Article

The International Journal of Robotics Research
2020, Vol. 39(1) 3–20

DOI: 10.1177/0278364919887447
journals.sagepub.com/home/ijr

Learning dexterous in-hand manipulation

OpenAI: Marcin Andrychowicz, Bowen Baker, Maciek Chociej, Rafal Józefowicz, Bob McGrew, Jakub Pachocki, Arthur Petron, Matthias Plappert, Glenn Powell, Alex Ray, Jonas Schneider, Szymon Sidor, Josh Tobin, Peter Welinder, Lilian Weng and Wojciech Zaremba

Abstract

We use reinforcement learning (RL) to learn dexterous in-hand manipulation policies that can perform vision-based object reorientation on a physical Shadow Dexterous Hand. The training is performed in a simulated environment in which we randomize many of the physical properties of the system such as friction coefficients and an object's appearance. Our policies transfer to the physical robot despite being trained entirely in simulation. Our method does not rely on any human demonstrations, but many behaviors found in human manipulation emerge naturally, including finger gaiting, multi-finger coordination, and the controlled use of gravity. Our results were obtained using the same distributed RL system that was used to train OpenAI Five. We also include a video of our results: https://youtu.be/jwSbzNHGflM.

Keywords

Dexterous manipulation, multifingered hands, adaptive control, learning and adaptive systems, humanoid robots

1. Introduction

While dexterous manipulation of objects is a fundamental everyday task for humans, it is still challenging for autonomous robots. Modern-day robots are typically designed for specific tasks in constrained settings and are largely unable to utilize complex end-effectors. In contrast, people are able to perform a wide range of dexterous manipulation tasks in a diverse set of environments, making the human hand a grounded source of inspiration for research into robotic manipulation.

The Shadow Dexterous Hand (ShadowRobot, 2005) is an example of a robotic hand designed for human-level dexterity; it has five fingers with a total of 24 degrees of freedom (DoFs). The hand has been commercially available since 2005; however, it still has not seen widespread adoption, which can be attributed to the daunting difficulty of controlling systems of such complexity. The state-of-the-art in controlling five-fingered hands is severely limited. Some prior methods have shown promising in-hand manipulation results in simulation but do not attempt to transfer to a real-world robot (Bai and Liu, 2014; Mordatch et al., 2012). Conversely, owing to the difficulty in modeling such complex systems, there has also been work in approaches that only train on a physical robot (Falco et al., 2018; Kumar et al., 2016a,b; van Hoof et al., 2015). However, because physical trials are so slow and costly to run, the learned behaviors are very limited.

In this work, we demonstrate methods to train control policies that perform in-hand manipulation and deploy them on a physical robot. The resulting policy exhibits unprecedented levels of dexterity and naturally discovers grasp types found in humans, such as the tripod, prismatic, and tip pinch grasps, and displays contact-rich, dynamic behaviors such as finger gaiting, multi-finger coordination, the controlled use of gravity, and coordinated application of translational and torsional forces to the object. Figure 1 depicts an exemplary manipulation sequence. Our policy can also use vision to sense an object's pose: an important aspect for robots that should ultimately work outside of a controlled lab setting.

Despite training entirely in a simulator that substantially differs from the real world, we obtain control policies that perform well on the physical robot. We attribute our transfer results to (1) extensive randomizations and added effects in the simulated environment alongside calibration, (2)

OpenAI, San Francisco, CA, USA

Corresponding author:
Matthias Plappert, OpenAI, 3180 18th Street, San Francisco, CA 94110, USA.
Email: matthias@openai.com

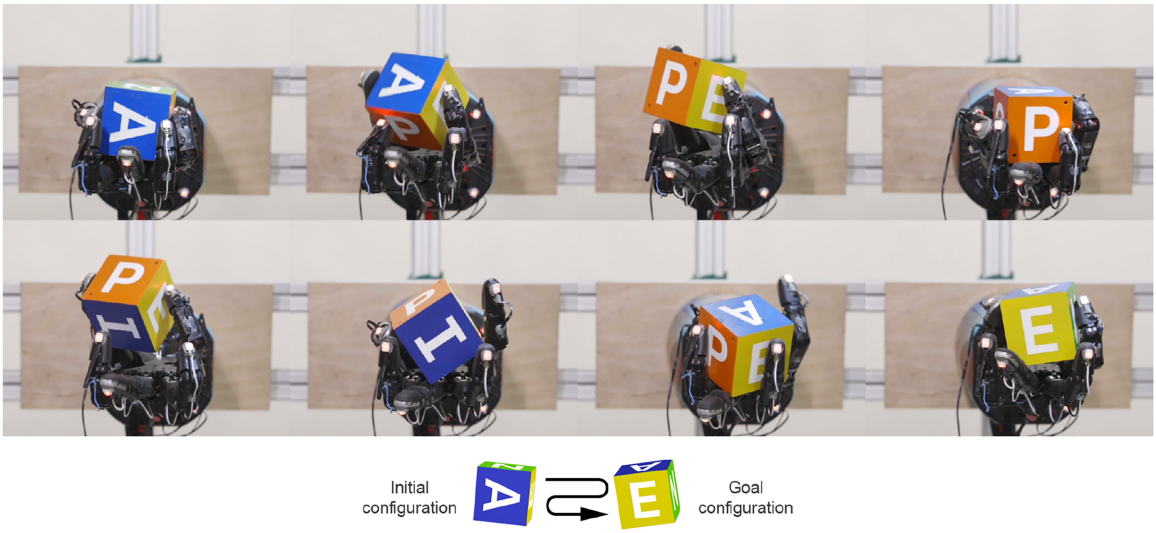

Fig. 1. A five-fingered humanoid hand trained with reinforcement learning manipulating a block from an initial configuration to a goal configuration using vision for sensing.

memory augmented control polices that admit the possibility to learn adaptive behavior and implicit system identification on the fly, and (3) training at large scale with distributed reinforcement learning (RL). An overview of our approach is depicted in Figure 2.

The paper is structured as follows. In Section 2 we briefly introduce the most important RL concepts and algorithms used in this work. Section 3 gives a system overview, describes the proposed task in more detail, and shows the hardware setup. Section 4 describes observations for the control policy, environment randomizations, and additional effects added to the simulator that make transfer possible. Section 5 outlines the control policy training procedure and our distributed RL system. Section 6 describes the vision model architecture and training procedure. Section 7 describes both qualitative and quantitative results from deploying the control policy and vision model on a physical robot. Section 8 discusses related work and we conclude with Section 9.

2. Background

In this section, we introduce the most fundamental RL concepts and discuss the algorithms that we use in this work. For an in-depth introduction to RL, please refer to Sutton and Barto (1998) and Bertsekas (2005).

2.1. RL

We consider the standard RL formalism consisting of an agent interacting with an environment. To simplify the exposition, we assume in this section that the environment is fully observable.[1] An environment is described by a set of states $\mathcal{S}$, a set of actions $\mathcal{A}$, a distribution of initial states $p(s_0)$, a reward function $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, transition probabilities $p(s_{t+1}|s_t, a_t)$, and a discount factor $\gamma \in [0, 1]$.

A policy π is a mapping from a state to a distribution over actions. Every episode starts by sampling an initial state s_0. At every timestep t the agent produces an action based on the current state: $a_t \sim \pi(\cdot\,|s_t)$. In turn, the agents receive a reward $r_t = r(s_t, a_t)$ and the environment's new state s_{t+1}, which is sampled from the distribution $p(\cdot\,|s_t, a_t)$. The discounted sum of future rewards, also referred to as the *return*, is defined as $R_t = \sum_{i=t}^{\infty} \gamma^{i-t} r_i$. The agent's goal is to maximize its expected return $\mathbb{E}[R_0|s_0]$, where the expectation is taken over the initial state distribution, policy, and environment transitions accordingly to the dynamics specified above.

The *Q-function* or *action-value* function is defined as $Q^\pi(s_t, a_t) = \mathbb{E}[R_t|s_t, a_t]$, while the *V-function* or *state-value* function is defined as $V^\pi(s_t) = \mathbb{E}[R_t|s_t]$. The value $A^\pi(s_t, a_t) = Q^\pi(s_t, a_t) - V^\pi(s_t)$ is called the *advantage* and indicates whether the action a_t is better or worse than an average action the policy π takes in the state s_t.

2.2. Generalized advantage estimator

Let V be an approximator to the value function of some policy, i.e., $V \approx V^\pi$. The value

$$\hat{V}_t^{(k)} = \sum_{i=t}^{t+k-1} \gamma^{i-t} r_i + \gamma^k V(s_{t+k}) \approx V^\pi(s_t, a_t)$$

is called the k-step return estimator. The parameter k controls the bias–variance tradeoff of the estimator with bigger values resulting in an estimator closer to empirical returns and having less bias and more variance. The *generalized advantage estimator* (GAE) (Schulman et al., 2015) is a method of combining multi-step returns in the following way:

