. When the controller is trained on a supporting surface with the proposed reward function, in-air reorientation emerges.

Although both three- and four-fingered hands can reorient objects on a supporting surface (Fig. 4A), only the four-fingered hand was capable of in-air reorientation (Fig. 4B). We hypothesize this to be the case because, with four fingers, more finger configurations can reorient the object, making it easier for policy optimization to find one solution. Furthermore, we hypothesize that the redundancy in the number of fingers makes the system more robust to errors in action prediction.

SO(3) object reorientation in air

Figure 1B shows how our controller trained in simulation reorients different real-world objects in the air. In-air reorientation can fail if the object is not accurately reoriented or if the robot drops the object. Because in-air reorientation is more challenging, it is possible that the controller is less accurate at reorienting objects. On evaluation with two objects, we found the distribution of orientation error in trials where the objects are not dropped (Fig. 4C) to be similar to reorientation with the supporting surface, indicating that the controller does not lose reorientation precision in the more challenging in-air scenario. In simulation analysis, we did not notice any notable correlation between orientation error and the distance between the initial and target orientations (fig. S12B), indicating that the controller performs similarly in the full SO(3) space.

Our controller performs dynamic reorientation. The median time for manipulation across objects and randomly sampled orientation distances in the full SO(3) space was less than 7 s (Fig. 4D), which makes it a fast in-air reorientation controller operating in the full SO(3) space. Figure 4D also shows that the reorientation times in the real world were longer than in simulation, which we believe is because of real-world contact dynamics being different from simulation.

Simulation analysis revealed that object dropping was the most notable source of errors (fig. S12C). Dropping rates vary substantially across objects. Real-world results followed the same trend. The dropping rate of a shape used in training, the truck (object #5), was 23%, much lower than the dropping rate of 56% for an OOD duck-shaped object (#10). The dropping rate for the duck-shaped object in the simulation was around 20%, showing a sim-to-real gap. However, it remains unclear whether 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. This is because, although the simulation and real-world experiments used the object with the same shape, properties such as surface friction that are critical in reorientation can be different. If an object is curved and has a smooth surface, which is the case with the duck, small differences

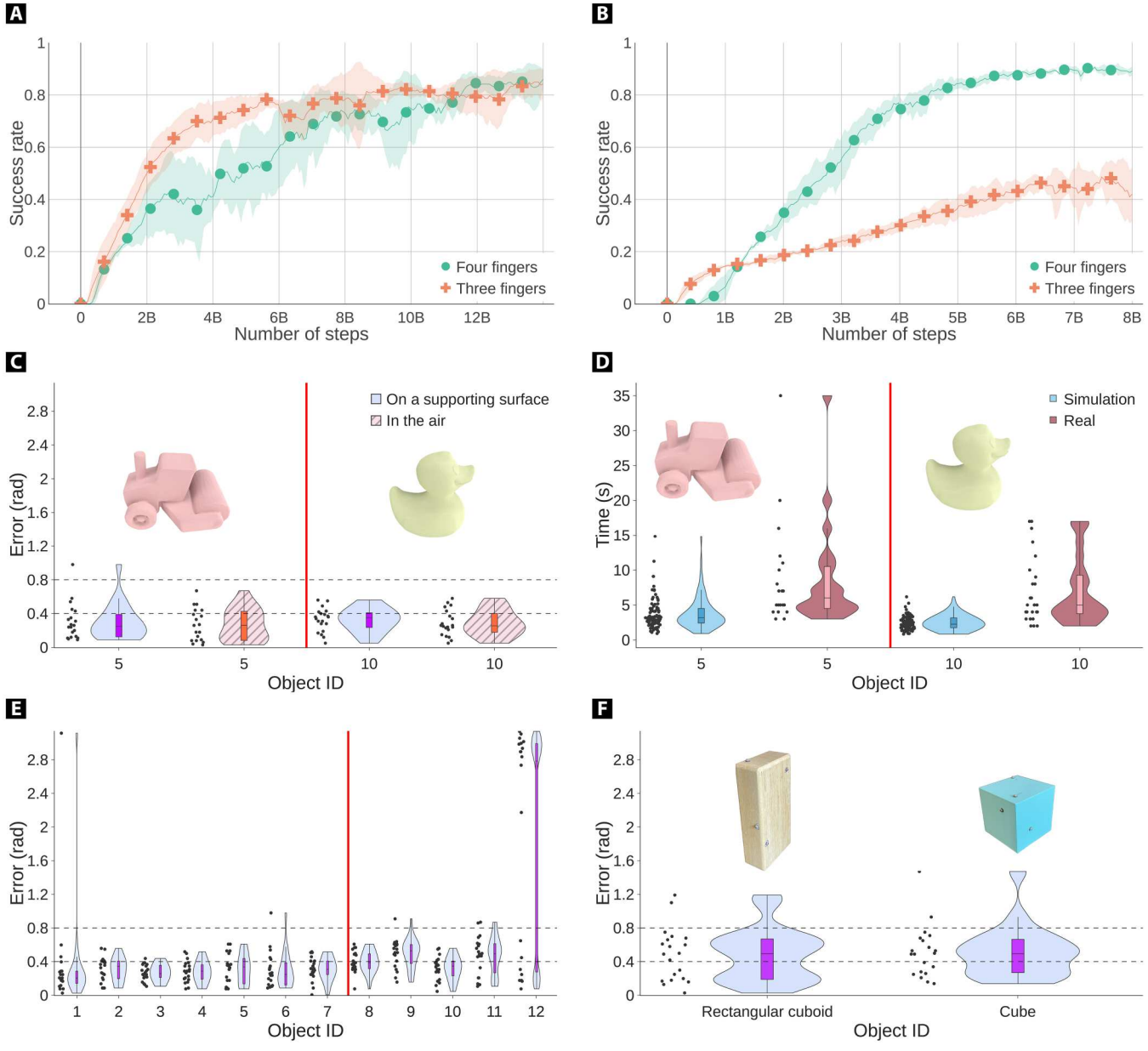


Fig. 4. Benefit and performance of reorientation with a four-fingered hand. (**A**) When training a controller to reorient objects with a supporting surface, the three-fingered and four-fingered hands achieved similar learning performance. (**B**) However, when we incentivized the hands to lift the object during reorientation, the four-fingered hand substantially outperformed the three-fingered hand. (**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 OOD object. The error distribution is similar to that in the case of tabletop reorientation. (**D**) Distribution of the episode time in both simulation and the real world. (**E**) Same controller's performance on 12 objects with a supporting surface. (**F**) We tested the controller on symmetric objects with a supporting surface. The controller behaved reasonably well even though it was never trained with symmetric objects.

in friction can substantially change the task difficulty. We chose to report results on the duck because it was used in prior work (*23*), is among the harder objects to reorient, and thus also highlights the limitations of our controller.

