UC San Diego

Learning Dexterous In-Hand Manipulation

Presenter: **SIDDARTH** Meenakshi Sundaram

21 May 2020

Outline

- Introduction
- The Task
- Method/ Experimentation
  - Training in Simulation (Transferable Simulations)
  - Learning Control Policy
  - State Estimation from Vision
- Results
- Conclusion

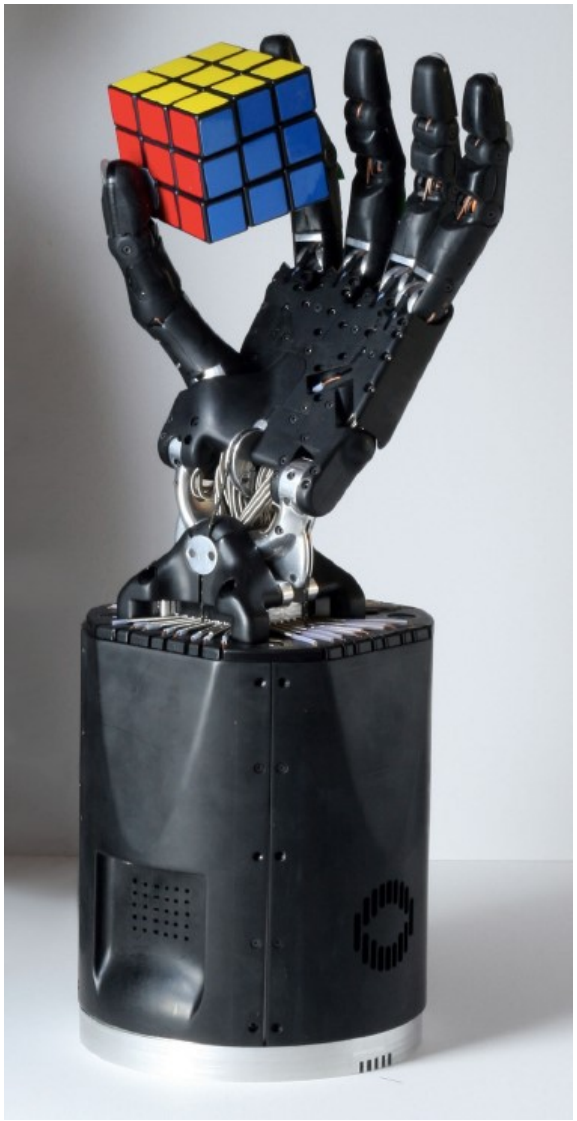
Introduction/Challenges

- Dexterous manipulation of objects is a fundamental everyday task for humans. But still challenging for autonomous robots

- Robots are typically designed for specific tasks in constrained settings and are largely unable to utilize complex end-effectors

- People are able to perform a wide range of dexterous manipulation tasks in a diverse set of environments

Introduction/Challenges



SETUP

- The Shadow Dexterous Hand is an example of a robotic hand designed for human-level dexterity; it has five fingers with a total of 24 degrees of freedom.
- 40 tendons controlled by 20 DC motors in the base of the hand
- 16 degrees of freedom can be controlled independently whereas the remaining 8 joints form 4 pairs of coupled joints

Introduction/Challenges

- NOT adopted widely. Can be attributed to the 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

- Hence, the task is to demonstrate methods to train control policies that perform in-hand manipulation and deploy them on a physical robot.

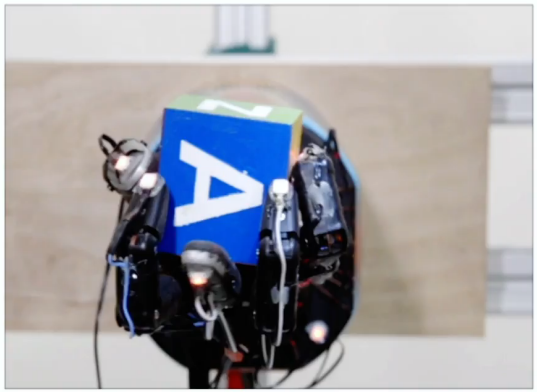
The Task

- We place an object such as a block or a prism in the palm of the hand and ask the system to reposition it into a different orientation;

