https://openai.com/blog/learning-dexterity/

OpenAI

ULY 30, 2018 • 9 MINUTE READ

Learning Dexterity

We've trained a human-like robot hand to manipulate physical objects with unprecedented dexterity.

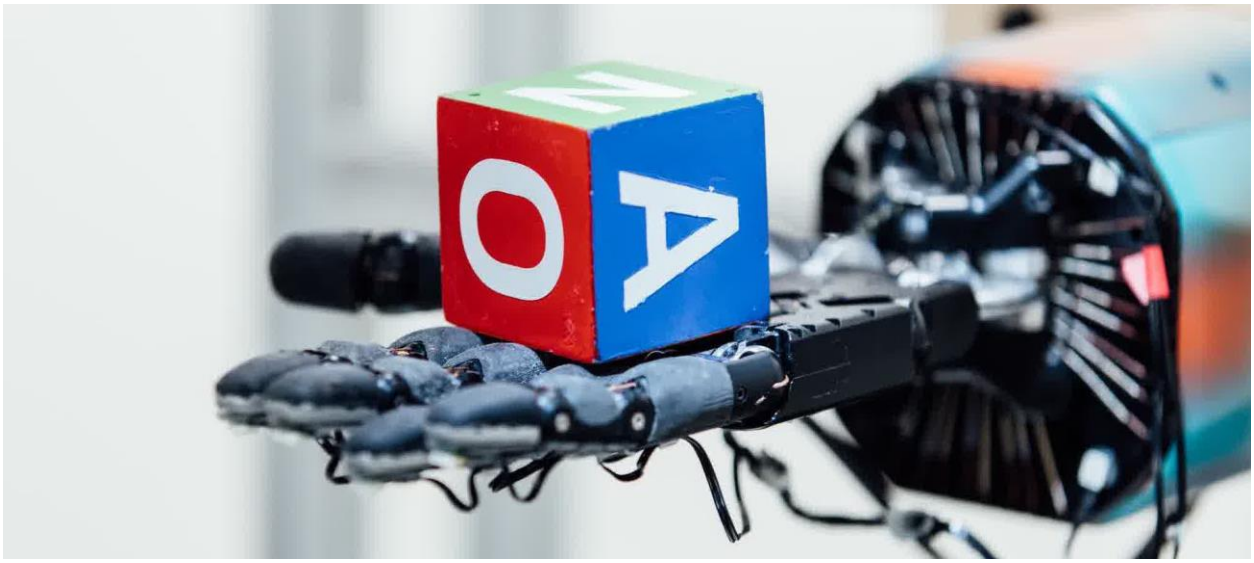
Our system, called Dactyl, is trained entirely in simulation and transfers its knowledge to reality, adapting to real-world physics using techniques we've been working on for the past year. Dactyl learns from scratch using the same general-purpose reinforcement learning algorithm and code as OpenAI Five. Our resultsshow that it's possible to train agents in simulation and have them solve real-world tasks, without physically-accurate modeling of the world.

READ PAPER

Examples of dexterous manipulation behaviors autonomously learned by Dactyl.

The task

Dactyl is a system for manipulating objects using a Shadow Dexterous Hand. We place an object such as a block or a prism in the palm of the hand and ask Dactyl to reposition it into a different orientation; for example, rotating the block to put a new face on top. The network observes only the coordinates of the fingertips and the images from three regular RGB cameras.



Although the first humanoid hands were developed decades ago, using them to manipulate objects effectively has been a long-standing challenge in robotic control. Unlike other problems such as locomotion, progress on dextrous manipulation using traditional robotics approaches has been slow, and current techniques remain limited in their ability to manipulate objects in the real world.

Reorienting an object in the hand requires the following problems to be solved:

- **Working in the real world.** Reinforcement learning has shown many successes in simulations and video games, but has had comparatively limited results in the real world. We test Dactyl on a physical robot.
- **High-dimensional control.** The Shadow Dexterous Hand has 24 degrees of freedom compared to 7 for a typical robot arm.
- **Noisy and partial observations.** Dactyl works in the physical world and therefore must handle noisy and delayed sensor readings. When a fingertip sensor is occluded by other fingers or by the object, Dactyl has to work with partial information. Many aspects of the physical system like friction and slippage are not directly observable and must be inferred.
- **Manipulating more than one object.** Dactyl is designed to be flexible enough to reorient multiple kinds of objects. This means that our approach cannot use strategies that are only applicable to a specific object geometry.