If a table was present below the hand (for example, the setup shown in the third row of Fig. 1B) and the object was dropped, we noticed that our controller picked up the object and continued reorienting—an instance of recovery from failures. It is possible that the reward term encouraging in-air reorientation might hurt on-table reorientation. However, the error distribution for on-table reorientation with the updated reward function (Eq. 6) (Fig. 4E) was similar to earlier on-table experiments. Moreover, although our controller was trained using objects with asymmetry or reflective symmetry, which makes learning much easier, we noticed some generalization to symmetric objects (Fig. 4F, more discussion in Supplementary Discussion). The in-air, on-table, and dropping recovery results demonstrate that it is possible to build a single controller that works across different scenarios.

Qualitatively looking at the reorientation behavior, it might appear that the object is not always moving toward the target orientation. One possibility is that the manipulator randomly moves the object until it gets close to the target orientation by chance and then stops. To rule out this possibility, we provide videos in movie S1 showing that for the same initial but different target orientation, the object motions are different. For the same initial and target orientation, object motions across trials are similar, which would not be the case if the object was randomly being reoriented.

Generalization to objects in daily life

In previous experiments, we used 3D-printed objects for quantitative evaluation. However, real-world objects have varying object dynamics because of differences in material properties, nonuniform mass distribution, and other factors that can vary across the object surface. To test the generalization ability of our controller on such objects, we conducted a qualitative evaluation on a few household objects. Because we did not have the CAD (computer-aided design) models of these objects to generate point clouds in target orientations, we used a free iPad application called Scaniverse to scan the objects. Note that the scan was only required to specify the target orientation, and the scanned object cloud was imperfect (see Fig. 5), resulting in noisy goal specification. Figures 1B and 5 illustrate examples of reorienting such objects. The results illustrate that the controller exhibits a certain degree of robustness against noise in the goal specification and some ability to generalize to new materials and shapes.

Comparison with prior works

Unfortunately, a strictly fair comparison with prior work is not possible because we make fewer assumptions [such as no object-specific pose trackers, reorientation in full SO(3) space, and not being quasi-static], and there are substantial differences in hardware/sensing. Nevertheless, to contextualize our research within the existing literature, we present an approximate comparison with the closest work that reported reorientation results on a duck-shaped object with a downward-facing but under-actuated hand with different morphology and mechanical properties (*23*). The authors reported a success rate of 60% (three of five tests) for reorienting the duck quasi-statically (reorientation time of more than 70 s compared with ~7 s for our controller) to within 0.1 rad, but only in a subset of the SO(3) space (rotation only along two axes). Further, they used a precise object-specific pose tracker (error < 2° or 0.034 rad). If we assume perfect stopping criteria (the agent stops reorientation if the object is within 0.1 rad of the target), then for the duck-shaped object, we achieved a success rate of 71% when dynamically reorienting in the

Fig. 5. Reorientation of real objects. Examples of reorienting real objects that were not 3D printed using a four-fingered and a three-fingered manipulator.

full SO(3) space in simulation. Because of challenges in setting up precise stopping in the real world, we could not run these evaluations in the real world. Even if we did, the differences in material properties between the duck used by us and prior research (*23*) would make the comparison unfair. Comparing our simulation and their real-world results is also unfair. However, the results indicate that with more assumptions, such as the precise stopping criterion, the performance of our system improves. Improving the precision of our system without any additional assumptions is an exciting avenue for future research.

The differences in experimental setups with other prior works (*8*, *9*, *17*, *25*) and concurrent work (*36*) are even larger. For instance, OpenAI's work (*8*) reported results on reorientation with a single object (no generalization) with a simple shape (cube), an upward-facing hand, and an extensive sensing system consisting of three RGB cameras, a motion capture system, and a different hand. Moreover, their success criterion was the number of times an object passes through a target pose, and they never trained their controller to stop the object at the target pose, which we experimentally found harder to learn. In the broader context of manipulation, the ability to stop at the target pose is vital: If the robot uses a tool, it must reorient it to the desired pose and hold the tool in that pose.

The focus of our work is not to increase the reorientation performance on a single object; rather, our work expands the scope of object reorientation to operate in more general and pragmatic settings. The result is a single controller for reorienting multiple objects, evidence of some generalization to new objects, and dynamic reorientation in the air without a highly specialized perception system. At the same time, there remains ample scope for improving performance, and we hope that our conscious use of open-source hardware, commodity sensing, computing, and fast-learning framework (Figs. 6 and 7) will facilitate future research in enhancing performance and comparing results.

DISCUSSION

Solving contact-rich tasks typically requires optimizing the location at which the robotic manipulator contacts the object (*4*, *37*, *38*). One would assume that predicting the contact location requires knowledge of the object's shape. However, inputs to the teacher policy have no information about object shape, yet the teacher policy could reorient diverse and new objects. One possibility is that the agent gathers shape information by integrating information across the sequence of touches made by the fingers. However, the teacher policy is not recurrent, ruling out this possibility. The unexpected observation of reorientation without knowledge of shape was made by earlier work in the context of a reorientation system in simulation (*7*). However, because real-world results were not demonstrated, it remained unclear whether such an observation was an artifact of the simulator or the property of the reorientation problem. With real-world evaluation, we have more confidence that shape information may not be as critical to object reorientation as one might think a priori. However, this is not to suggest that shape is not useful at all. The results show that one can go quite far without shape information, but the performance, especially on precise manipulation and in generalization to new shapes, can likely be improved by incorporating shape features into the teacher policy, an exciting direction for future research.

Typically, having more fingers introduces more optimization variables, making the optimization problem harder in the conventional view. However, we have some evidence to the contrary (Fig. 4B). Having more fingers can make it easier for deep RL to find a solution, especially in challenging manipulation scenarios such as in the air, similar to how over-parameterized deep networks find better solutions (a conjecture). We conjecture that over-parameterized hardware results in a larger pool of good solutions (more ways to reorient an object with more fingers), making it easier for current optimizers in deep learning to find a good solution.

In designing the proposed system, we either devised or made several technical choices: two-stage student training, representing both the camera recordings and proprioceptive readings as a point cloud, sparse convolution neural network for real-time control, limited range of domain randomization due to system identification, system identification using parallel simulation on graphics processing unit (GPU), use of soft material on fingertips, and using a larger number of fingers instead of the conventional wisdom of using fewer fingers. These choices, however, are not specific to in-hand reorientation but can be applied to a broad spectrum of vision-based manipulation tasks involving rigid bodies. We hope that the knowledge of these choices, along with a low-cost platform, can further the goal of democratizing research in dexterous manipulation.

Limitations and possible extensions

Object reorientation with a downward-facing hand has notable room for improving precision and reducing the drop rate. We hypothesize that one possible cause for dropping objects is that the control frequency of 12 Hz is not fast enough. The robot dynamically manipulates the object, and it takes a fraction of a second to lose control. It might be challenging to determine when the object is slipping from the fingers in real time using visual feedback at 12 Hz. Feedback control at a higher frequency may mitigate such failures but requires either more efficient neural network architectures or more processing power.