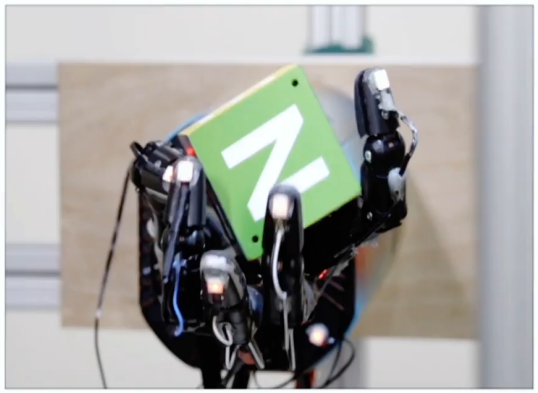
- For example, rotating the block to put a new face on top, or finger-gaiting to change the side of the robot without changing the top or the bottom of the block.

- The network observes only the coordinates of the fingertips and the images from three regular RGB cameras.

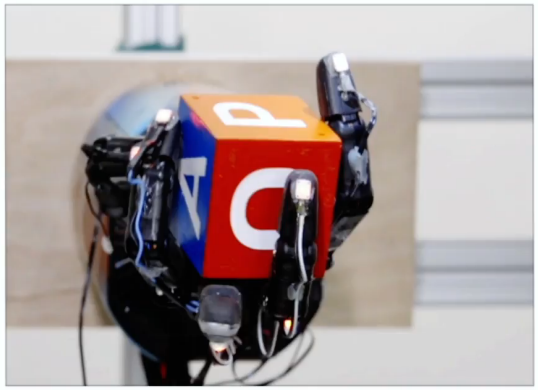
The Task



FINGER PIVOTING



SLIDING



FINGER GAITING

The Task
(Re-orienting an hand)

PROBLEMS TO BE SOLVED:

- **Working in the real world.**
- **High-dimensional control.**
- **Noisy and partial observations.**
- **Manipulating more than one object**

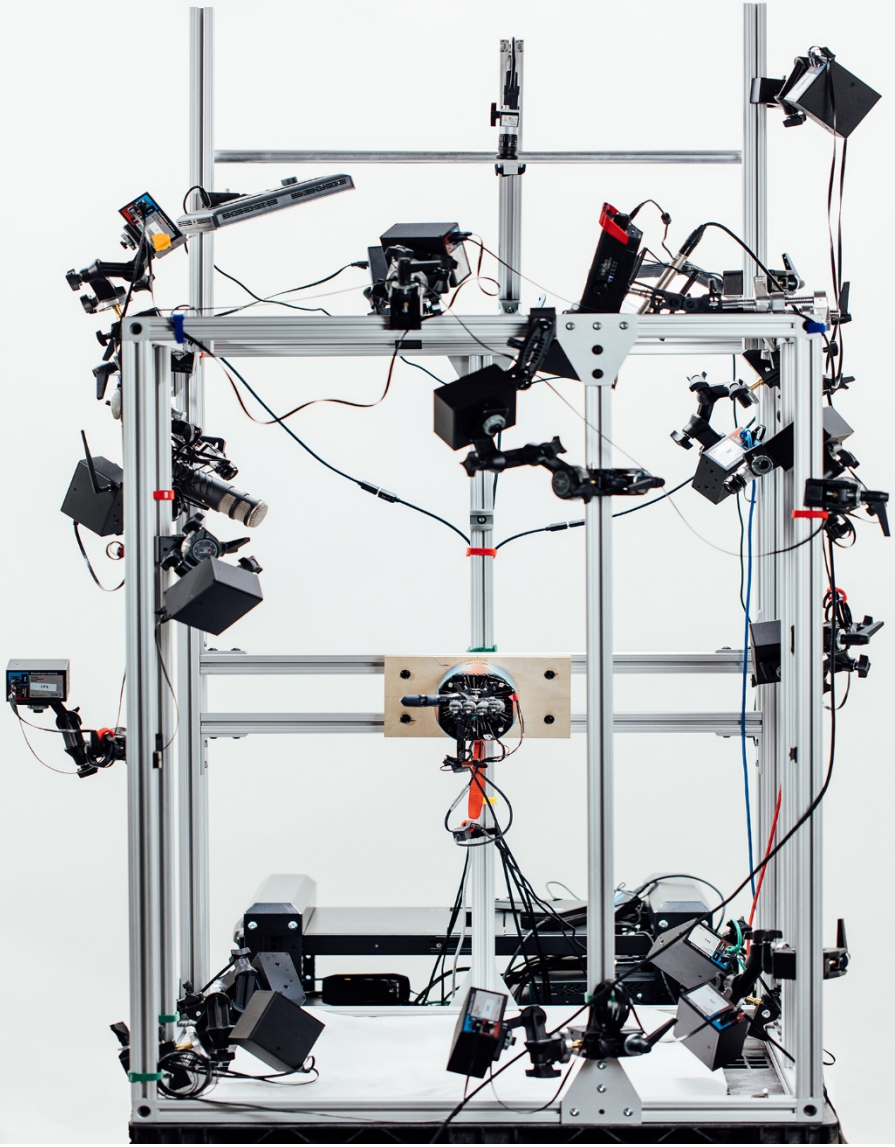
Method – [TRAINING IN SIMULATION]