Our approach

Dactyl learns to solve the object reorientation task entirely in simulation without any human input. After this training phase, the learned policy works on the real robot without any fine-tuning.

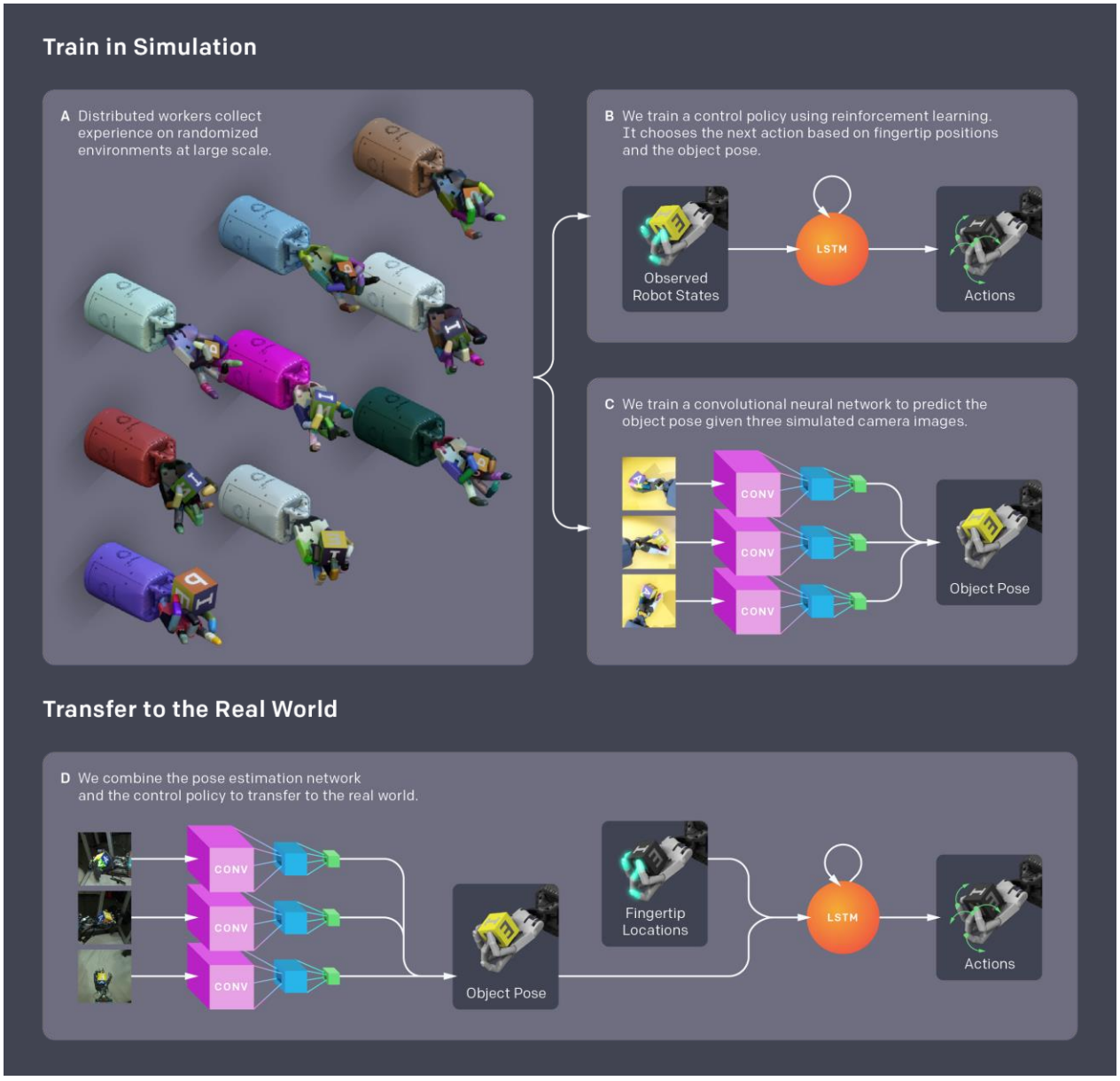
Dactyl achieving the tested maximum of 50 consecutive rotations. Real-time.

Learning methods for robotic manipulation face a dilemma. Simulated robots can easily provide enough data to train complex policies, but most manipulation problems can't be modeled accurately enough for those policies to transfer to real robots. Even modeling what happens when two objects touch — the most basic problem in manipulation — is an active area of research with no widely accepted solution. Training directly on physical robots allows the policy to learn from real-world physics, but today's algorithms would require years of experience to solve a problem like object reorientation.