Another hypothesis for object dropping is missing information regarding whether the finger is in contact with the object, whether the object is slipping, or how much force is being applied. We conjecture that explicit knowledge of contact, contact force, and other signals such as slip can substantially improve performance. Currently, the robot relies purely on occluded vision observations to infer contacts. Augmenting the robot's observation with touch sensors is therefore an exciting direction for future investigation.

We also found that inaccurate prediction of rotational distance is another cause for imprecise object reorientation. The prediction of rotational distance is less accurate when the actual rotational distance is less than 0.4 rad (see the "Discussion on precise manipulation" section in Supplementary Discussion).

We hypothesize that generalization and precision can be improved by training on a larger object dataset, investigating RGB sensing to complement depth sensing to capture fine geometric structures and reduce noise, and integrating visual and tactile sensing to obtain more complete point clouds. Further, there remains a sim-to-real gap that future research should investigate.

We used D'Claw manipulators in this work because they are open source and low cost. However, many aspects of the D'Claw, such as the finger design and the number of fingers, are suboptimal. For instance, although we observed some robustness to the softness

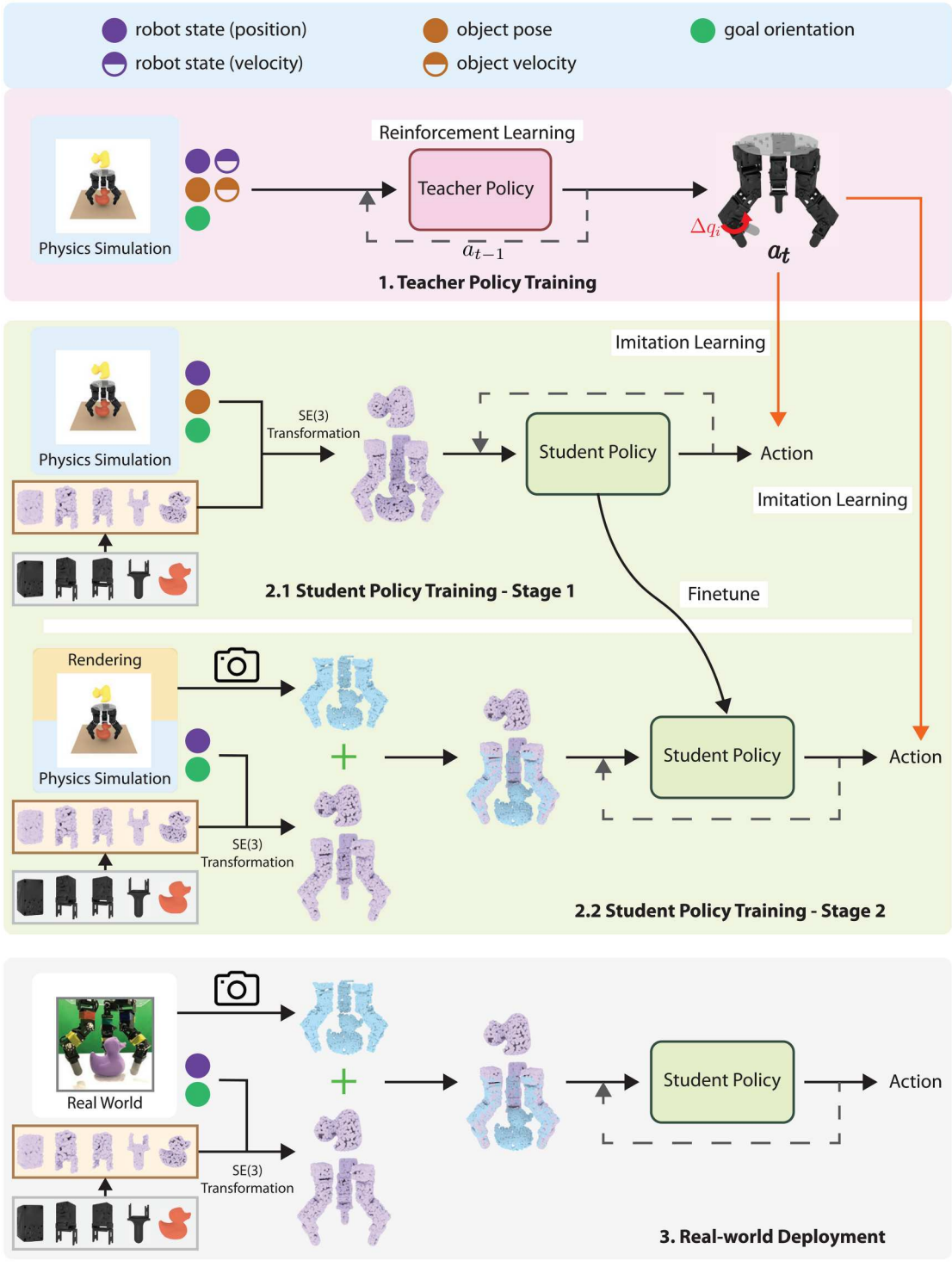


Fig. 6. Teacher and two-stage student training framework. First, a teacher policy was trained using RL with privileged state information. Then, a student policy was trained to imitate the teacher using synthetic and complete point clouds as input. The student policy was further fine-tuned using rendered point clouds. During deployment, the student policy can be directly used to control real robots.