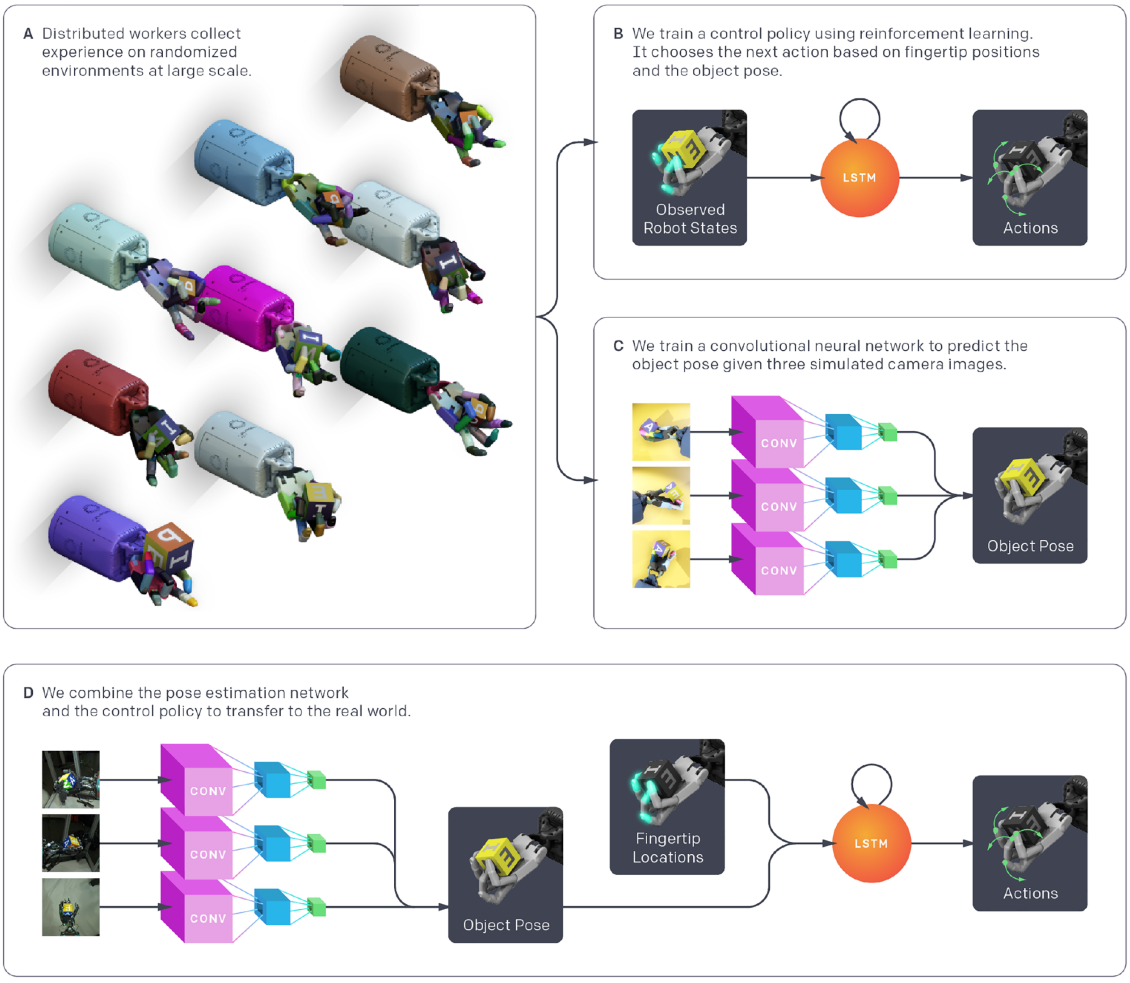

Fig. 2. System overview. (a) We use a large distribution of simulations with randomized parameters and appearances to collect data for both the control policy and vision-based pose estimator. (b) The control policy receives observed robot states and rewards from the distributed simulations and learns to map observations to actions using a recurrent neural network and RL. (c) The vision-based pose estimator renders scenes collected from the distributed simulations and learns to predict the pose of the object from images using a convolutional neural network (CNN), trained separately from the control policy. (d) To transfer to the real world, we predict the object pose from three real camera feeds with the CNN, measure the robot fingertip locations using a 3D motion capture system, and give both of these to the control policy to produce an action that gets executed on the physical robot.

$$\hat{V}_t^{\mathrm{GAE}} = (1-\lambda)\sum_{k>0}\lambda^{k-1}\hat{V}_t^{(k)} \approx V^{\pi}(s_t, a_t)$$

where $0<\lambda<1$ is a hyperparameter. The *advantage* can then be estimated as follows:

$$\hat{A}_t^{\mathrm{GAE}} = \hat{V}_t^{\mathrm{GAE}} - V(s_t) \approx A^{\pi}(s_t, a_t)$$

It is possible to compute the values of this estimator for all states encountered in an episode in linear time (Schulman et al., 2015).

2.3. Proximal policy optimization

Proximal policy optimization (PPO) (Schulman et al., 2017) is one of the most popular on-policy RL algorithms. It simultaneously optimizes a stochastic policy as well as an approximator to the value function. PPO interleaves the collection of new episodes with policy optimization. After a batch of new transitions is collected, optimization is performed with minibatch stochastic gradient descent to maximize the objective:

$$L_{\mathrm{PPO}} = \mathbb{E}\min\left(\frac{\pi(a_t|s_t)}{\pi_{\mathrm{old}}(a_t|s_t)}\hat{A}_t^{\mathrm{GAE}}, \mathrm{clip}\left(\frac{\pi(a_t|s_t)}{\pi_{\mathrm{old}}(a_t|s_t)}, 1-\epsilon, 1+\epsilon\right)\hat{A}_t^{\mathrm{GAE}}\right) \quad (1)$$

where $\frac{\pi(a_t|s_t)}{\pi_{\mathrm{old}}(a_t|s_t)}$ is the ratio of the probability of taking the given action under the current policy π to the probability of taking the same action under the old behavioral policy that was used to generate the data. Here ϵ is a

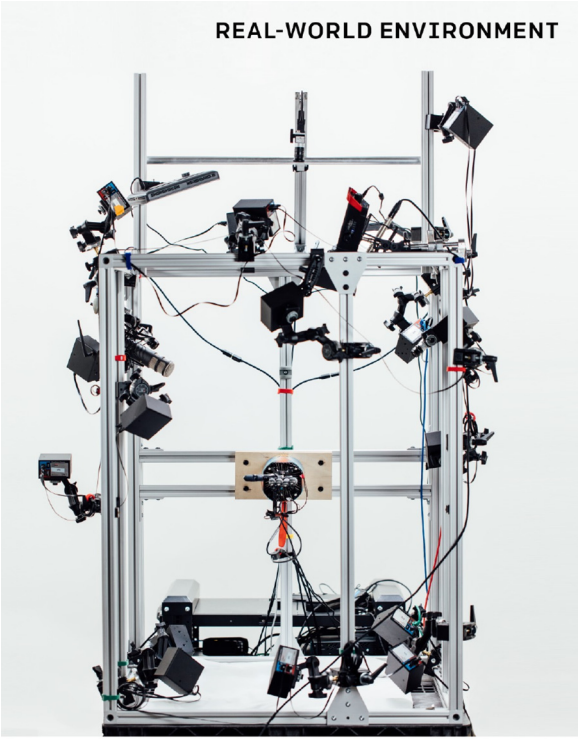

Fig. 3. The "cage" that houses the robot hand, 16 PhaseSpace tracking cameras, and 3 Basler RGB cameras.

hyperparameter (usually $\epsilon \approx 0.2$) that controls the amount of clipping. This loss encourages the policy to take actions that are better than average (have positive advantage) while clipping discourages bigger changes to the policy by limiting how much can be gained by changing the policy on a particular data point.

The value function approximator is trained with supervised learning with the target for $V(s_t)$ being $\hat{V}_t^{\mathrm{GAE}}$. To boost exploration, it is a common practice to encourage the policy distribution to have high entropy by including an entropy bonus in the optimization objective.

3. Task and system overview

In this work, we consider the problem of in-hand object reorientation. We place the object under consideration onto the palm of a humanoid robot hand. The goal is to reorient the object to a desired target configuration in-hand. As soon as the current goal is (approximately) achieved, a new goal is provided until the object is eventually dropped. We use two different objects, a block and an octagonal prism.

This section first describes our hardware setup in detail and then describes how we model the task in simulation.

3.1. Hardware

Our hardware setup consists of a Shadow Dexterous Hand, a PhaseSpace tracking system, as well as a RGB camera system for vision. The entire setup is depicted in Figure 3.

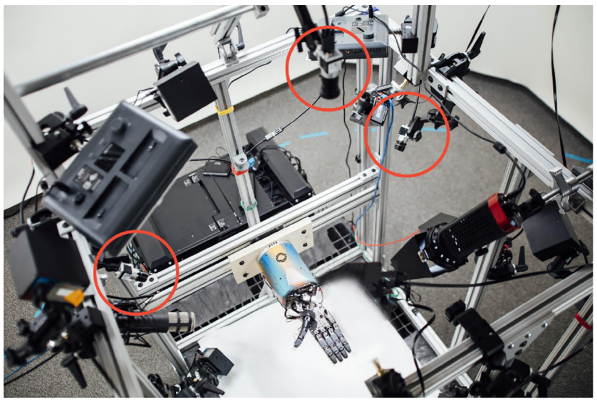

Fig. 4. Our three-camera setup for vision-based state estimation.

3.1.1. Shadow Dexterous Hand. We use the Shadow Dexterous Hand, which is a humanoid robotic hand with 24 DoFs actuated by 20 pairs of agonist–antagonist tendons. Of the 24 DoFs, 16 can be controlled independently whereas the remaining 8 joints (which are the joints between the non-thumb finger proximal, middle, and distal segments) form 4 pairs of coupled joints. We use the version with electric motor actuators.

3.1.2. PhaseSpace tracking. We use a 3D tracking system to localize the tips of the fingers, to perform calibration procedures, and as ground truth for the RGB image-based object tracking. The PhaseSpace Impulse X2E tracking system uses active LED markers that blink to transmit a unique ID code and linear detector arrays in the cameras to detect the positions and IDs. The system features capture speeds of up to 960 Hz and positional accuracies of below 20 μm. The data is exposed as a 3D point cloud together with labels associating the points with stable numerical IDs. Our setup uses 16 cameras distributed spherically around the hand and centered on the palm with a radius of approximately 0.8 meters.

3.1.3. RGB cameras. For estimating the pose of the object that the hand is manipulating, we have two setups: one that uses PhaseSpace markers to track the object (described above) and one that uses three Basler RGB cameras for vision-based pose estimation. This is because our goal is to eventually have a system that works outside of a lab environment, and vision-based systems are better equipped to handle the real world.

Each Basler acA640-750uc RGB camera has a resolution of 640×480 and is placed approximately 50 cm from the Shadow hand. We use three cameras to resolve pose ambiguities that may occur with monocular vision. We chose these cameras for their flexible parameterization and low latency. Figure 4 shows the placement of the cameras relative to the hand.

3.1.4. Control. The high-level controller is implemented as a Python program running a neural network policy using

Tensorflow (Abadi et al., 2016) on a GPU. Every 80 ms it queries the PhaseSpace sensors and then runs inference with the neural network to obtain the action, which takes roughly 25 ms. If vision-based state estimation is used, we additionally use the video feed from the three RGB cameras and produce a pose estimate of the object before feeding it into the policy. The policy outputs an action that specifies the change of position for each actuator, relative to the current position of the joints controlled by the actuator. It then sends the action to the low-level controller.

The low-level controller is implemented in C++ as a separate process on a different machine that is connected to the Shadow hand via an Ethernet cable. The controller is written as a real-time system: it is pinned to a CPU core, has preallocated memory, and does not depend on any garbage collector to avoid non-deterministic delays. The controller receives the relative action, converts it into an absolute joint angle and clips to the valid range, then sets each component of the action as the target for a PD controller. Every 5 ms, the PD controller queries the Shadow Hand joint angle sensors, then attempts to achieve the desired position.

3.1.5. Joint sensor calibration. The hand contains 26 Hall effect sensors that sense magnetic field rotations along the joint axis. To transform the raw magnetic measurements from the Hall sensors into joint angles, we use a piecewise linear function interpolated from 3–5 truth points per joint. To calibrate this function, we initialize to the factory default created using physical calibration jigs. For further accuracy, we attach PhaseSpace markers to the fingertips, and minimize the error between the position reported by the PhaseSpace markers and the position estimated from the joint angles. We estimate these linear functions by minimizing the reprojection error with `scipy.minimize`.