Our approach, *domain randomization*, learns in a simulation which is designed to provide a variety of experiences rather than maximizing realism. This gives us the best of both approaches: by learning in simulation, we can gather more experience quickly by scaling up, and by de-emphasizing realism, we can tackle problems that simulators can only model approximately.

It's been shown (by OpenAI and others) that domain randomization can work on increasingly complex problems — domain randomizations were even used to train OpenAI Five. Here, we

wanted to see if scaling up domain randomization could solve a task well beyond the reach of current methods in robotics.



We built a simulated version of our robotics setup using the MuJoCo physics engine. This simulation is only a coarse approximation of the real robot:

- Measuring physical attributes like friction, damping, and rolling resistance is cumbersome and difficult. They also change over time as the robot experiences wear and tear.
- MuJoCo is a rigid body simulator, which means that it cannot simulate the deformable rubber found at the fingertips of the hand or the stretching of tendons.
- Our robot can only manipulate the object by repeatedly making contact with it. However, contact forces are notoriously difficult to reproduce accurately in simulation.

The simulation can be made more realistic by calibrating its parameters to match robot behavior, but many of these effects simply cannot be modeled accurately in current simulators.

Instead, we train the policy on a distribution of simulated environments where the physical and visual attributes are chosen randomly. Randomized values are a natural way to represent the uncertainties that we have about the physical system and also prevent overfitting to a single simulated environment. If a policy can accomplish the task across all of the simulated environments, it will more likely be able to accomplish it in the real world.

Learning to control

By building simulations that support transfer, we have reduced the problem of controlling a robot in the real world to accomplishing a task in simulation, which is a problem well-suited for reinforcement learning. While the task of manipulating an object in a simulated hand is already somewhat difficult, learning to do so across all combinations of randomized physical parameters is substantially more difficult.

To generalize across environments, it is helpful for the policy to be able to take different actions in environments with different dynamics. Because most dynamics parameters cannot be inferred from a single observation, we used an LSTM — a type of neural network with memory — to make it possible for the network to learn about the dynamics of the environment. The LSTM achieved about twice as many rotations in simulation as a policy without memory.

Dactyl learns using Rapid, the massively scaled implementation of Proximal Policy Optimization developed to allow OpenAI Five to solve Dota 2. We use a different model architecture, environment, and hyperparameters than OpenAI Five does, but we use the same algorithms and training code. Rapid used 6144 CPU cores and 8 GPUs to train our policy, collecting about one hundred years of experience in 50 hours.

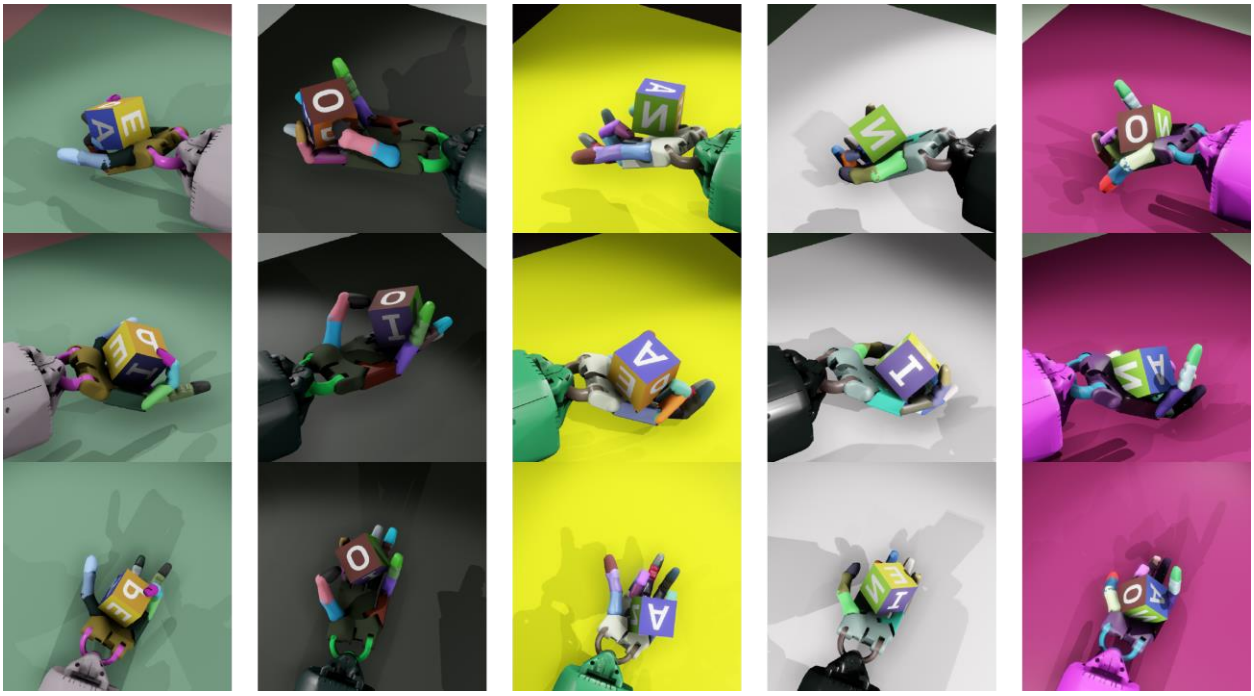
For development and testing, we validated our control policy against objects with embedded motion tracking sensors to isolate the performance of our control and vision networks.