Training in Simulation Consists of

- Observation
- Randomization

AIM is to collect as much experience as possible

Method - Part 1 - Observation



PhaseSpace motion tracking cameras, and Basler RGB cameras.

Method - Part 1 - Observation

- 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

Method – part 2 - Randomization

- **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.

- **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 which is sampled once per episode as well as an uncorrelated noise sampled at every timestep

Method – part 2 - Randomization

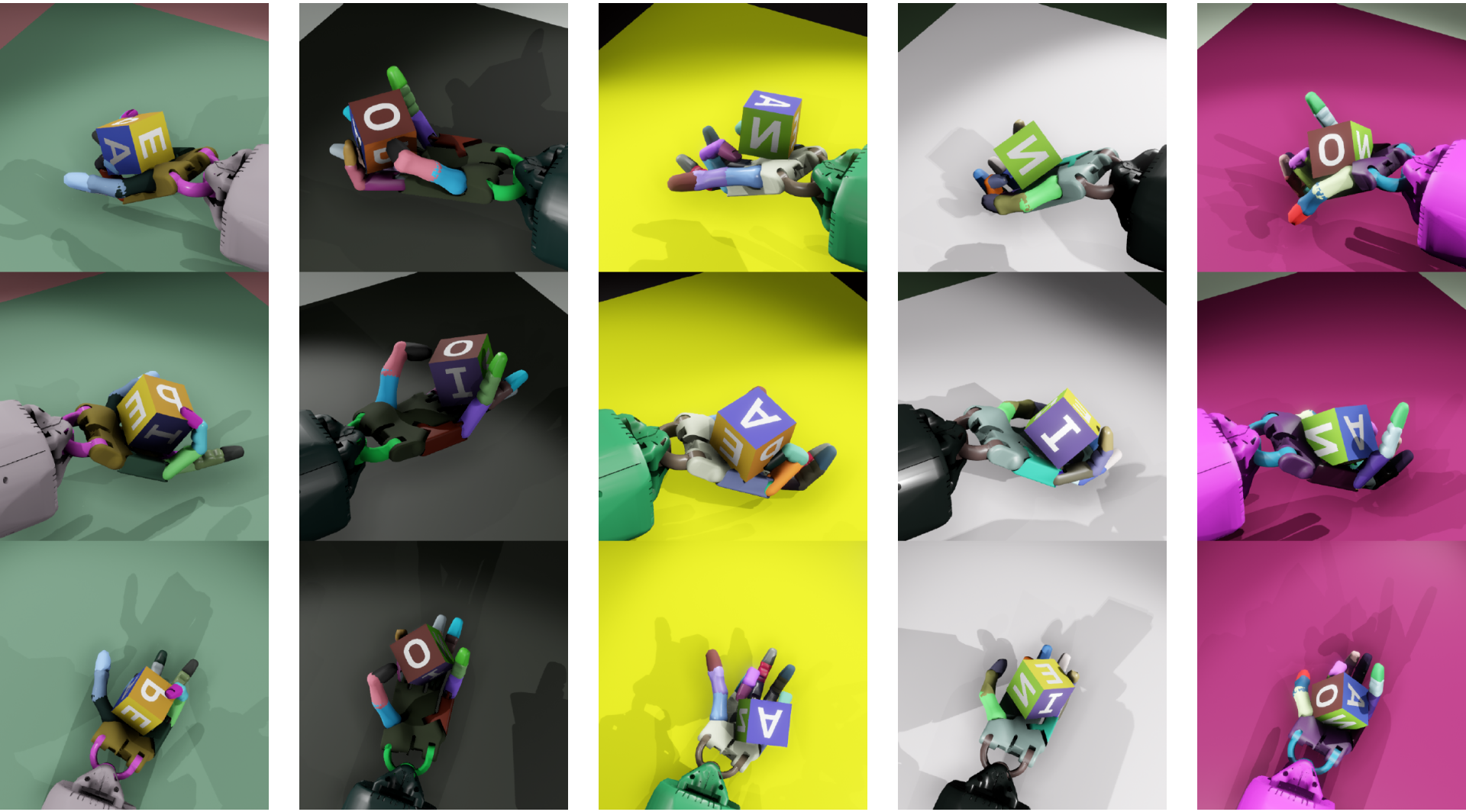
- **Unmodeled effects**

The physical robot experiences many effects that are not modeled by our simulation. To account for imperfect actuation, we use a simple model of motor backlash and introduce action delays and action noise before applying them in simulation. We also simulate marker occlusion by freezing its simulated position whenever it is close to another marker or the object. To handle additional unmodeled dynamics, we apply small random forces to the object.

- **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.