3.2. Simulation

We simulate the physical system with the MuJoCo physics engine (Todorov et al., 2012) and we use Unity (Unity Technologies, 2005) to render the images for training the vision-based pose estimator. Our model of the Shadow Dexterous Hand is based on that used in the OpenAI Gym robotics environments (Brockman et al., 2016; Plappert et al., 2018), but has been improved to match the physical system more closely through calibration. A rendering of our simulation is depicted in Figure 5. In the remainder of this section, we describe all aspects of our simulation in detail.

3.2.1. States. The state of the system is 60-dimensional and consists of angles and velocities of all robot joints as well as the position, rotation, and velocities (linear and angular) of the object. Initial states are sampled by placing the object on the robot's palm in a random orientation and applying random actions for 100 steps (we discard the trial if the object is dropped in the meantime).

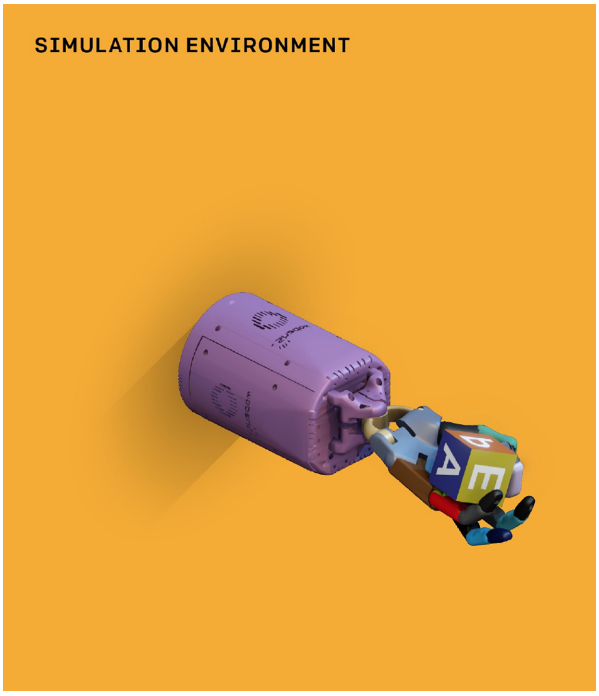

Fig. 5. A rendering of our simulated environment.

3.2.2. Goals. The goal is the desired orientation of the object represented as a quaternion. A new goal is generated after the current one has been achieved within a tolerance of 0.4 rad. We consider a goal achieved if there exists a rotation of the object around an arbitrary axis with an angle smaller than 0.4 rad which transforms the current orientation into the desired one.

3.2.3. Actions. Actions are 20-dimensional and correspond to the desired angles of the hand joints. We discretize each action coordinate into 11 bins of equal size. Owing to the inaccuracy of joint angle sensors on the physical hand, actions are specified relative to the current hand state. In particular, the torque applied to the given joint in simulation is equal to $P \cdot (s_t + a - s_{t'})$, where s_t is the joint angle at the time when the action was specified, a is the corresponding action coordinate, $s_{t'}$ is the current joint angle, and P is the proportionality coefficient. For the coupled joints, the desired and actual positions represent the sum of the two joint angles.

All actions are rescaled to the range $[-1, 1]$. To avoid abrupt changes to the action signal, which could harm a physical robot, we smooth the actions using an exponential moving average using a coefficient of 0.3 per 80 ms. before applying them (both in simulation and during deployments on the physical robot).

3.2.4. Rewards. The reward given at timestep t is $r_t = d_t - d_{t+1}$, where d_t and d_{t+1} are the rotation angles between the desired and current object orientations before and after the transition, respectively. We give an additional

reward of 5 whenever a goal is achieved with the tolerance of 0.4 rad (i.e., $d_{t+1} < 0.4$) and a reward of -20 (penalty) whenever the object is dropped.

3.2.5. Timing. Each environment step corresponds to 80 ms of real time and consists of 10 consecutive MuJoCo simulation steps, each corresponding to 8 ms. An episode ends whenever either the policy achieves 50 consecutive goals, the policy fails to achieve the current goal within 8 seconds of simulated time, or the object is dropped.

3.2.6. Model calibration. We calibrate the parameters of our MuJoCo XML model to better match our physical setup. To do so, we record a trajectory on the physical robot and then optimize over parameters to minimize the error between the simulated and real trajectory.

To create the trajectory, we run two hand-designed policies in sequence against each finger. The first policy measures the behavior of the joints near their limits by extending the joints of each finger completely inward and then completely outward until they stop moving. The second policy measures the dynamic response of the finger by moving the joints of each finger inward and then outward in a series of oscillations. The recorded trajectory across all fingers lasts a few minutes.

To optimize the model parameters, these trajectories are then replayed as open-loop action sequences in the simulator. The optimization objective is to match simulated and real joint angles after 1 second. Parameters are adjusted using iterative coordinate descent until the error is minimized. We exclude modifications to the XML that do not yield an improvement larger than 0.1%. For each joint, we optimize damping, equilibrium position, static friction loss, stiffness, margin, and the minimum and maximum of the joint range. For each actuator, we optimize its proportional gain, force range, and the magnitude of backlash in each direction. Collectively, this corresponds to 264 values.

4. Transferable simulations

Despite our calibration and modeling efforts, the simulation is still only a rough approximation of the physical setup. For example, our model directly applies torque to joints instead of tendon-based actuation and uses rigid-body contact models instead of deformable-body contact models even though the physical robot has deformable fingertips made out of rubber. These differences cause a "reality gap" and make it unlikely for a policy trained in a simulation with these inaccuracies to transfer well.

We therefore face a dilemma: we cannot train on the physical robot because deep RL algorithms require millions of samples; conversely, training only in simulation results in policies that do no transfer well due to the gap between the simulated and real environments. To overcome the reality gap, we modify the basic version of our simulation to a *distribution over many simulations* that foster transfer (Peng et al., 2017; Sadeghi and Levine, 2017; Tobin et al., 2017a). By carefully selecting the sensing modalities and by randomizing most aspects of our simulated environment we are able to train policies that are less likely to overfit to a specific simulated environment and more likely to transfer successfully to the physical robot.

4.1. Observations

We give the control policy observations of the fingertips using PhaseSpace markers and the object pose either from PhaseSpace markers or the vision-based pose estimator. Although the Shadow Dexterous Hand contains a broad array of built-in sensors, we specifically avoided providing these as observations to the policy because they are subject to state-dependent noise that would have been difficult to model in the simulator. For example, the fingertip tactile sensor measures the pressure of a fluid stored in a balloon inside the fingertip, which correlates with the force applied to the fingertip but also with a number of confounding variables, including atmospheric pressure, temperature, and the shape of the contact and intersection geometry. Although it is straightforward to determine the existence of contacts in the simulator, it would be difficult to model the distribution of sensor values. Similar considerations apply to the joint angles measured by Hall effect sensors, which are used by the low-level controllers but not provided to the policy due to their tendency to be noisy and hard to calibrate.

4.2. Randomizations

Following previous work on *domain randomization* (Peng et al., 2017; Sadeghi and Levine, 2017; Tobin et al., 2017a), we randomize most of the aspects of the simulated environment in order to learn both a policy and a vision model that generalizes to reality. We overall found that it is important to center randomized parameters on reasonable physical values of the actual setup, which we obtain via the previously described calibration step. Randomizations also allow us to model uncertainty: we typically randomize parameters with high uncertainty (e.g., actuation parameters) more than parameters for which we have values with low uncertainty (e.g., object dimensions).

4.2.1. Observation noise. To better mimic the kind of noise we expect to experience in reality, we add Gaussian noise to policy observations. In particular, we apply a correlated noise that is sampled once per episode as well as an uncorrelated noise sampled at every timestep. Apart from Gaussian correlated noise, we also add more structured noise coming from inaccurate placement of the motion capture markers by computing the observations using slightly misplaced markers in the simulator. The configuration of noise levels is described in Table 1.

4.2.2. Physics randomizations. The physical parameters are sampled at the beginning of every episode and held fixed for the whole episode. We typically randomize around

Table 1. Standard deviation of applied Gaussian observation noise.

Observation	Correlated	Uncorrelated
Fingertips positions	±1 mm	±2 mm
Object position	±5 mm	±1 mm
Object orientation	±0.1 rad	±0.1 rad
Fingertip marker positions	±3 mm	
Hand base marker position	±1 mm	

Table 2. Ranges of physics parameter randomizations.

Parameter	Scaling factor range
Object dimensions	uniform([0.95, 1.05])
Object and robot link masses	uniform([0.5, 1.5])
Surface friction coefficients	uniform([0.7, 1.3])
Robot joint damping coefficients	loguniform([0.3, 3.0])
Actuator force gains (P term)	loguniform([0.75, 1.5])
Parameter	**Additive term range**
Joint limits	$\mathcal{N}(0, 0.15)$ rad
Gravity vector (per coordinate)	$\mathcal{N}(0, 0.4)$ m/s^2

values that we obtained through calibration. The full set of randomized values are available in Table 2.

We also randomize the timing of environment steps. Every environment step is simulated as 10 MuJoCo physics simulator steps with $\Delta t = 8\text{ms} + \text{Exp}(\lambda)$, where $\text{Exp}(\lambda)$ denotes the exponential distribution and the coefficient λ is uniformly sampled once per episode from the range $[1, 250, 10, 000]$.

4.2.3. Unmodeled effects. The physical robot experiences many effects that are not modeled by our simulation, e.g., motor backlash or motion capture occlusions. Here, we briefly describe each randomization.

PhaseSpace tracking errors Noise aside, readings of the motion capture markers from the PhaseSpace system might be occasionally unavailable for a short period of time due to instability of the service. To model such error in the simulator, we mask the fingertip markers with a small probability (0.2 per second) for 1 second so that the policy has a chance to learn how to interact with the environment while the system temporarily loses track of some markers. Furthermore, the markers might be occluded while in motion, causing a brief delay of readings of some fingertip positions. In the simulator, a small weightless cuboid site[2] is attached to the back of each nail and we consider a marker occluded whenever a collision with the site is detected as another finger or object is getting too close. If a fingertip marker is deemed occluded, we use its last available position readings instead of the current one.

Action noise and delay We add correlated and uncorrelated Gaussian noise to all actions to account for an imperfect actuation system. The detailed noise levels can be found in Table 3. Moreover, the real system contains many potential sources of delays between the time that observations are sensed and actions are executed, from network delay to the computation time of the neural network. Therefore, we introduce a simple model of action delay to the simulator. At the beginning of every episode we sample for every actuator whether it is going to be delayed (with probability 0.5) or not. The actions corresponding to delayed actuator are delayed by one environment step, i.e., approximately 80 ms.

Table 3. Standard deviation of action noise.