Learning to see

Dactyl was designed to be able to manipulate arbitrary objects, not just those that have been specially modified to support tracking. Therefore, Dactyl uses regular RGB camera images to estimate the position and orientation of the object.

We train a pose estimator using a convolutional neural network. The neural network takes the video streams from three cameras positioned around the robot hand and outputs the estimated position and orientation of the object. We use multiple cameras to resolve ambiguities and occlusion. We again use domain randomization to train this network only in simulation using the Unity game development platform, which can model a wider variety of visual phenomena than Mujoco.

By combining these two independent networks, the control network that reorients the object given its pose and the vision network that maps images from cameras to the object's pose, Dactyl can manipulate an object by seeing it.



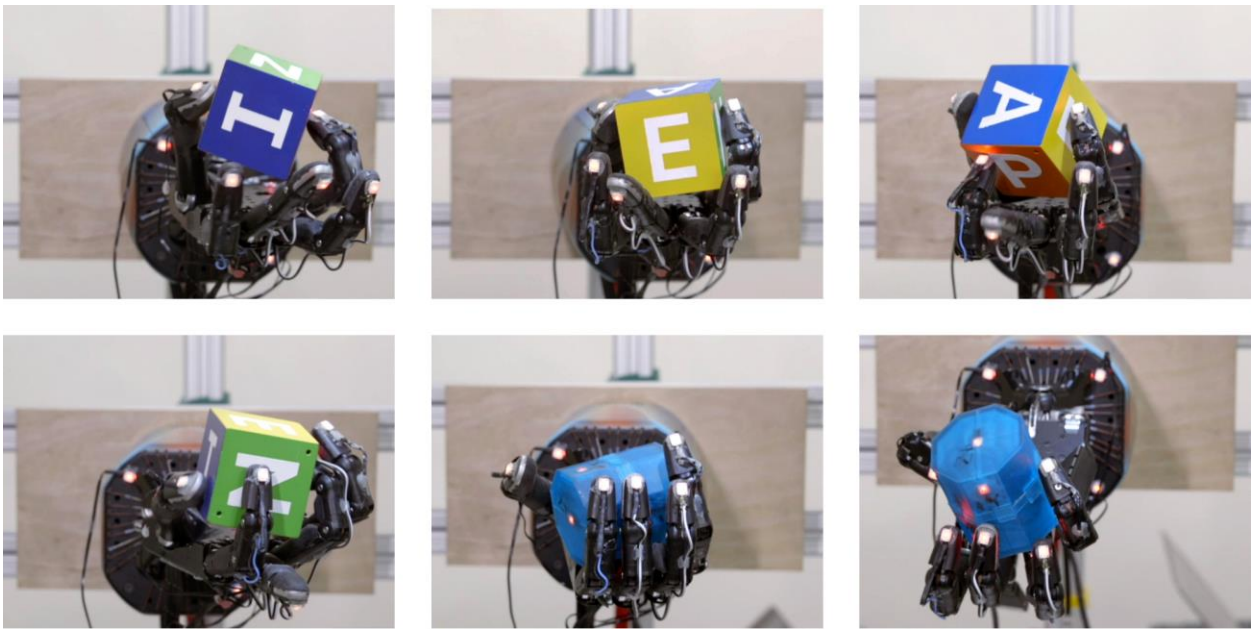
Example training images used for learning to estimate the pose of the block.