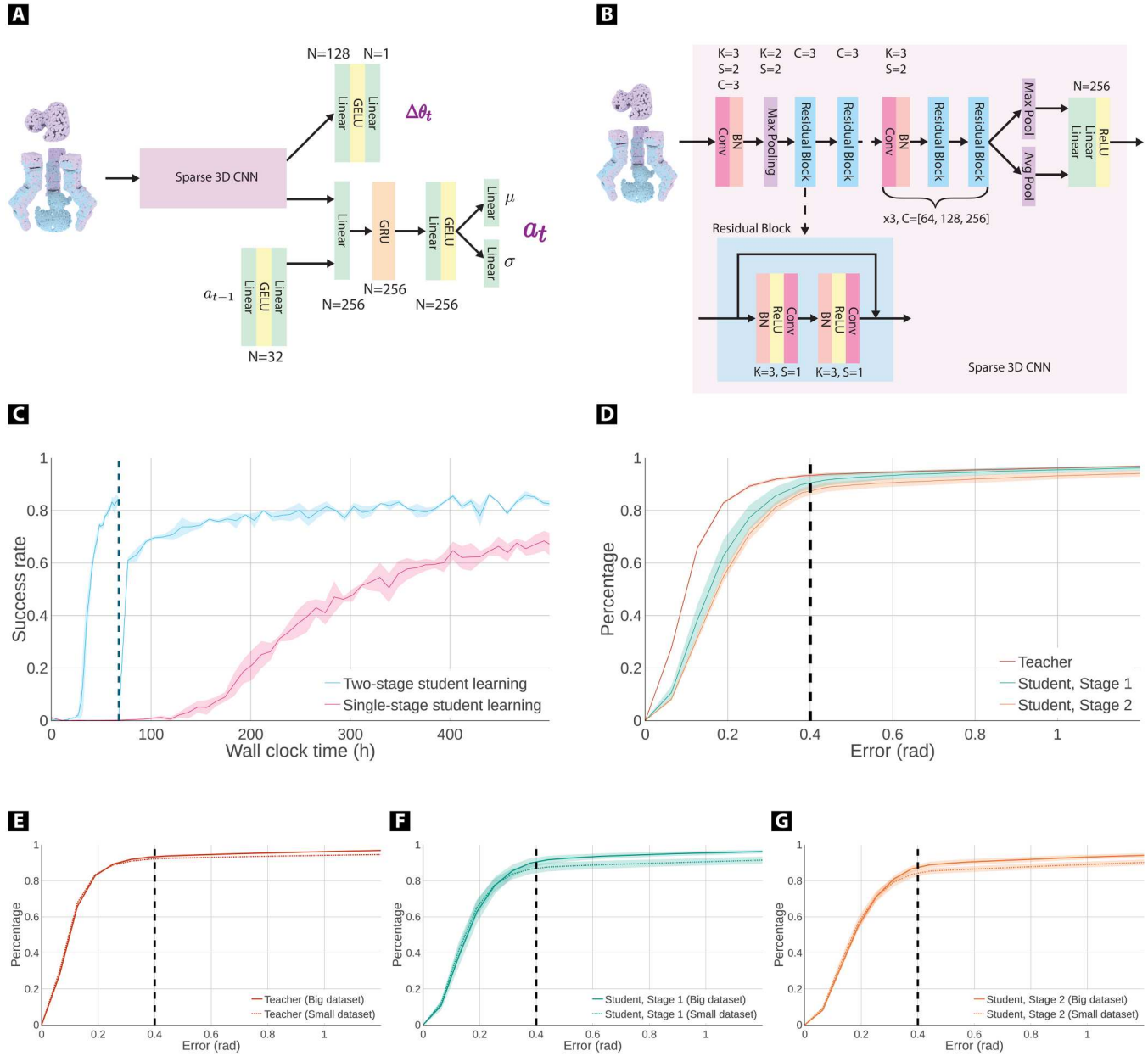


Fig. 7. Student policy learning. (**A**) Student vision policy network architecture. (**B**) Sparse 3D CNN (convolutional neural network) component of the policy network. (**C**) Proposed two-stage student learning learns faster than single-stage student learning. The dashed vertical line denotes the transition from the first to the second stage of student learning. The performance dip happens because of a change in the distribution of point cloud inputs from being unoccluded in the first stage to being occluded in the second. (**D**) Post-training evaluation of teacher and student policies on the training dataset $\mathbb{B}$. For each object, the initial and target orientations were randomly sampled 50 times, resulting in 7500 samples. The empirical cumulative distribution function (ECDF) of the orientation error is plotted. The results show that the students are close to the teacher's performance. (**E** to **G**) Comparing the ECDFs of the policies being evaluated on dataset $\mathbb{B}$ and dataset $\mathbb{S}$ reveals a small generalization gap for all the policies. In (C to G), the shaded area shows the total area within one SD of the mean, based on running each experiment with three different random seeds.

of fingertips, different softness and skeleton designs can notably affect the longevity of fingertips. We manually iterated over many soft fingertip designs, which was time consuming. Similarly, the fingertips have a hemispherical shape, quite different from humans and presumably not optimal. The performance of the task can be improved by better hardware design: the shape of fingers, the degrees of actuation on each finger, the placement of fingers, and the choice of materials. Manually iterating over these choices is infeasible. A promising future direction is to use a computational approach for automatically designing the hand for specific tasks (*39*).

In summary, we present a real-time controller that can dynamically reorient complex and new objects by any desired amount using a single depth camera. The system is both simple and affordable, which aligns with the objective of making dexterous manipulation research accessible to a wider audience.

MATERIALS AND METHODS

Given a random object in a random initial pose, the robot was tasked to reorient the object to a user-provided target orientation in SO(3) space. We trained a single vision–based object reorientation controller (or policy) in simulation to reorient hundreds of objects. The controller trained in simulation was directly deployed in the real world (zero-shot transfer). The choices in our experimental setup were made to support future deployment of reorientation in service of tool use and on a mobile manipulator.

Object datasets

We used two object datasets in this work: big dataset ($\mathbb{B}$) and small dataset ($\mathbb{S}$). $\mathbb{B}$ contains 150 objects from internet sources. $\mathbb{S}$ contains 12 objects from the ContactDB (*40*) dataset. These two datasets do not have overlapping shapes. More details on the object dataset are in Supplementary Methods.

Simulation setup

We used Isaac Gym (*41*) as the rigid-body physics simulator. We trained all the policies on a tabletop setup: Hands face downward with a supporting table.

Success criteria

During training, the success criterion for reorienting an object acted both as a reward signal and as a criterion for success to end the episode. A straightforward success criterion is judging whether an object′s orientation is close to the target orientation (orientation criterion). However, a controller trained using this criterion tends to cause the object to oscillate around the target orientation. To address this issue, the success criterion was expanded to explicitly penalize finger and object movements. For further details on how we designed the success criteria for training, please refer to Supplementary Methods.

Training the visuomotor policy

We modeled the problem of learning the controller, π, as a finite-horizon discrete-time decision process with horizon length T. The policy π takes as input sensory observations ($\boldsymbol{o}_t$) and outputs action commands ($\boldsymbol{a}_t$) at every time step t. Learning π using RL is data inefficient when the observation ($\boldsymbol{o}_t$) is high dimensional (for example, point clouds). The reason is that the policy needs to simultaneously learn which features to extract from visual observations and what the high-rewarding actions are. The problem would be simplified if one of these factors were known: Learning a policy via RL from sufficient state information would be much easier than direct learning from sensory observations. Similarly, a priori knowledge of high-rewarding actions would reduce the data requirements of learning from visual observations.

Prior work has used this intuition to ease policy learning by decomposing the learning process into two steps (*7*, *27*, *28*, *30*). In the first step, a teacher policy is trained in simulation with RL using low-dimensional state space that includes privileged information. In the case of in-hand object reorientation, privileged information includes quantities such as fingertip velocity, object pose, and object velocity that can be directly accessed from the simulator but can be challenging to measure in the real world. Because the teacher policy operates from a low-dimensional state space, it can be more efficiently trained using RL. Next, to enable operation in the real world, one can either train a perception system to predict the privileged information (*8*, *26*) or train a second student policy to predict high-rewarding teacher actions from raw sensory observations via supervised learning (*7*, *27*, *28*, *30*).

An underlying assumption of the two-stage training paradigm is that a low-dimensional state for learning a teacher policy can be identified. Because there are no tools available to theoretically analyze whether a particular choice of state space is sufficient for policy learning, selecting the state inputs for the teacher policy is a manual process based on human intuition. At first, object reorientation might seem to require knowledge of object shape because the controller must reason about where to make contact. If object shape is necessary, then it will not be possible to reduce depth observations into a low-dimensional state. However, past work found that even without any shape information, it is possible to train RL policies to achieve good reorientation performance on a diverse set of objects in simulation (*7*). Therefore, teacher-student training can be leveraged to simplify the learning of object reorientation.