Noise type	Percentage of range
Uncorrelated additive	5%
Correlated additive	1.5%
Uncorrelated multiplicative	1.5%

Backlash model The physical Shadow Dexterous Hand is tendon-actuated, which causes a substantial amount of backlash, while the MuJoCo model assumes direct actuation on the joints. In order to account for it, we introduce a simple model of backlash that modifies actions before they are sent to MuJoCo. In particular, for every joint we have two parameters that specify the amount of backlash in each direction, and are denoted δ_{-1} and δ_{+1}, as well as a time-varying variable s denoting the current state of slack. We obtained the values of δ_{-1}, δ_{+1} through calibration. At the beginning of every episode we sample the values of δ_{-1}, δ_{+1} from the Gaussian distribution centered around the calibrated values with the standard deviation of 0.1. Let $a_{\text{in}} \in [-1, 1]$ be an action specified by the policy. Our backlash model works as follows: we compute the new value of the slack variable $s' = [s + a_{\text{in}}\delta_{\mathbf{sgn}(a_{\text{in}})}\Delta t]_{-1}^{+1}$, compute the scaling factor $\alpha = 1 - \left[\frac{|\mathbf{sgn}(a_{\text{in}}) - s|}{|s' - s| + \epsilon}\right]_0^1$, where $\epsilon = 10^{-12}$ is a constant used for numerical stability, and finally multiply the action by α: $a_{\text{out}} = \alpha a_{\text{in}}$.

Random forces To represent unmodeled dynamics, we sometimes apply random forces to the object. The probability p that a random force is applied is sampled at the beginning of the episode from the loguniform distribution between 0.1% and 10%. Then, at every timestep, with probability p we apply a random force from the three-dimensional Gaussian distribution with the standard deviation equal to 1 m/s^2 times the mass of the object on each coordinate and decay the force with a coefficient of 0.99 per 80 ms.

4.2.4. Visual appearance randomizations. We randomize the following aspects of the rendered scene: camera positions and intrinsics, lighting conditions, the pose of the hand and object, and the materials and textures for all objects in the scene. Figure 6 depicts some examples of these randomized environments.

Fig. 6. Simulations with different randomized visual appearances. Rows correspond to the renderings from the same camera, and columns correspond to renderings from three separate cameras that are simultaneously fed into the neural network.

The materials and textures are randomized for every visible object in the scene. We randomize the hue, saturation, and value for the object faces around calibrated values from real-world measurements. The color of the robot is uniformly randomized. Material properties such as glossiness and shininess are randomized as well. Camera position and orientation are slightly randomized around values calibrated to real-world locations. Lights are randomized individually, and intensities are scaled based on a randomly drawn total intensity. After rendering the scene to images from the three separate cameras, additional augmentation is applied. The images are linearly normalized to have zero mean and unit variance. Then the image contrast is randomized, and finally per-pixel Gaussian noise is added. Details are given in Table 4.

Table 4. Ranges of vision randomizations.

Randomization type	Range
Number of cameras	3
Camera position	± 1.5 mm
Camera rotation	0–3° around a random axis
Camera field of view	± 1°
Robot material colors	uniform over RGB values
Robot material metallic level	5–25%[a]
Robot material glossiness level	0–100%[a]
Object material hue	± 1%
Object material saturation	± 15%
Object material value	± 15%
Object metallic level	5–15%[a]
Object glossiness level	5–15%[a]
Number of lights	4–6
Light position	uniform over upper half-sphere
Light relative intensity	1–5
Total light intensity	0–15[a]
Image contrast adjustment	50–150%
Additive per-pixel Gaussian noise	± 10%

[a]In units used by Unity. See https://unity3d.com/learn/tutorials/s/graphics.

5. Learning control policies from state

We use the previously described distribution over randomized simulations to train a single control policy using RL. Since we optimize for performance over all randomizations, the policy cannot overfit to a single variant of our simulation, thus making it transferable to the physical robot. However, since the policy has to handle a large number of different variants of the same problem, we propose to use a recurrent policy with access to memory. We further use a distributed RL system in order to make solving this challenging problem tractable. Both the policy architecture and our training procedure using our distributed system are described in this section.

5.1. Policy architecture

Many of the randomizations we employ persist across an episode, and thus it should be possible for a memory augmented policy to identify properties of the current environment and adapt its own behavior accordingly. For instance, initial steps of interaction with the environment can reveal the weight of the object or how fast the index finger can move. We therefore represent the policy as a recurrent neural network with memory, namely an LSTM (Hochreiter and Schmidhuber, 1997) with an additional hidden layer with ReLU (Nair and Hinton, 2010) activations inserted between inputs and the LSTM.

The policy is trained with PPO (Schulman et al., 2017). PPO requires the training of two networks: a policy

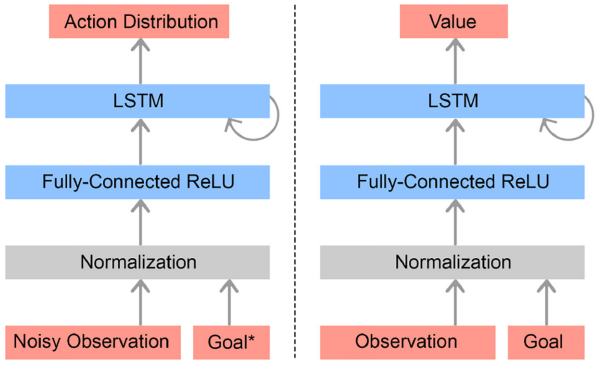

Fig. 7. Policy network (left) and value network (right). Each network consists of an input normalization, a single fully connected hidden layer with 1,024 units and ReLU activations (Nair and Hinton, 2010), and a recurrent LSTM block (Hochreiter and Schmidhuber, 1997) with 512 units. The normalization block subtracts the mean value of each coordinate (across all data gathered so far), divides by the standard deviation, and removes outliers by clipping. There is no weight sharing between the two networks. The goal provided to the policy is the noisy relative target orientation (see Table 5 for details).

network, which maps observations to actions; and a value network, which predicts the discounted sum of future rewards starting from a given state. Both networks have the same architecture but have independent parameters. The network architecture is depicted in Figure 7.

Since the value network is only used during training, we use an Asymmetric Actor–Critic (Pinto et al., 2017a) approach. Asymmetric Actor–Critic exploits the fact that the value network can have access to information that is not available on the real robot system. This includes noiseless observation and additional observations such as joint angles and angular velocities, which we cannot sense reliably but which are readily available in simulation during training. The additional input potentially simplifies the problem of learning good value estimates since less information needs to be inferred.

We also normalize all observations given to the policy and value networks with running means and standard deviations. We then clip observations such that they are within five standard deviations of the mean. We normalize the advantage estimates within each minibatch. We also normalize targets for the value function with running statistics. The list of inputs fed to both networks can be found in Table 5.

5.2. Distributed training with Rapid

We use the same distributed implementation of PPO that was used to train OpenAI Five (OpenAI, 2018) without any modifications. Overall, we found that PPO scales up easily and requires little hyperparameter tuning. The architecture of our distributed training system is depicted in Figure 8.

Table 5. Observations of the policy and value networks, respectively. "Noisy" means that observation noise has been applied as described in Section 4.

Input	Dimension	Policy	Value
Fingertip positions	15	×	
Noisy fingertip positions	15	✓	×
Object position	3	×	✓
Noisy object position	3	✓	×
Object orientation	4	× [a]	✓
Target orientation	4	×	✓
Relative target orientation	4	×	✓
Noisy relative target orientation	4	✓	×
Hand joints angles	24	×	✓
Hand joints velocities	24	×	✓
Object velocity	3	×	✓
Object angular velocity	4	×	✓

[a] We accidentally did not include the current object orientation in the policy observations but found that it makes little difference since this information is indirectly available through the relative target orientation.

In our implementation, a pool of 384 worker machines, each with 16 CPU cores, generate experience by rolling out the current version of the policy in a sample from the previously described distribution of randomized simulations. Workers download the newest policy parameters from the optimizer at the beginning of every epoch, generate training episodes, and send the generated episodes back to the optimizer. The communication between the optimizer and workers is implemented using the Redis in-memory data store. We use multiple Redis instances for load-balancing, and workers are assigned to an instance randomly. This setup allows us to generate about 2 years of simulated experience per hour.

The optimization is performed on a single machine with eight GPUs. The optimizer threads pull down generated experience from Redis and then stage it to their respective GPU's memory for processing. After computing gradients locally, they are averaged across all threads using MPI, which we then use to update the network parameters.

The hyperparameters that we used can be found in Table 6.

6. State estimation from vision

The policy that we described in the previous section takes the object's position as input and requires a motion capture system for tracking the object on the physical robot. This is undesirable because tracking objects with such a system is only feasible in a lab setting where markers can be placed on each object. Since our ultimate goal is to build robots for the real world that can interact with arbitrary objects, sensing them using vision is an important building block. In this work, we therefore wish to infer the object's pose from vision alone. Similar to the policy, we train this estimator only on synthetic data coming from the simulator.

Table 6. Hyperparameters used for PPO.

Hyperparameter	Value
Hardware configuration	8 GPUs + 6,144 CPU cores[a]
Action distribution	categorical (11 bins per coordinate)
Discount factor γ	0.998
GAE λ	0.95
Entropy regularization	0.01
Clipping ϵ	0.2
Optimizer	Adam (Kingma and Ba, 2014)
Learning rate	0.0003
Batch size (per GPU)	80,000 chunks × 10 transitions
Minibatch size (per GPU)	25,600 transitions
Minibatches per step	60

[a]We use NVIDIA V100 GPUs for policy training.

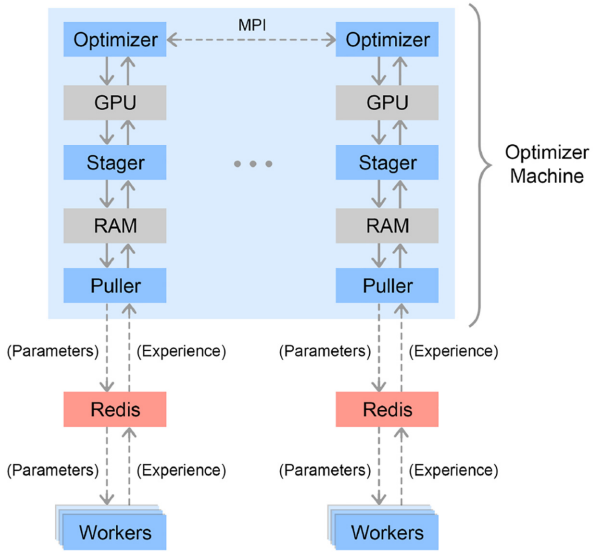

Fig. 8. Our distributed training infrastructure in Rapid. Individual threads are depicted as blue squares. Worker machines randomly connect to a Redis server from which they pull new policy parameters and to which they send new experience. The optimizer machine has one MPI process for each GPU, each of which gets a dedicated Redis server. Each process has a *Puller* thread which pulls down new experience from Redis into a buffer. Each process also has a *Stager* thread which samples minibatches from the buffer and stages them on the GPU. Finally, each *Optimizer* thread uses a GPU to optimize over a minibatch after which gradients are accumulated across threads and new parameters are sent to the Redis servers.