Results

Emergent behaviors

When deploying our system, we noticed that Dactyl uses a rich set of in-hand dexterous manipulation strategies to solve the task. These strategies are commonly used by humans as well. However, we do not teach them to our system explicitly; all behaviors are discovered autonomously.

Examples of dexterous manipulation behaviors autonomously learned by Dactyl.

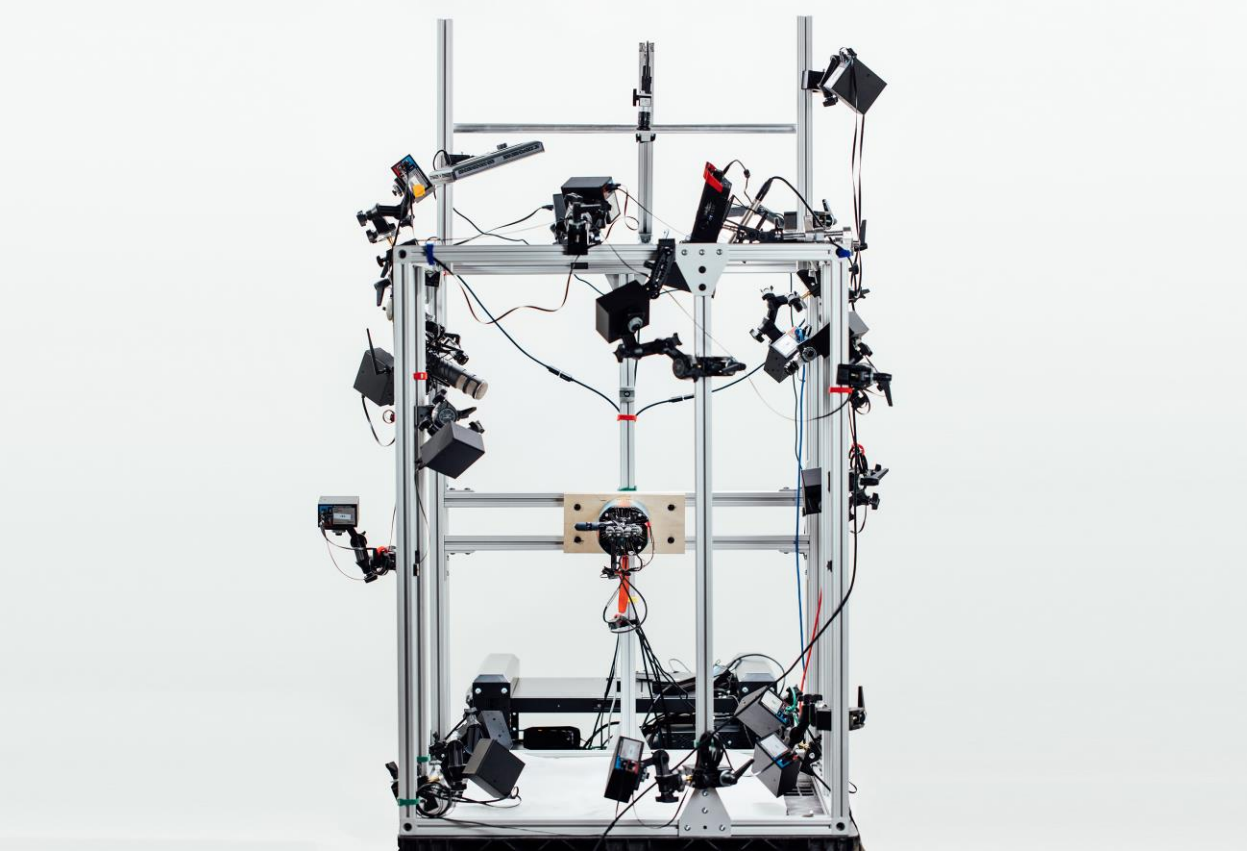


Dactyl grasp types according to the GRASP taxonomy. Top left to bottom right: Tip Pinch, Palmar Pinch, Tripod, Quadpod, Power grasp, and 5-Finger Precision grasp.

We observed that for precision grasps, such as the Tip Pinch grasp, Dactyl uses the thumb and little finger. Humans tend to use the thumb and either the index or middle finger instead. However, the robot hand's little finger is more flexible due to an extra degree of freedom, which may explain why Dactyl prefers it. This means that Dactyl can rediscover grasps found in humans, but adapt them to better fit the limitations and abilities of its own body.