To deploy the policy in the real world, some prior works trained a perception system to predict the object pose (*8*, *9*). However, object pose is only defined with respect to a particular reference frame. Choosing a common frame of reference across different objects is not possible. As a consequence, pose estimators cannot generalize across objects. Therefore, we chose to train an end-to-end student policy that takes as input the raw sensory observations and is optimized to match the actions predicted by the teacher policy via supervised learning (*42*). Because supervised learning is considerably more data efficient than RL, such an approach solves the hard problem of learning a policy from raw sensory observations.

The teacher-student training paradigm has been used to learn object reorientation policy in simulation from visual and proprioceptive observations (*7*). However, a separate policy was trained per object. Second, 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 (Fig. 6) 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: RL with privileged information

The learning of teacher policy ($\pi^{\mathcal{E}}$) is formulated as a RL problem where the robot observes the current observation ($\mathbf{o}_t^{\mathcal{E}}$), takes an action ($\boldsymbol{a}_t$), and receives a reward (r_t) afterward. A single policy ($\pi^{\mathcal{E}}$) is trained across multiple objects using proximal policy optimization (*43*) to maximize the expected discounted episodic return: $\pi^{\mathcal{E}*} = \arg\max_{\pi^{\mathcal{E}}} \mathbb{E}\left[\sum_{t=0}^{T-1} \gamma^t r_t\right]$. Because the observation $\boldsymbol{o}_t$ at a single time step t does not convey the full state information such as the geometric shape of an object, our setup is an instance of partially observable Markov decision process. However, for the sake of simplicity and based on the finding that knowledge of object shape may not be critical as discussed above, we chose to model the policy as a Markov decision process: $\boldsymbol{a}_t = \pi^{\mathcal{E}}(\boldsymbol{o}_t; \boldsymbol{a}_{t-1})$. The policy also takes as input the previous action ($\mathbf{a}_{t-1}$) to encourage smooth control.

Observation space

The inputs to the teacher policy, $\boldsymbol{o}_t$, include proprioceptive state information, object state, and target orientation. Details are shown in Supplementary Methods.

Action space

We used position controllers to actuate the robot joints at a frequency of 12 Hz. The policy outputs the relative joint position changes $\mathbf{a}_t \in \mathbb{R}^{3G}$. Instead of directly using $\boldsymbol{a}_t$, we used the exponential moving average of actions $\bar{\mathbf{a}}_t = \alpha \mathbf{a}_t + (1-\alpha)\bar{\mathbf{a}}_{t-1}$ for smooth control, where $\alpha \in [0,1]$ is a smoothing coefficient. In our experiments, we set $\alpha = 0.8$. Given the smoothed action $\bar{\mathbf{a}}_t$, the target joint position at the next time step is $\mathbf{q}_{t+1}^{tgt} = \mathbf{q}_t + \bar{\mathbf{a}}_t$.

Reward

We first describe the reward function for the hand to reorient objects on a table. The first term in the reward function (Eq. 1) is the success criterion for the task. However, because this only provides sparse reward supervision, the criterion by itself is insufficient for successful learning. Therefore, we added additional reward shaping (*44*) terms to encourage reorientation. We used a dense reward term that encourages minimization of the distance ($\Delta\theta_t$) between the agent's current and target orientation (Eq. 2). We penalized the agent for moving fingertips far away from the object (Eq. 3). Without this term, fingers barely made any contact with the object during training. We also penalized the agent for expending energy (Eq. 4) and for pushing the object too far from the robot's hand (Eq. 5), in which case the episode was also terminated. The reward terms are mathematically expressed as

$$r_{1t} = c_1(\text{Task successful}) \quad \text{sparse task reward} \tag{1}$$

$$+c_2 \frac{1}{|\Delta\theta_t| + \epsilon_\theta} \quad \text{dense task reward} \tag{2}$$

$$+c_3 \sum_{i=1}^{G} \|\mathbf{p}_t^{f_i} - \mathbf{p}_t^{o}\|_2^2 \quad \text{keep fingertip close to the object} \tag{3}$$

$$+c_4 |\dot{\mathbf{q}}_t|^T |\tau_t| \quad \text{energy reward} \tag{4}$$

$$+c_5 \left(\|\mathbf{p}_t^{o}\|_2^2 > \bar{p}\right) \quad \text{penalty for pushing the object away} \tag{5}$$

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

Using the aforementioned reward function, we were able to train reorientation policies that used the support of the table. Next, to enable the more challenging behavior of reorienting objects in the air, we added a penalty for the contact between the object and table (Eq. 7) and a penalty for using the penultimate joint instead of the fingertip for reorientation (Eq. 8). Although the term in Eq. 8 is not critical, it results in more natural-looking behaviors. The overall reward function is

$$r_{2t} = r_{1t} \tag{6}$$

$$+c_6 \text{ (object contacts with the table)} \tag{7}$$

$$+c_7 \sum_{i=1}^{N} \left(p_{t,z}^{f_i} > \bar{p}_z\right) \tag{8}$$

where c_6, $c_7 < 0$ are coefficients.

Student policy—Imitation learning from depth observations

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

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 $\mathbf{P}^g$. In other words, the policy sees how the object should look in the end (Fig. 7A, top left).

Observation space

The input to π^S is the point cloud $\mathbf{P}_t = \boldsymbol{P}_t^s \cup \mathbf{P}^g$ (see Fig. 7A). We also did an ablation study on different ways to process the goal point cloud in the Supplementary Discussion section "Using a different encoder for goal." The results show that merging $\boldsymbol{P}_t^s$ and $\mathbf{P}^g$ before they are input to the network leads to faster learning.

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 to process point cloud ($\mathbf{P}_t$) using the Minkowski Engine (see Fig. 7A) (*45*). Compared with the architecture used in (*7*), our convolutional network has a higher capacity to make it possible to learn the reorientation of multiple objects. Without direct access to object velocity, it is necessary to integrate temporal information in π^s, for which we used the gated recurrent unit (*46*) in the network.

Optimization

The student policy π^S was trained using DAGGER (*42*) to imitate the teacher policy π^ε.

Need for two-stage student learning

We found training a vision policy in simulation to be slow, consuming more than 20 days on an NVIDIA V100 GPU (Fig. 7C). The main reason for slow training is that the simulator performs rendering to generate a point cloud that 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 (no occlusions), which makes training easier. The vision policy (π_1^S) can be trained with the synthetic point cloud in less than 3 days, which is a sevenfold speedup compared with training with the rendered point cloud (stage 1; see Fig. 7C). However, the policy, π_1^S, cannot be deployed in the real world because it operates on an idealized point cloud (no occlusions). Therefore, once the student reached high performance, we initiated stage 2, where the policy was fine-tuned with the rendered point cloud. Such fine-tuning is quick in wall-clock time (around 1 day), and the resulting policy (π_2^S) performed better than training from scratch with rendered point clouds (see Fig. 7C). It is possible to further reduce the training time of the student policy by using visual pretraining with passive data that we discuss in the Supplementary Discussion section "Stage 0: speeding up vision policy training with visual pretraining." An additional benefit of the two-stage student policy training is that π_1^S is agnostic to the camera pose. Therefore, a policy from a new viewpoint (π_2^S) can be quickly obtained by fine-tuning using rendered point clouds from that camera pose. Training the vision policy from scratch is not necessary.