Method – part 2 - Randomization



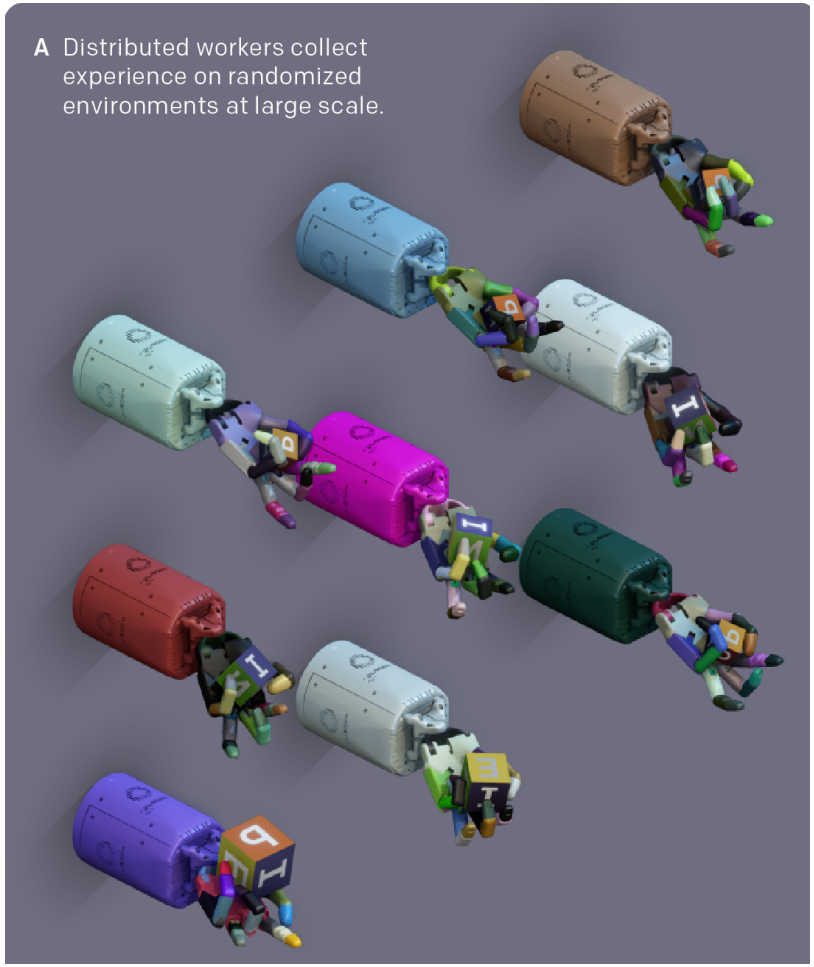
Method – part 2 - Randomization

- **Physics randomization**

Table 1: Ranges of physics parameter randomizations.

Parameter	Scaling factor range	Additive term 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])	
joint limits		$\mathcal{N}(0, 0.15)$ rad
gravity vector (each coordinate)		$\mathcal{N}(0, 0.4)$ m/s^2

Physical parameters like friction are randomized at the beginning of every episode and held fixed. Many parameters are centered on values found during model calibration in an effort to make the simulation distribution match reality more closely



After the completion of this step, Experience is collected.