Transfer performance

We tested how many rotations Dactyl could achieve before it dropped the object, timed out, or reached 50 successes. Our policies trained purely in simulation were able to successfully manipulate objects in the real world.



Dactyl lab setup with Shadow Dexterous Hand, PhaseSpace motion tracking cameras, and Basler RGB cameras.

For the task of block manipulation, policies trained with randomization could achieve many more rotations than those trained without randomization, as can be seen in the results below. Also, using the control network with pose estimated from vision performs nearly as well as reading the pose directly from motion tracking sensors.

RANDOMIZATIONS	OBJECT TRACKING	MAX NUMBER OF SUCCESSES	MEDIAN NUMBER OF SUCCESSES
All randomizations	Vision	**46**	**11.5**
All randomizations	Motion tracking	**50**	**13**
No randomizations	Motion tracking	**6**	**0**

Learning progress

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 similar performance in a fully randomized simulation requires about 100 years of experience.

Learning progress with and without randomizations over years of simulated experience.

What surprised us

- **Tactile sensing is not necessary to manipulate real-world objects.** Our robot receives only the locations of the five fingertips along with the position and orientation of the cube. Although the robot hand has touch sensors on its fingertips, we didn't need to use them. Generally, we found better performance from using a limited set of sensors that could be modeled effectively in the simulator instead of a rich sensor set with values that were hard to model.
- **Randomizations developed for one object generalize to others with similar properties.** After developing our system for the problem of manipulating a block, we printed an octagonal prism, trained a new policy using its shape, and attempted to manipulate it. Somewhat to our surprise, it achieved high performance using only the randomizations we had designed for the block. By contrast, a policy that manipulated a sphere could only achieve a few successes in a row, perhaps because we had not randomized any simulation parameters that model rolling behavior.
- **With physical robots, good systems engineering is as important as good algorithms.** At one point, we noticed that one engineer consistently achieved much better performance than others when running the exact same policy. We later discovered that he had a faster laptop, which hid a timing bug that reduced performance. After the bug was fixed, performance improved for the rest of the team.

What didn't pan out

We also found to our surprise that a number of commonly employed techniques did not improve our results.

- **Decreasing reaction time did not improve performance.** Conventional wisdom states that reducing the time between actions should improve performance because the changes between states are smaller and therefore easier to predict. Our current time between actions is 80ms, which is smaller than human reaction time of 150-250ms, but significantly larger than neural network computation time of roughly 25ms. Surprisingly, decreasing time between actions to 40ms required additional training time but did not noticeably improve performance in the real world. It's possible that this rule of thumb is less applicable to neural network models than to the linear models that are in common use today.
- **Using real data to train our vision policies didn't make a difference.** In early experiments, we used a combination of simulated and real data to improve our models. The real data was gathered from trials of our policy against an object with embedded tracking markers. However, real data has significant disadvantages compared to simulated data. Position information from tracking markers has latency and measurement error. Worse, real data is easily invalidated by common configuration changes, making it a hassle to collect enough to be useful. As our methods developed, our simulator-only

error improved until it matched our error from using a mixture of simulated and real data. Our final vision models were trained without real data.

This project completes a full cycle of AI development that OpenAI has been pursuing for the past two years: we've developed a new learning algorithm, scaled it massively to solve hard simulated tasks, and then applied the resulting system to the real world. Repeating this cycle at increasing scale is the primary route we are pursuing to increase the capabilities of today's AI systems towards safe artificial general intelligence. If you'd like to be part of what comes next, we're hiring!

Authors
Peter WelinderAlex RayArthur PetronLilian WengJosh TobinRafał JózefowiczMatthias PlappertMarcin AndrychowiczJakub PachockiJonas SchneiderSzymon SidorBowen Baker
(ORDER RANDOMIZED EACH PAGELOAD)

Contributors
Larissa Schiavo, Jack Clark, Greg Brockman, Ilya Sutskever, Diane Yoon & Ben Barry

---

Acknowledgments
Thanks to the following for feedback on drafts of this post: Pieter Abbeel, Tamim Asfour, Marek Cygan, Ken Goldberg, Anna Goldie, Edward Mehr, Azalia Mirhoseini, Lerrel Pinto, Aditya Ramesh & Ian Rust.

Cover Artwork
Ben Barry & Eric Haines