6.1. Model architecture

To resolve ambiguities and to increase robustness, we use three RGB cameras mounted with differing viewpoints of the scene. The recorded images are passed through a convolutional neural network (CNN), which is depicted in Figure 9. The network predicts both the position and the orientation of the object. During execution of the control policy on the physical robot, we feed the pose estimator's prediction into the policy, which in turn produces the next action. The hyperparameters for the vision model architecture are listed in Table 7.

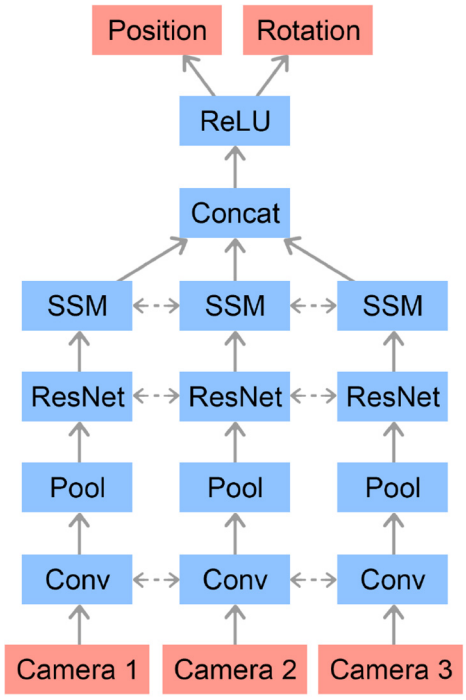

Fig. 9. Vision network architecture. Camera images are passed through a convolutional feature stack that consists of two convolutional layers, max-pooling, 4 ResNet blocks (He et al., 2016), and spatial softmax (SSM) (Finn et al., 2015) layers with shared weights between the feature stacks for each camera. The resulting representations are flattened, concatenated, and fed to a fully connected network. All layers use ReLU (Nair and Hinton, 2010) activation function. Linear outputs from the last layer form the estimates of the position and orientation of the object.

Table 7. Hyperparameters for the vision model architecture.

Layer	Details
Input RGB Image	200 × 200 × 3
Conv2D	32 filters, 5 × 5, stride 1, no padding
Conv2D	32 filters, 3 × 3, stride 1, no padding
Max pooling	3 × 3, stride 3
ResNet	1 block, 16 filters, 3 × 3, stride 3
ResNet	2 blocks, 32 filters, 3 × 3, stride 3
ResNet	2 blocks, 64 filters, 3 × 3, stride 3
ResNet	2 blocks, 64 filters, 3 × 3, stride 3
Spatial Softmax	—
Flatten	—
Concatenate	All 3 image towers combined
Fully connected	128 units
Fully connected	Output dimension (3 position + 4 rotation)

6.2. Training

We run the trained policy in the simulator until we gather one million states. We then train the vision network by

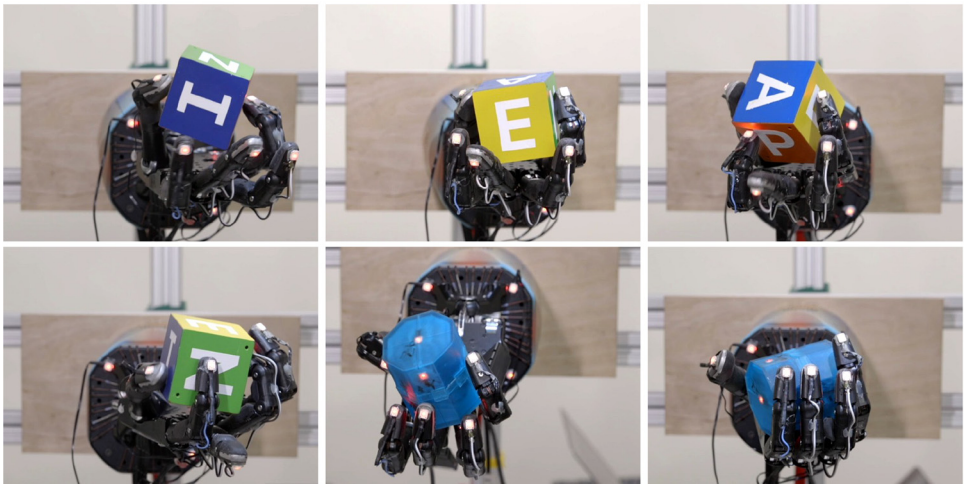

Fig. 10. Different grasp types learned by our policy. From top left to bottom right: Tip Pinch grasp, Palmar Pinch grasp, Tripod grasp, Quadpod grasp, 5-Finger Precision grasp, and a Power grasp. Classified according to Feix et al. (2016).

Table 8. Hyperparameters used for the vision model training.

Hyperparameter	Value
Hardware configuration	3 GPUs + 32 CPU cores[a]
Optimizer	Adam (Kingma and Ba, 2014)
Learning rate	0.0005, halved every 20,000 batches
Minibatch size	$64 \times 3 = 192$ RGB images
Image size	200×200 pixels
Weight decay regularization	0.001
Number of training batches	400,000

[a]We use NVIDIA P40 GPUs for vision training. Two GPUs are used for rendering and one for the optimization.

minimizing the mean squared error between the normalized prediction and the ground-truth with minibatch gradient descent. For each minibatch, we render the images with randomized appearance before feeding them to the network. Moreover, we augment the data by modifying the object pose. More specifically, we leave the object pose as is with 20% probability, rotate the object by 90° around its main axes with 40% probability, and "jitter" the object by adding Gaussian noise to both the position and rotation independently with 40% probability. We use 2 GPUs for rendering and 1 GPU for running the network and training. The hyperparameters used for the vision model training are listed in Table 8.

7. Results

In this section, we evaluate the proposed system. We start by deploying the system on the physical robot, and evaluating its performance on in-hand manipulation of a block and an octagonal prism. We then focus on individual aspects of our system: we conducted an ablation study of the importance of randomizations and policies with memory capabilities in order to successfully transfer. Next, we consider the sample complexity of our proposed method. Finally, we investigate the performance of the proposed vision pose estimator and show that using only synthetic images is sufficient to achieve good performance.

7.1. Qualitative results

During deployment on the robot as well as in simulation, we note that our policies naturally exhibit many of the grasps found in humans (see Figure 10). Furthermore, the policy also naturally discovers many strategies for dexterous in-hand manipulation described by the robotics community (Ma and Dollar, 2011) such as finger pivoting, finger gaiting, multi-finger coordination, the controlled use of gravity, and coordinated application of translational and torsional forces to the object. It is important to note that we did not incentivize this directly: we do not use any human demonstrations and do not encode any prior into the reward function.

For precision grasps, our policy tends to use the little finger instead of the index or middle finger. This may be because the little finger of the Shadow Dexterous Hand has an extra DoF compared with the index, middle, and ring fingers, making it more dexterous. In humans, the index and middle finger are typically more dexterous. This means that our system can rediscover grasps found in humans, but adapt them to better fit the limitations and abilities of its own body.

We observe another interesting parallel between humans and our policy in finger pivoting, which is a strategy in which an object is held between two fingers and rotate around this axis. It was found that young children have not yet fully developed their motor skills and therefore tend to rotate objects using the proximal or middle phalanges of a finger (Pehoski et al., 1997). Only later in their lives do they switch to primarily using the distal phalanx, which is

Table 9. The number of successful consecutive rotations in simulation and on the physical robot. All policies were trained on environments with all randomizations enabled. We performed 100 trials in simulation and 10 trails per policy on the physical robot. Each trial terminates when the object is dropped, 50 rotations are achieved or a timeout is reached. For physical trials, results were taken at different times on the physical robot.

Simulated task	Mean	Median	Individual trials (sorted)									
Block (state)	43.4±13.8	50	-									
Block (state, locked wrist)	44.2±13.4	50	-									
Block (vision)	30.0±10.3	33	-									
Octagonal prism (state)	29.0±19.7	30	-									
Physical task												
Block (state)	18.8±17.1	13	50	41	29	27	14	12	6	4	4	1
Block (state, locked wrist)	26.4±13.4	28.5	50	43	32	29	29	28	19	13	12	9
Block (vision)	15.2±14.3	11.5	46	28	26	15	13	10	8	3	2	1
Octagonal prism (state)	7.8±7.8	5	27	15	8	8	5	5	4	3	2	1

the dominant strategy found in adults. It is interesting that our policy also typically relies on the distal phalanx for finger pivoting.

During experiments on the physical robot we noticed that the most common failure mode was dropping the object while rotating the wrist pitch joint down. Moreover, the vertical joint was the most common source of robot breakages, probably because it handles the biggest load. Given these difficulties, we also trained a policy with the wrist pitch joint locked.[3] We noticed that not only does this policy transfer better to the physical robot but it also seems to handle the object much more deliberately with many of the above grasps emerging frequently in this setting.

Other failure modes that we observed were dropping the object shortly after the start of a trial (which may be explained by incorrectly identifying some aspect of the environment) and getting stuck because the edge of an object got caught in a screw hole (which we do not model).

We encourage the reader to watch the accompanying video to get a better sense of the learned behaviors (please refer to the supplement material).

7.2. Quantitative results

In this section, we evaluate our results quantitatively. To do so, we measure the number of *consecutive* successful rotations until the object is either dropped, a goal has not been achieved within 80 seconds, or until 50 rotations are achieved. All results are available in Table 9.

Our results allow us to directly compare the performance of each task in simulation and on the real robot. For instance, manipulating a block in simulation achieves a median of 50 successes while the median on the physical setup is 13. This is the overall trend that we observe: even though randomizations and calibration narrow the reality gap, it still exists and performance on the real system is worse than in simulation. We discuss the importance of individual randomizations in greater detail in Section 7.3.

When using vision for pose estimation, we achieve slightly worse results both in simulation and on the real robot. This is because even in simulation, our model has to perform transfer because it was only trained on images rendered with Unity but we use MuJoCo rendering for evaluation in simulation (thus making this a sim-to-sim transfer problem). On the real robot, our vision model does slightly worse compared with pose estimation with PhaseSpace. However, the difference is surprisingly small, suggesting that training the vision model only in simulation is enough to achieve good performance on the real robot. For vision pose estimation, we found that it helps to use a white background and to wipe the object with a tack cloth between trials to remove detritus from the robot hand.

We also evaluate the performance on a second type of object, an octagonal prism. To do so, we finetuned a trained block rotation control policy to the same randomized distribution of environments but with the octagonal prism as the target object instead of the block. Even though our randomizations were all originally designed for the block, we were able to learn successful policies that transfer. Compared with the block, however, there is still a performance gap both in simulation and on the real robot. This suggests that further tuning is necessary and that the introduction of additional randomization could improve transfer to the physical system.