Stage 1: Details of synthetic point cloud

In stage 1, the simulation is not used for rendering but only for physics simulation. We generated 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 $\mathrm{P}^{l_j} \in \mathbb{R}^{K\times 3}$. Given link orientation ($\boldsymbol{R}_t^{l_j} \in \mathbb{R}^{3\times 3}$) and position ($\mathbf{p}_t^{l_j} \in \mathbb{R}^{3\times 1}$) at time step t, the point cloud can be computed in the global frame, $\mathbf{P}_t^{l_j} = \mathbf{P}^{l_j}(\mathbf{R}_t^{l_j})^T + (\mathbf{p}_t^{l_j})^T$. 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: $\mathbf{P}_t^s = \bigcup_{j=1}^{j=M} \mathbf{P}_t^{l_j}$, where M is the total number of links (bodies) in the environment. The point cloud $\mathbf{P}_t^s$ can be efficiently generated using matrix multiplication.

Stage 2: Details of rendered point cloud

In stage 2, at each time step, we acquired depth images from the simulator and converted them into point clouds (which we call exteroceptive point clouds) using the camera's intrinsic and extrinsic matrices. Note that such a point cloud is incomplete because of occlusions. We also converted the joint angle information into poses of the links on the robot hand via forward kinematics and then generated the complete point cloud of the robot (which we call a proprioceptive point cloud). Note that such a proprioceptive point cloud of a robot can be easily obtained in the real world in real time from the joint position readings. The policy input is the union of the exteroceptive and the proprioceptive point cloud.

Reducing the simulation to reality gap

There are two main sources of the gap between simulation and reality. The first one is the 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. The other source is the perception gap caused by differences in statistics of sensor readings and/or noise. One way to reduce these gaps is to train a single policy across many different settings of the simulation parameters [domain randomization (*32*)]. The success of domain randomization hinges on the hope that the real world is well approximated by one of the many simulation parameter settings used during training. The chances of such a match increase by randomizing parameters over a larger range. However, excessive randomization may result in an overly conservative policy with low performance (*47*). Therefore, we made design choices that reduce the need for domain randomization and used it only when needed.

The perception gap is reduced by using only depth readings, which are more similar between simulation and reality than RGB. To account for noisy depth sensing, we added noise to the simulated point cloud. The dynamics gap can be reduced by identifying simulation parameters closest to the real world. Although such identification is possible for the robotic manipulator, it is infeasible for object dynamics that vary in material and mass distribution. Therefore, we performed system identification on the robot dynamics and used only small randomization to account for unmodeled errors. We used a larger range of domain randomization on the object and environment dynamics. To make the policy more robust to unmodeled real-world physics, we applied random forces on the object during training, which pressures the policy to reorient objects while being robust to external disturbance. Last, to increase compliance and friction between the object and the manipulator, we used soft fingertips. Such a choice makes the system more tolerant of errors in control commands. Empirically, we noticed that soft fingertips make the robot less aggressive and reduced overshoot.

Identification of robot dynamics

We built the Unified Robot Description Format (URDF) model for the manipulator using its CAD model, which provides accurate kinematics parameters, but the dynamics parameters, such as joint damping and stiffness, must be estimated. One way of identifying dynamics parameters is to leverage the equations of motion (or the dynamics model) and solve for the unknown variables using a dataset of motion trajectories. The Isaac Gym simulator has a built-in dynamics model. However, because the simulator's code is not open source, we did not have access to the precise dynamics model or the gradients of dynamics parameters. We, therefore, used a black-box approach that leverages the ability of Isaac Gym to perform massively parallel simulations. We spawned many simulations with different dynamics parameters and used the one that has the closest match to the real robot's motion.

Let $\boldsymbol{\lambda}_i \in \boldsymbol{\Lambda}$ denote the dynamics parameters 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 computed the score: $h\left(\boldsymbol{q}_A^{C^{\text{real}}}(\cdot), \boldsymbol{q}_A^{C^{\boldsymbol{\lambda}_i}}(\cdot)\right) = -\|\boldsymbol{q}_A^{C^{\text{real}}}(\cdot) - \boldsymbol{q}_A^{C^{\boldsymbol{\lambda}_i}}(\cdot)\|_2^2$, where $\boldsymbol{q}_A^C(\cdot)$ represents the joint position trajectories of a robot C given action commands $\mathbf{A}(\cdot)$, which are detailed in the Supplementary Methods. The closer the motion of the simulated robot is to that of the real world, the higher the score will be. We used the black-box optimization method of Covariance Matrix Adaptation Evolution Strategy (CMA-ES) (*48*), an instance of evolutionary search

algorithms, to determine the optimal dynamics parameter: $\boldsymbol{\lambda}^* = \arg\max_{\boldsymbol{\lambda} \in \Lambda} h\left(\boldsymbol{q}_A^{c^{\text{real}}}(\cdot), \boldsymbol{q}_A^{C^{\boldsymbol{\lambda}}}(\cdot)\right)$. Note that it might be impossible to find a simulated robot that exactly matches the real robot because of the approximate parameterization of real-world dynamics in simulation and the stochasticity in the real-world resulting from actuation/sensing noise. More details on the identification are in the Supplementary Methods.

Real-world deployment

Real-world observation

The real-world observation includes the joint positions of each motor in the manipulator and the depth image from a RealSense camera. Details of how the joint positions and depth image were converted into a unified point cloud input can be found in the Supplementary Methods.

Stopping criteria

To automatically stop the robot, we trained a predictor that reuses features from the policy network to predict $|\Delta\theta_t|$ (see Fig. 7A). The robot is stopped when $\Delta\theta_t^{\text{pred}} < \overline{\theta}$ and $\|\mathbf{a}_t\| < \overline{a}$. More details on the stopping criteria, the real-world experimental setup, and the procedure for quantitative evaluation are in the Supplementary Methods.