We also briefly experimented with a sphere but failed to achieve more than a few rotations in a row, perhaps because we did not randomize any MuJoCo parameters related to rolling behavior or because rolling objects are much more sensitive to unmodeled imperfections in the hand such as screw holes. It would also be interesting to train a unified policy that can handle multiple objects, but we leave this for future work.

Obtaining the results in Table 9 proved to be challenging due to robot breakages during experiments. Repairing the robot takes time and often changes some aspects of the system, which is why the results were obtained at different times. In general, we found that problems with hardware breakage were one of the key challenges we had to overcome in this work.

Table 10. The number of successful consecutive rotations on the physical robot of five policies trained separately in environments with different randomizations held out. The first five rows use PhaseSpace for object pose estimation and were run on the same robot at the same time. Trials for each row were interleaved in case the state of the robot changed during the trials. The last two rows were measured at a different time from the first five and used the vision model to estimate the object pose.

Training environment	Mean	Median	Individual trials (sorted)									
All randomizations (state)	18.8±17.1	13	50	41	29	27	14	12	6	4	4	1
No randomizations (state)	1.1±1.9	0	6	2	2	1	0	0	0	0	0	0
No observation noise (state)	15.1±14.5	8.5	45	35	23	11	9	8	7	6	6	1
No physics randomizations (state)	3.5±2.5	2	7	7	7	3	2	2	2	2	2	1
No unmodeled effects (state)	3.5±4.8	2	16	7	3	3	2	2	1	1	0	0
All randomizations (vision)	15.2±14.3	11.5	46	28	26	15	13	10	8	3	2	1
No observation noise (vision)	5.9±6.6	3.5	20	12	11	6	5	2	2	1	0	0

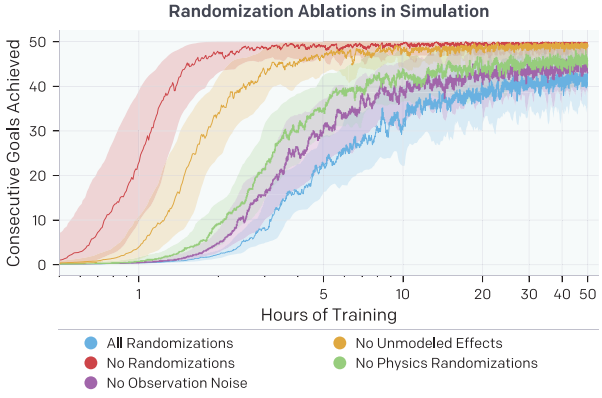

Fig. 11. Performance when training in environments with groups of randomizations held out. All runs show exponential moving averaged performance and 90% confidence interval over a moving window of the RL agent in the environment it was trained. We see that training is faster in environments that are easier, e.g., *no randomizations* and *no unmodeled effects*. We only show one seed per experiment; however, in general we have noticed almost no instability in training.

7.3. Ablation of randomizations

In Section 4.2 we detail a list of parameters we randomize and effects we add that are not already modeled in the simulator. In this section, we show that these additions to the simulator are vital for transfer. We train five separate RL policies in environments with various randomizations held out: *all randomizations* (baseline), *no observation noise, no unmodeled effects, no physics randomizations*, and *no randomizations* (basic simulator, i.e., no domain randomization).

Adding randomizations or effects to the simulation does not come without cost; in Figure 11 we show the training performance in simulation for each environment plotted over wall-clock time. Policies trained in environments with a more difficult set of randomizations, e.g., *all randomizations* and *no observation noise*, converge much slower and therefore require more compute and simulated experience to train in. However, when deploying these policies on the real robot we find that training with randomizations is critical for transfer. Table 10 summarizes our results. Specifically, we find that training with all randomizations leads to a median of 13 consecutive goals achieved, while policies trained with *no randomizations, no physics randomizations*, and *no unmodeled effects* achieve only median of 0, 2, and 2 consecutive goals, respectively.

When holding out *observation noise* randomizations, the performance gap is less clear than for the other randomization groups. We believe that is because our motion capture system has very little noise. However, we still include this randomization because it is important when the vision and control policies are composed. In this case, the pose estimate of the object is much more noisy, and, therefore, training with observation noise should be more important. The results in Table 10 suggest that this is indeed the case, with a drop from median performance of 11.5 to 3.5 if the observation noise randomizations are withheld.

The vast majority of training time is spent making the policy robust to different physical dynamics. Learning to rotate an object in simulation without randomizations requires about 3 years of simulated experience, while achieving the same performance in a fully randomized simulation requires about 100 years of experience. This corresponds to a wall-clock time of around 1.5 hours and 50 hours in our simulation setup, respectively.

7.4. Effect of memory in policies

We find that using memory is helpful to achieve good performance in the randomized simulation. In Figure 12 we show the simulation performance of three different RL architectures: the baseline which has a LSTM policy and value function, a feed-forward (FF) policy and a LSTM value function, and both a FF policy and FF value function. We include results for a FF policy with LSTM value function because it was plausible that having a more expressive value function would accelerate training, allowing the policy to act robustly without memory once it converged. However, we see that the baseline outperforms both variants, indicating that it is beneficial to have some amount of memory in the actual policy.

Table 11. The number of successful consecutive rotations on the physical robot of three policies with different network architectures trained on an environment with all randomizations. Results for each row were collected at different times on the physical robot.

Network architecture	Mean	Median	Individual trials (sorted)									
LSTM policy/LSTM value (state)	18.8±17.1	13	50	41	29	27	14	12	6	4	4	1
FF policy/LSTM value (state)	4.7±4.1	3.5	15	7	6	5	4	3	3	2	2	0
FF policy/FF value (state)	4.6±4.3	3	15	8	6	5	3	3	2	2	2	0

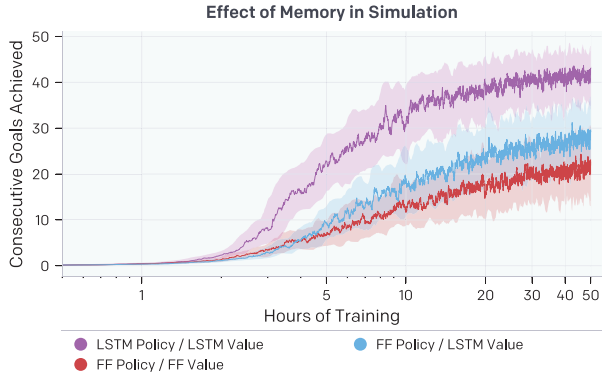

Fig. 12. Performance when comparing LSTM and FF policy and value function networks. We train on an environment with all randomizations enabled. All runs show exponential moving averaged performance and 90% confidence interval over a moving window for a single seed. We find that using recurrence in both the policy and value function helps to achieve good performance in simulation.

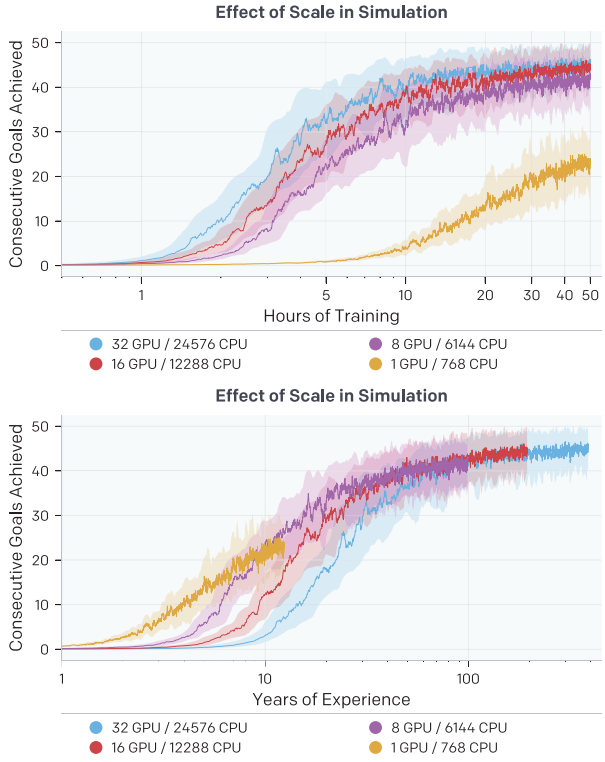

Fig. 13. We show performance in simulation when varying the amount of compute used during training versus wall-clock training time (top) and years of experience consumed (bottom). Batch size used is proportional to the number of GPUs used, such that time per optimization step should remain constant apart from slow downs due to gradient syncing across optimizer machines.

Moreover, we found out that LSTM state is predictive of the environment randomizations. In particular, we discovered that the LSTM hidden state after 5 seconds of simulated interaction with the block allows to predict whether the block is bigger or smaller than average in 80% of cases.

To investigate the importance of memory-augmented policies for transfer, we evaluate the same three network architectures as described above on the physical robot. Table 11 summarizes the results. Our results show that having a policy with access to memory yields a higher median of successful rotations, suggesting that the policy may use memory to adapt to the current environment.[4] Qualitatively we also find that FF policies often get stuck and then run out of time.

In Figure 13 we show results when varying the number of CPU cores and GPUs used in training, where we keep the batch size per GPU fixed such that overall batch size is directly proportional to number of GPUs. Because we could linearly slow down training by simply using less CPU machines and having the GPUs wait longer for data, it is more informative to vary the batch size. We see that our default setup with an 8 GPU optimizer and 6,144 rollout CPU cores reaches 20 consecutive achieved goals approximately 5.5 times faster than a setup with a 1 GPU optimizer and 768 rollout cores. Furthermore, when using 16 GPUs we reach 40 consecutive achieved goals roughly 1.8 times faster than when using the default 8 GPU setup. Scaling up further results in diminishing returns, but it seems that scaling up to 16 GPUs and 12,288 CPU cores gives close to linear speedup.

7.5. Vision performance

In Table 9 we show that we can combine a vision-based pose estimator and the control policy to successfully transfer to the real robot without embedding sensors in the target object. To better understand why this is possible, we evaluate the precision of the pose estimator on both synthetic and real data. Evaluating the system in simulation is easy because we can generate the necessary data and have access

Table 12. Performance of a vision based pose estimator on synthetic and real data.

Test set	Rotation error	Position error
Rendered images (Unity)	2.71°±1.62	3.12mm±1.52
Rendered images (MuJoCo)	3.23°±2.91	3.71mm±4.07
Real images	5.01°±2.47	9.27mm±4.02

to the precise object's pose to compare against. In contrast, real images had to be collected by running a state-based policy on our robot platform. We use PhaseSpace to estimate the object's pose, which is therefore subject to errors. The resulting collected test set consists of 992 real samples.[5] For simulation, we use test sets rendered using Unity and MuJoCo. The MuJoCo renderer was not used during training, thus the evaluation can be also considered as an instance of sim-to-sim transfer. Table 12 summarizes our results.

Our results show that the model achieves low error for both rotation and position prediction when tested on synthetic data.[6] On the images rendered with MuJoCo, there is only a slight increase in error, suggesting successful sim-to-sim transfer. The error further increases on the real data, which is due to the gap between simulation and reality but also because the ground truth is more challenging to obtain due to noise, occlusions, imperfect marker placement, and delayed sensor readings. Despite that the prediction error is bigger than the observation noise used during policy training (Table 1), the vision-based policy performs well on the physical robot (Table 9).

8. Related work

In order to make it easier to understand the state of the art in dexterous in-hand manipulation we gathered a representative set of videos from related work, and created a playlist out of them (see the supplemental material).

8.1. Dexterous manipulation

Dexterous manipulation has been an active area of research for decades (Bicchi, 2000; Fearing, 1986; Ma and Dollar, 2011; Okamura et al., 2000; Rus, 1999). Many different approaches and strategies have been proposed over the years. This includes rolling (Bicchi and Sorrentino, 1995; Cherif and Gupta, 1999; Doulgeri and Droukas, 2013; Han et al., 1997; Han and Trinkle, 1998), sliding (Cherif and Gupta, 1999; Shi et al., 2017), finger gaiting (Han and Trinkle, 1998), finger tracking (Rus, 1992), pushing (Dafle and Rodriguez, 2017), and re-grasping (Dafle et al., 2014; Tournassoud et al., 1987). For some hand types, strategies such as pivoting (Aiyama et al., 1993), tilting (Erdmann and Mason, 1988), tumbling (Sawasaki and Inoue, 1991), tapping (Huang and Mason, 2000), two-point manipulation (Abell and Erdmann, 1995), and two-palm manipulation (Erdmann, 1998) are also options. These approaches use planning and therefore require exact models of both the hand and object. After computing a trajectory, the plan is typically executed open-loop, thus making these methods prone to failure if the model is not accurate.[7]

Other approaches take a closed-loop approach to dexterous manipulation and incorporate sensor feedback during execution, e.g., tactile sensing (Li et al., 2014a,b, 2013; Tahara et al., 2010). While those approaches allow mistakes to be corrected at runtime, they still require reasonable models of the robot kinematics and dynamics, which can be challenging to obtain for under-actuated hands with many DoFs.

Deep RL has also been used successfully to learn complex manipulation skills on physical robots. Guided policy search (Levine and Koltun, 2013; Levine et al., 2015) learns simple local policies directly on the robot and distills them into a global policy represented by a neural network. An alternative is to use many physical robots simultaneously in order to be able to collect sufficient experience (Gu et al., 2017; Kalashnikov et al., 2018; Levine et al., 2018).

8.2. Dexterous in-hand manipulation

Since a very large body of past work on dexterous manipulation exists, we limit the more detailed discussion to setups that are most closely related to our work on dexterous in-hand manipulation.

Mordatch et al. (2012) and Bai and Liu (2014) proposed methods to generate trajectories for complex and dynamic in-hand manipulation, but results were limited to simulation. There has also been significant progress in learning complex in-hand dexterous manipulation (Barth-Maron et al., 2018; Plappert et al., 2018) and even tool use (Rajeswaran et al., 2017) using deep RL, but those approaches were also only evaluated in simulation.

In contrast, multiple authors learn policies for dexterous in-hand manipulation directly on the robot. van Hoof et al. (2015) learned in-hand manipulation for a simple three-fingered gripper whereas Kumar et al. (2016a,b) and Falco et al. (2018) learned such policies for more complex humanoid hands. While learning directly on the robot means that modeling the system is not an issue, it also means that learning has to be performed with only a handful of trials. This is only possible when learning very simple (e.g., linear or local) policies that, in turn, do not exhibit sophisticated behaviors.

8.3. Sim-to-real transfer

Domain adaption methods (Gupta et al., 2017; Tzeng et al., 2015), progressive nets (Rusu et al., 2017), and learning inverse dynamics models (Christiano et al., 2016) were all proposed to help with sim-to-real transfer. All of these methods assume access to real data. An alternative

approach is to make the policy itself more adaptive during training in simulation using *domain randomization*. Domain randomization was used to transfer object pose estimators (Tobin et al., 2017a) and vision policies for fly drones (Sadeghi and Levine, 2017). This idea has also been extended to dynamics randomization (Antonova et al., 2017; Tan et al., 2018; Yu et al., 2017) to learn a robust policy that transfers to similar environments but with different dynamics. Domain randomization was also used to plan robust grasps (Mahler et al., 2017a,b; Tobin et al., 2017b) and to transfer learned locomotion (Tan et al., 2018) and grasping (Zhu et al., 2018) policies for relatively simple robots. Pinto et al. (2017b,c) proposed the use of *adversarial training* to obtain more robust policies and showed that it also helps with transfer to physical robots.

9. Conclusion

In this work, we have demonstrated that in-hand manipulation skills learned with RL in a simulator can achieve an unprecedented level of dexterity on a physical five-fingered hand. This is possible due to extensive randomizations of the simulator, large-scale distributed training infrastructure, policies with memory, and a choice of sensing modalities that can be modeled in the simulator. Our results have demonstrated that, contrary to a common belief, contemporary deep RL algorithms can be applied to solving complex real-world robotics problems that are beyond the reach of existing non-learning-based approaches.

Acknowledgements

We would like to thank Rachel Fong, Ankur Handa, and a former OpenAI employee for exploratory work and helpful discussions, a former OpenAI employee for advice and some repairs on hardware and contributions to the low-level PID controller, Pieter Abbeel for helpful discussions, Gavin Cassidy and Luke Moss for their support in maintaining the Shadow Hand, and everybody at OpenAI for their help and support. We would also like to thank the following people for providing feedback on earlier versions of this manuscript: Pieter Abbeel, Joshua Achiam, Tamim Asfour, Aleksandar Botev, Greg Brockman, Rewon Child, Jack Clark, Marek Cygan, Harri Edwards, Ron Fearing, Ken Goldberg, Anna Goldie, Edward Mehr, Azalia Mirhoseini, Lerrel Pinto, Aditya Ramesh, Ian Rust, John Schulman, Shubho Sengupta, and Ilya Sutskever.

Funding

This research received no specific grant from any funding agency in the public, commercial, or not-for-profit sectors.

Supplementary material

A video summarizing our work can be found at: https://youtu.be/jwSbzNHGflM.

We have also made available the uncut and real-time video footage of a trial in which the robot hand successfully performed 50 consecutive rotations: https://youtu.be/DKe8FumoD4E.

We also gathered a representative set of videos from related work to illustrate the previous state of the art in dexterous manipulation: https://bit.ly/2uOK21Q.

Finally, we have released a supplementary blog post that summarizes the most important findings in an accessible manner: https://blog.openai.com/learning-dexterity/.

Notes

1. The environments we consider in this article are only partially observable.
2. A site represents a location of interest relative to the body frame in MuJoCo. See also http://mujoco.org/book/modeling.html#site.
3. We had trouble training in this environment from scratch, so we fine-tuned a policy trained in the original environment instead.
4. When training in an environment with no randomizations, the FF and LSTM policy converge to the same performance in the same amount of time. This shows that a FF policy has the capacity and observations to solve the non-randomized task but cannot solve it reliably with all randomizations, plausibly because it cannot adapt to the environment.
5. A sample contains three images of the same scene. We removed a few samples that had no object in them after it being dropped.
6. For comparison, PhaseSpace is rated for a position accuracy of around 20 μm but requires markers and a complex setup.
7. Some methods use iterative re-planning to partially mitigate this issue.

ORCID iDs

Arthur Petron https://orcid.org/0000-0001-9664-5097
Matthias Plappert https://orcid.org/0000-0002-0751-8094

References

Abadi M, Agarwal A, Barham P, et al. (2016) Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*.

Abell T and Erdmann MA (1995) Stably supported rotations of a planar polygon with two frictionless contacts. In: *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 1995)*, Pittsburgh, PA, 5–9 August 1995, pp. 411–418.

Aiyama Y, Inaba M and Inoue H (1993) Pivoting: A new method of graspless manipulation of object by robot fingers. In: *Proceedings of 1993 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 1993)*, Tokyo, Japan, 26–30 July 1993, pp. 136–143.

Antonova R, Cruciani S, Smith C and Kragic D. (2017) Reinforcement learning for pivoting task. *CoRR* abs/1703.00472.

Bai Y and Liu CK. (2014) Dexterous manipulation using both palm and fingers. In: *2014 IEEE International Conference on Robotics and Automation (ICRA 2014)*, Hong Kong, China, 31 May–7 June 2014, pp. 1560–1565.

Barth-Maron G, Hoffman MW, Budden D, et al. (2018) Distributed distributional deterministic policy gradients. *CoRR* abs/1804.08617.

Bertsekas DP. (2005) *Dynamic Programming and Optimal Control*, Vol. 1. Belmont, MA: Athena Scientific.

Bicchi A. (2000) Hands for dexterous manipulation and robust grasping: A difficult road toward simplicity. *IEEE Transactions on Robotics and Automation* 16(6): 652–662.

Bicchi A and Sorrentino R. (1995) Dexterous manipulation through rolling. In: *Proceedings of the 1995 International Conference on Robotics and Automation*, Nagoya, Aichi, Japan, 21–27 May 1995, pp. 452–457.

Brockman G, Cheung V, Pettersson L, et al. (2016) OpenAI GYM. *CoRR* abs/1606.01540.

Cherif M and Gupta KK. (1999) Planning quasi-static fingertip manipulations for reconfiguring objects. *IEEE Transactions Robotics and Automation* 15(5): 837–848.

Christiano PF, Shah Z, Mordatch I, et al. (2016) Transfer from simulation to real world through learning deep inverse dynamics model. *CoRR* abs/1610.03518.

Dafle NC and Rodriguez A. (2017) Sampling-based planning of in-hand manipulation with external pushes. *CoRR* abs/1707.00318.

Dafle NC, Rodriguez A, Paolini R, et al. (2014) Extrinsic dexterity: In-hand manipulation with external forces. In: *2014 IEEE International Conference on Robotics and Automation (ICRA 2014)*, Hong Kong, China, 31 May–7 June 2014, pp. 1578–1585.

Doulgeri Z and Droukas L (2013) On rolling contact motion by robotic fingers via prescribed performance control. In: *2013 IEEE International Conference on Robotics and Automation*, Karlsruhe, Germany, 6–10 May 2013, pp. 3976–3981.

Erdmann MA. (1998) An exploration of nonprehensile two-palm manipulation. *The International Journal of Robotics Research* 17(5): 485–503.

Erdmann MA and Mason MT. (1988) An exploration of sensorless manipulation. *IEEE Journal of Robotics and Automation* 4(4): 369–379.