Supplementary Materials

This PDF file includes:

Methods
Discussion
Figs. S1 to S14
Tables S1 to S3
References (*49*–*56*)

Other Supplementary Material for this manuscript includes the following:

Movies S1 and S2
MDAR Reproducibility Checklist

REFERENCES AND NOTES

1. M. T. Mason, J. K. Salisbury, J. K. Parker, *Robot Hands and the Mechanics of Manipulation* (MIT Press, 1989).
2. J. K. Salisbury, J. J. Craig, Articulated hands force control and kinematic issues. *Int. J. Rob. Res.* **1**, 4–17 (1982).
3. D. Rus, In-hand dexterous manipulation of piecewise-smooth 3-D objects. *Int. J. Rob. Res.* **18**, 355–381 (1999).
4. I. Mordatch, Z. Popović, E. Todorov, Contact-invariant optimization for hand manipulation, in *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (ACM, 2012), pp. 137–144.
5. Y. Bai, C. K. Liu, Dexterous manipulation using both palm and fingers. *IEEE Int. Conf. Robot.* **39**, 1560–1565 (2014).
6. V. Kumar, Y. Tassa, T. Erez, E. Todorov, Real-time behaviour synthesis for dynamic hand-manipulation. *IEEE Int. Conf. Robot.*, 6808–6815 (2014).
7. T. Chen, J. Xu, P. Agrawal, A system for general in-hand object reorientation. *Conf. Robot Learning* **164**, 297–307 (2022).
8. O. 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, W. Zaremba, Learning dexterous in-hand manipulation. *Int. J. Robot. Res.* **39**, 3–20 (2020).
9. Open AI, I. Akkaya, M. Andrychowicz, M. Chociej, M. Litwin, B. M. Grew, A. Petron, A. Paino, M. Plappert, G. Powell, R. Ribas, J. Schneider, N. Tezak, J. Tworek, P. Welinder, L. Weng, Q. Yuan, W. Zaremba, L. Zhang, Solving rubik's cube with a robot hand. arXiv:1910.07113 [cs.LG] (16 October 2019).
10. A. Nagabandi, K. Konolige, S. Levine, V. Kumar, Deep dynamics models for learning dexterous manipulation, in *Conference on Robot Learning* (ML Research Press, 2020), pp. 1101–1112.
11. N. Furukawa, A. Namiki, S. Taku, 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* (IEEE, 2006), pp. 181–187.
12. T. Ishihara, A. Namiki, M. Ishikawa, M. Shimojo, Dynamic pen spinning using a high-speed multifingered hand with high-speed tactile sensor, in *6th IEEE-RAS International Conference on Humanoid Robots* (IEEE, 2006), pp. 258–263.
13. V. Kumar, A. Gupta, E. Todorov, and S. Levine, Learning dexterous manipulation policies from experience and imitation. arXiv:1611.05095 [cs.LG] (15 November 2016).
14. B. Calli, A. M. Dollar, Vision-based model predictive control for within-hand precision manipulation with underactuated grippers, in *2017 IEEE International Conference on Robotics and Automation (ICRA)* (IEEE, 2017), pp. 2839–2845.
15. B. Sundaralingam, T. Hermans, Relaxed-rigidity constraints: Kinematic trajectory optimization and collision avoidance for in-grasp manipulation. *Auton. Robots* **43**, 469–483 (2019).
16. H. Van Hoof, T. Hermans, G. Neumann, J. Peters, Learning robot in-hand manipulation with tactile features, in *2015 IEEE-RAS 15th International Conference on Humanoid Robots (Humanoids)* (IEEE, 2015), pp. 121–127.
17. S. Abondance, C. B. Teeple, R. J. Wood, A dexterous soft robotic hand for delicate in-hand manipulation. *IEEE Robot. Autom. Lett.* **5**, 5502–5509 (2020).
18. A. Bhatt, A. Sieler, S. Puhlmann, O. Brock, Surprisingly robust in-hand manipulation: An empirical study. *Robot. Sci. Syst.*, (2021).
19. B. Calli, A. Kimmel, K. Hang, K. Bekris, A. Dollar, Path planning for within-hand manipulation over learned representations of safe states, in *International Symposium on Experimental Robotics* (Springer, 2020), pp. 437–447.
20. H. Zhu, A. Gupta, A. Rajeswaran, S. Levine, V. Kumar, Dexterous manipulation with deep reinforcement learning: Efficient, general, and low-cost, in *2019 International Conference on Robotics and Automation (ICRA)* (IEEE, 2019), pp. 3651–3657.
21. A. Rajeswaran, V. Kumar, A. Gupta, G. Vezzani, J. Schulman, E. Todorov, S. Levine, Learning complex dexterous manipulation with deep reinforcement learning and demonstrations. arXiv:1709.10087 [cs.LG] (18 September 2017).
22. G. Khandate, M. Haas-Heger, M. Ciocarlie, On the feasibility of learning finger-gaiting in-hand manipulation with intrinsic sensing, in *2022 International Conference on Robotics and Automation (ICRA)* (IEEE, 2022), pp. 2752–2758.
23. A. S. Morgan, K. Hang, B. Wen, K. Bekris, A. M. Dollar, Complex in-hand manipulation via compliance-enabled finger gaiting and multi-modal planning. *IEEE Robot. Autom. Lett.* **7**, 4821–4828 (2022).
24. R. Jeong, J. T. Springenberg, J. Kay, D. Zheng, Y. Zhou, A. Galashov, N. Heess, F. Nori, Learning dexterous manipulation from suboptimal experts, in *Conference on Robot Learning* (ML Research Press, 2020), pp. 915–934.
25. L. Sievers, J. Pitz, B. Bäuml, Learning purely tactile in-hand manipulation with a torque-controlled hand, in *2022 International Conference on Robotics and Automation (ICRA)* (IEEE, 2022), pp. 2745–2751.
26. A. Allshire, M. Mittal, V. Lodaya, V. Makoviychuk, D. Makoviichuk, F. Widmaier, M. Wüthrich, S. Bauer, A. Handa, A. Garg, Transferring dexterous manipulation from gpu simulation to a remote real-world trifinger, in *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (IEEE, 2022), pp. 11,802–11,809.
27. D. Chen, B. Zhou, V. Koltun, and P. Krähenbühl, Learning by cheating, in *Conference on Robot Learning* (ML Research Press, 2020), pp. 66–75.
28. G. B. Margolis, T. Chen, K. Paigwar, X. Fu, D. Kim, S. Kim, and P. Agrawal, Learning to jump from pixels, in *Conference on Robot Learning* (ML Research Press, 2022), pp. 1025–1034.
29. A. Kumar, Z. Fu, D. Pathak, J. Malik, RMA: Rapid motor adaptation for legged robots. arXiv:2107.04034. [cs.LG] (8 July 2021).
30. J. Lee, J. Hwangbo, L. Wellhausen, V. Koltun, M. Hutter, Learning quadrupedal locomotion over challenging terrain. *Sci. Robot.* **5**, eabc5986 (2020).
31. J. Xu, T. Aykut, D. Ma, E. Steinbach, 6dls: Modeling nonplanar frictional surface contacts for grasping using 6-d limit surfaces. *IEEE Trans. Robot.* **37**, 2099–2116 (2021).
32. J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, 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)* (IEEE, 2017), pp. 23–30.
33. N. C. Dafle, A. Rodriguez, R. Paolini, B. Tang, S. S. Srinivasa, M. Erdmann, M. T. Mason, I. Lundberg, H. Staab, T. Fuhlbrigge, Extrinsic dexterity: In-hand manipulation with external forces, in *2014 IEEE International Conference on Robotics and Automation (ICRA)* (IEEE, 2014), pp. 1578–1585.
34. M. Ahn, H. Zhu, K. Hartikainen, H. Ponte, A. Gupta, S. Levine, V. Kumar, Robel: Robotics benchmarks for learning with low-cost robots, in *Conference on Robot Learning* (PMLR, 2020), pp. 1300–1313.