Falco P, Attawia A, Saveriano M and Lee D. (2018) On policy learning robust to irreversible events: An application to robotic in-hand manipulation. *IEEE Robotics and Automation Letters* 3(3): 1482–1489.

Fearing RS (1986) Implementing a force strategy for object reorientation. In: *Proceedings of the 1986 IEEE International Conference on Robotics and Automation*, San Francisco, CA, 7–10 April 1986, pp. 96–102.

Feix T, Romero J, Schmiedmayer HB, Dollar A and Kragic D. (2016) The grasp taxonomy of human grasp types. *IEEE Transactions on Human–Machine Systems* 46(1): 66–77.

Finn C, Tan XY, Duan Y, Darrell T, Levine S and Abbeel P. (2015) Deep spatial autoencoders for visuomotor learning. *arXiv preprint arXiv:1509.06113*.

Gu S, Holly E, Lillicrap TP and Levine S (2017) Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates. In: *2017 IEEE International Conference on Robotics and Automation (ICRA 2017)*, Singapore, 29 May–3 June 2017, pp. 3389–3396.

Gupta A, Devin C, Liu Y, Abbeel P and Levine S. (2017) Learning invariant feature spaces to transfer skills with reinforcement learning. *CoRR* abs/1703.02949.

Han L, Guan Y, Li ZX, Qi S and Trinkle JC. (1997) Dextrous manipulation with rolling contacts. In: *Proceedings of the 1997 IEEE International Conference on Robotics and Automation*, Albuquerque, NM, 20–25 April 1997, pp. 992–997.

Han L and Trinkle JC. (1998) Dextrous manipulation by rolling and finger gaiting. In: *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA-98)*, Leuven, Belgium, 16–20 May 1998, pp. 730–735.

He K, Zhang X, Ren S and Sun J. (2016) Deep residual learning for image recognition. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778.

Hochreiter S and Schmidhuber J. (1997) Long short-term memory. *Neural Computation* 9(8): 1735–1780.

Huang WH and Mason MT. (2000) Mechanics, planning, and control for tapping. *The International Journal of Robotics Research* 19(10): 883–894.

Kalashnikov D, Irpan A, Pastor P, et al. (2018) QT-Opt: Scalable Deep Reinforcement Learning for Vision-Based Robotic Manipulation. *ArXiv e-prints*.

Kingma D and Ba J. (2014) Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Kumar V, Gupta A, Todorov E and Levine S. (2016a) Learning dexterous manipulation policies from experience and imitation. *CoRR* abs/1611.05095.

Kumar V, Todorov E and Levine S. (2016b) Optimal control with learned local models: Application to dexterous manipulation. In: *2016 IEEE International Conference on Robotics and Automation (ICRA 2016)*, Stockholm, Sweden, 16–21 May 2016, pp. 378–383.

Levine S and Koltun V. (2013) Guided policy search. In: *Proceedings of the 30th International Conference on Machine Learning (ICML 2013)*, Atlanta, GA, 16–21 June 2013, pp. 1–9.

Levine S, Pastor P, Krizhevsky A, Ibarz J and Quillen D. (2018) Learning hand–eye coordination for robotic grasping with deep learning and large-scale data collection. *The International Journal of Robotics Research* 37(4–5): 421–436.

Levine S, Wagener N and Abbeel P. (2015) Learning contact-rich manipulation skills with guided policy search. In: *IEEE International Conference on Robotics and Automation (ICRA 2015)*, Seattle, WA, 26–30 May 2015, pp. 156–163.

Li M, Bekiroglu Y, Kragic D and Billard A (2014a) Learning of grasp adaptation through experience and tactile sensing. In: *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Chicago, IL, 14–18 September 2014, pp. 3339–3346.

Li M, Yin H, Tahara K and Billard A (2014b) Learning object-level impedance control for robust grasping and dexterous manipulation. In: *2014 IEEE International Conference on Robotics and Automation (ICRA 2014)*, Hong Kong, China, 31 May–7 June 2014, pp. 6784–6791.

Li Q, Meier M, Haschke R, Ritter HJ and Bolder B. (2013) Rotary object dexterous manipulation in hand: A feedback-based method. *IJMA* 3(1): 36–47.

Ma RR and Dollar AM (2011) On dexterity and dexterous manipulation. In: *15th International Conference on Advanced Robotics: New Boundaries for Robotics (ICAR 2011)*, Tallinn, Estonia, 20–23 June 2011, pp. 1–7.

Mahler J, Liang J, Niyaz S, et al. (2017a) Dex-net 2.0: Deep learning to plan robust grasps with synthetic point clouds and analytic grasp metrics. In: *Robotics: Science and Systems XIII*, Massachusetts Institute of Technology, Cambridge, MA, 12–16 July 2017.

Mahler J, Matl M, Liu X, Li A, Gealy DV and Goldberg K. (2017b) Dex-net 3.0: Computing robust robot suction grasp targets in point clouds using a new analytic model and deep learning. *CoRR* abs/1709.06670.

Marcin A, Bowen B, Maciek C, et al. (2018) *OpenAI Five*. https://blog.openai.com/openai-five/.

Mordatch I, Popovic Z and Todorov E (2012) Contact-invariant optimization for hand manipulation. In: *Proceedings of the 2012 Eurographics/ACM SIGGRAPH Symposium on Computer Animation (SCA 2012)*, Lausanne, Switzerland, 2012, pp. 137–144.

Nair V and Hinton GE. (2010) Rectified linear units improve restricted Boltzmann machines. In: *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pp. 807–814.

Okamura AM, Smaby N and Cutkosky MR (2000) An overview of dexterous manipulation. In: *Proceedings of the 2000 IEEE International Conference on Robotics and Automation (ICRA 2000)*, San Francisco, CA, 24–28 April 2000, pp. 255–262.

OpenAI (2018) OpenAI Five. https://blog.openai.com/openai-five/.

Pehoski C, Henderson A and Tickle-Degnen L. (1997) In-hand manipulation in young children: Rotation of an object in the fingers. *American Journal of Occupational Therapy* 51(7): 544–552.

Peng XB, Andrychowicz M, Zaremba W and Abbeel P. (2017) Sim-to-real transfer of robotic control with dynamics randomization. *CoRR* abs/1710.06537.

Pinto L, Andrychowicz M, Welinder P, Zaremba W and Abbeel P. (2017a) Asymmetric actor critic for image-based robot learning. *arXiv preprint arXiv:1710.06542*.

Pinto L, Davidson J and Gupta A (2017b) Supervision via competition: Robot adversaries for learning tasks. In: *2017 IEEE International Conference on Robotics and Automation (ICRA 2017)*, Singapore, 29 May–3 June 2017, pp. 1601–1608.

Pinto L, Davidson J, Sukthankar R and Gupta A. (2017c) Robust adversarial reinforcement learning. In: *Proceedings of the 34th International Conference on Machine Learning (ICML 2017)*, Sydney, NSW, Australia, 6–11 August 2017, pp. 2817–2826.

Plappert M, Andrychowicz M, Ray A, et al. (2018) Multi-goal reinforcement learning: Challenging robotics environments and request for research. *arXiv preprint arXiv:1802.09464*.

Rajeswaran A, Kumar V, Gupta A, Schulman J, Todorov E and Levine S. (2017) Learning complex dexterous manipulation with deep reinforcement learning and demonstrations. *CoRR* abs/1709.10087.

Rus D (1992) Dexterous rotations of polyhedra. In: *Proceedings of the 1992 IEEE International Conference on Robotics and Automation*, Nice, France, 12–14 May 1992, pp. 2758–2763.

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

Rusu AA, Vecerik M, Rothörl T, Heess N, Pascanu R and Hadsell R. (2017) Sim-to-real robot learning from pixels with progressive nets. In: *Proceedings 1st Annual Conference on Robot Learning (CoRL 2017)*, Mountain View, CA, 13–15 November 2017, pp. 262–270.

Sadeghi F and Levine S. (2017) CAD2RL: Real single-image flight without a single real image. In: *Robotics: Science and Systems XIII*, Massachusetts Institute of Technology, Cambridge, MA, 12–16 July 2017.

Sawasaki N and Inoue H. (1991) Tumbling objects using a multi-fingered robot. *Journal of the Robotics Society of Japan* 9(5): 560–571.

Schulman J, Moritz P, Levine S, Jordan M and Abbeel P. (2015) High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*.

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

ShadowRobot (2005) ShadowRobot Dexterous Hand. https://www.shadowrobot.com/products/dexterous-hand/.

Shi J, Woodruff JZ, Umbanhowar PB and Lynch K. (2017) Dynamic in-hand sliding manipulation. *IEEE Transactions on Robotics* 33(4): 778–795.

Sutton RS and Barto AG. (1998) *Reinforcement Learning: An Introduction*. Cambridge, MA: MIT Press.

Tahara K, Arimoto S and Yoshida M. (2010) Dynamic object manipulation using a virtual frame by a triple soft-fingered robotic hand. In: *IEEE International Conference on Robotics and Automation (ICRA 2010)*, Anchorage, AK, 3–7 May 2010, pp. 4322–4327.

Tan J, Zhang T, Coumans E, et al. (2018) Sim-to-real: Learning agile locomotion for quadruped robots. *CoRR* abs/1804.10332.

Tobin J, Fong R, Ray A, Schneider J, Zaremba W and Abbeel P. (2017a) Domain randomization for transferring deep neural networks from simulation to the real world. *arXiv preprint arXiv:1703.06907*.

Tobin J, Zaremba W and Abbeel P. (2017b) Domain randomization and generative models for robotic grasping. *CoRR* abs/1710.06425.

Todorov E, Erez T and Tassa Y. (2012) Mujoco: A physics engine for model-based control. In: *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). IEEE*, pp. 5026–5033.

Tournassoud P, Lozano-Pérez T and Mazer E. (1987) Regrasping. In: *Proceedings of the 1987 IEEE International Conference on Robotics and Automation*, Raleigh, NC, 31 March–3 April 1987, pp. 1924–1928.

Tzeng E, Devin C, Hoffman J, et al. (2015) Towards adapting deep visuomotor representations from simulated to real environments. *CoRR* abs/1511.07111.

Unity Technologies (2005) Unity game engine. http://unity3d.

van Hoof H, Hermans T, Neumann G and Peters J (2015) Learning robot in-hand manipulation with tactile features. In: *15th IEEE-RAS International Conference on Humanoid Robots (Humanoids 2015)*, Seoul, South Korea, 3–5 November 2015, pp. 121–127.

Yu W, Tan J, Liu CK and Turk G (2017) Preparing for the unknown: Learning a universal policy with online system identification. In: *Robotics: Science and Systems XIII*, Massachusetts Institute of Technology, Cambridge, MA, 12–16 July 2017.

Zhu Y, Wang Z, Merel J, et al. (2018) Reinforcement and imitation learning for diverse visuomotor skills. *CoRR* abs/1802.09564.