35. J. L. Hintze, R. D. Nelson, Violin plots: A box plot-density trace synergism. *Am. Stat.* **52**, 181–184 (1998).
36. A. Handa, A. Allshire, V. Makoviychuk, A. Petrenko, R. Singh, J. Liu, D. Makoviichuk, K. Van Wyk, A. Zhurkevich, B. Sundaralingam, Y. S. Narang, Dextreme: Transfer of agile in-hand manipulation from simulation to reality, in *2023 IEEE International Conference on Robotics and Automation (ICRA)* (IEEE, 2023), pp. 5977–5984.
37. C. Chen, P. Culbertson, M. Lepert, M. Schwager, J. Bohg, Trajectotree: Trajectory optimization meets tree search for planning multi-contact dexterous manipulation, in *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (IEEE, 2021), pp. 8262–8268.
38. T. Pang, H. Suh, L. Yang, R. Tedrake, Global planning for contact-rich manipulation via local smoothing of quasi-dynamic contact models. arXiv:2206.10787 [cs.RO] (22 June 2022).
39. J. Xu, T. Chen, L. Zlokapa, M. Foshey, W. Matusik, S. Sueda, P. Agrawal, An end-to-end differentiable framework for contact-aware robot design. arXiv:2107.07501 [cs.RO] (15 June 2021).
40. S. Brahmbhatt, C. Ham, C. C. Kemp, J. Hays, ContactDB: Analyzing and predicting grasp contact via thermal imaging, in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (IEEE, 2019).
41. V. Makoviychuk, L. Wawrzyniak, Y. Guo, M. Lu, K. Storey, M. Macklin, D. Hoeller, N. Rudin, A. Allshire, A. Handa, G. State, Isaac gym: High performance GPU based physics simulation for robot learning, in *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track* (NeurIPS, 2021).
42. S. Ross, G. Gordon, 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* (ML Research Press, 2011), pp. 627–635.
43. J. Schulman, F. Wolski, P. Dhariwal, A. Radford, O. Klimov, Proximal policy optimization algorithms. arXiv:1707.06347 [cs.LG] (20 July 2017).
44. A. Y. Ng, D. Harada, S. Russell, Policy invariance under reward transformations: Theory and application to reward shaping, in *Proceedings of the Sixteenth International Conference on Machine Learning*, vol. 99 (ICML, 1999), pp. 278–287.
45. C. Choy, J. Gwak, S. Savarese, 4D spatio-temporal convnets: Minkowski convolutional neural networks, in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (IEEE, 2019), pp. 3075–3084.
46. K. Cho, B. van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, Y. Bengio, 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*) (Association for Computational Linguistics, 2014), pp. 1724–1734.
47. J. Tan, T. Zhang, E. Coumans, A. Iscen, Y. Bai, D. Hafner, S. Bohez, V. Vanhoucke, Sim-to-real: Learning agile locomotion for quadruped robots. arXiv:1804.10332v2 [cs.RO] (27 April 2018).
48. N. Hansen, S. D. Müller, P. Koumoutsakos, Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (CMA-ES). *Evol. Comput.* **11**, 1–18 (2003).
49. L. Downs, A. Francis, N. Koenig, B. Kinman, R. Hickman, K. Reymann, T. B. McHugh, V. Vanhoucke, Google scanned objects: A high-quality dataset of 3D scanned household items, in *2022 International Conference on Robotics and Automation (ICRA)* (IEEE, 2022), pp. 2553–2560.
50. K. Mamou, E. Lengyel, A. Peters, Volumetric hierarchical approximate convex decomposition, in *Game Engine Gems 3* (AK Peters, 2016), pp. 141–158.
51. D.-A. Clevert, T. Unterthiner, S. Hochreiter, Fast and accurate deep network learning by exponential linear units (elus), in *4th International Conference on Learning Representations* (*ICLR*, 2016).
52. D. P. Kingma, J. Ba, Adam: A method for stochastic optimization, in *3rd International Conference on Learning Representations* (ICLR, 2015).
53. K. Daniilidis, Hand-eye calibration using dual quaternions. *Int. J. Rob. Res.* **18**, 286–298 (1999).
54. M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, A. Y. Ng, ROS: An open-source robot operating system, in *ICRA Workshop on Open Source Software* (IEEE, 2009), vol. 3, p. 5.
55. P. Florence, C. Lynch, A. Zeng, O. A. Ramirez, A. Wahid, L. Downs, A. Wong, J. Lee, I. Mordatch, J. Tompson, Implicit behavioral cloning, in *Conference on Robot Learning* (ML Research Press, 2022), pp. 158–168.
56. Y. Zhou, C. Barnes, J. Lu, J. Yang, H. Li, On the continuity of rotation representations in neural networks, in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (IEEE, 2019), pp. 5745–5753.

Acknowledgments: We thank the members of the Improbable AI Laboratory for the helpful discussions and feedback on the paper. We are grateful to MIT Supercloud and the Lincoln Laboratory Supercomputing Center for providing HPC resources. **Funding:** Toyota Research Institute, DARPA Machine Common Sense, MIT-IBM Watson AI Lab, and MIT-Airforce AI Accelerator provided funds to Improbable AI Laboratory to support this work. M.T. was supported by a National Science Foundation Graduate Research Fellowship. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the DARPA, the U.S. Air Force, or the U.S. government. The U.S. government is authorized to reproduce and distribute reprints for government purposes notwithstanding any copyright notation herein. **Author contributions:** T.C. and P.A. jointly conceived the project. T.C. formulated the main idea of the training and control methods, set up the simulation and training code, trained the controllers, designed and built the real-world hardware platforms, developed the software for controlling the real hands, designed and conducted experiments in simulation and in the real world, and led the manuscript writing. M.T. designed and fabricated the fingertips of the robot hand and most of the objects used for real-world evaluation and built the four-finger hand. S.W. experimented with vision network pretraining (stage 0 in the paper) and performed ablations quantifying the effect of each pretraining task. V.K. provided feedback on the manuscript and on experimental results. E.A. provided advice and support for hardware fabrication. P.A. was responsible for overall project supervision, contributed to research discussions, provided advice on experimental design and setup, and played a substantial role in manuscript writing. **Competing interests:** P.A. is an advisor to Tutor Intelligence Inc., Common Sense Machines Inc., and Lab0 Inc. A provisional patent application (US application number 63/468,664) has been filed covering some aspects of this work. **Data and materials availability:** All data needed to evaluate the conclusions in the paper are present in the paper or the Supplementary Materials. The code is available at https://zenodo.org/records/10039109 and https://github.com/Improbable-AI/dexenv.

Submitted 15 November 2022
Accepted 26 October 2023
Published 22 November 2023
10.1126/scirobotics.adc9244