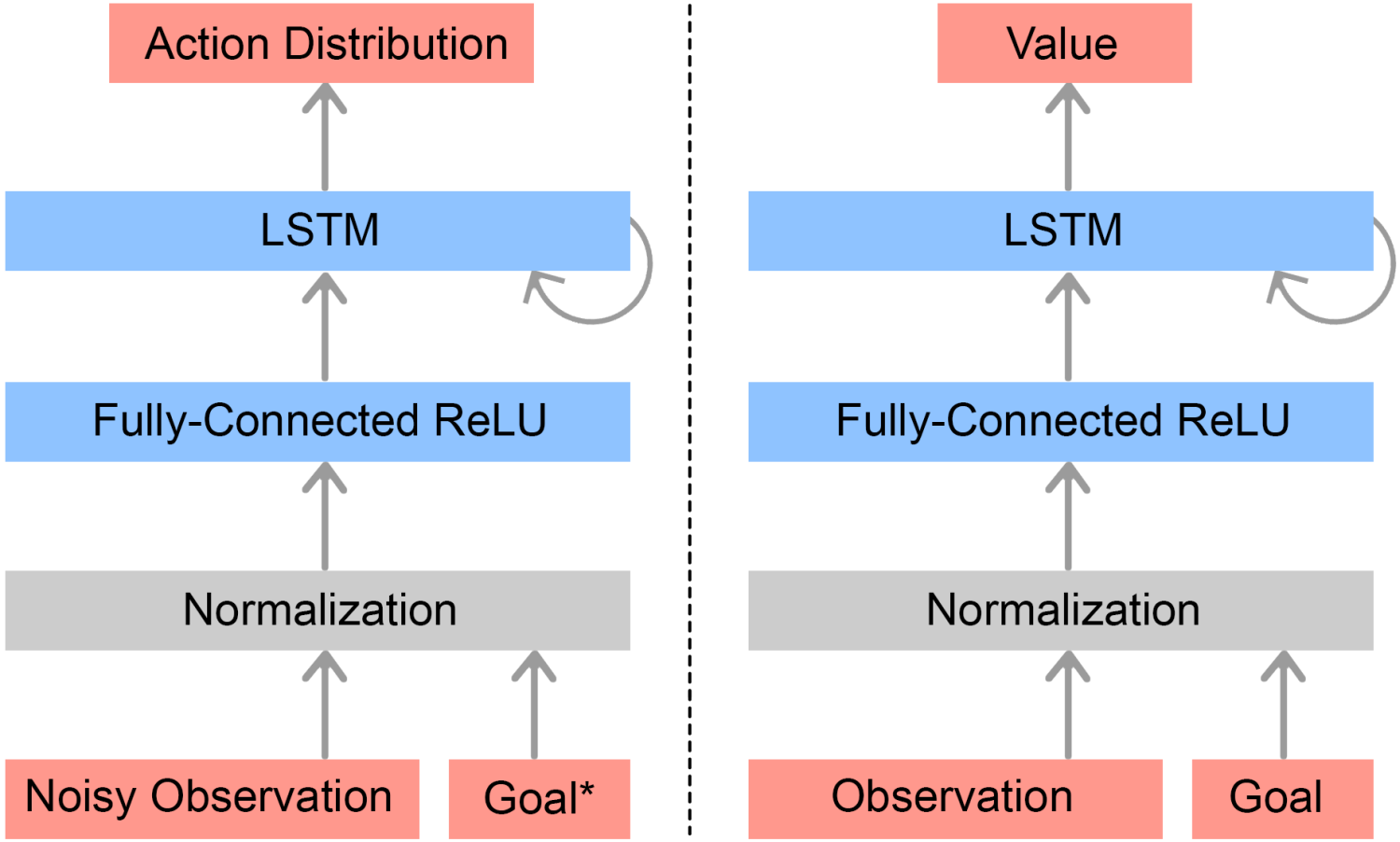
Method – Learning Control Policy

- Policy Architecture
- Actions and Rewards
- Distributed Training

Method – Learning Control Policy

- **Policy Architecture:**

We use Proximal policy optimization:



Method – Learning Control Policy

- **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 with an additional hidden layer with ReLU activations inserted between inputs and the LSTM.

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

Method – Learning Control Policy

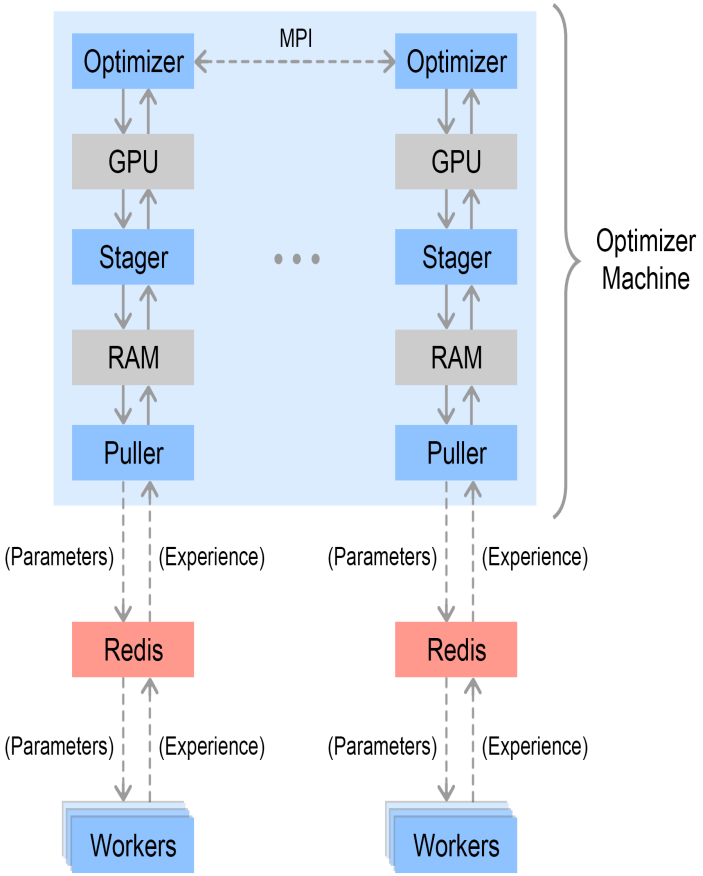
- **Actions and Rewards**

Policy actions correspond to desired joints angles relative to the current ones5 (e.g. rotate this joint by 10 degrees). While PPO can handle both continuous and discrete action spaces, we noticed that discrete action spaces work much better. This may be because a discrete probability distribution is more expressive than a multivariate Gaussian or because discretization of actions makes learning a good advantage function potentially simpler. We discretize each action coordinate into 11 bins.

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 and a reward of - 20 (a penalty) whenever the object is dropped

Method – Learning Control Policy

- **Distributed Training:**



distributed training infrastructure in Rapid (same as the one in OpenAI Five)

Method – Learning Control Policy

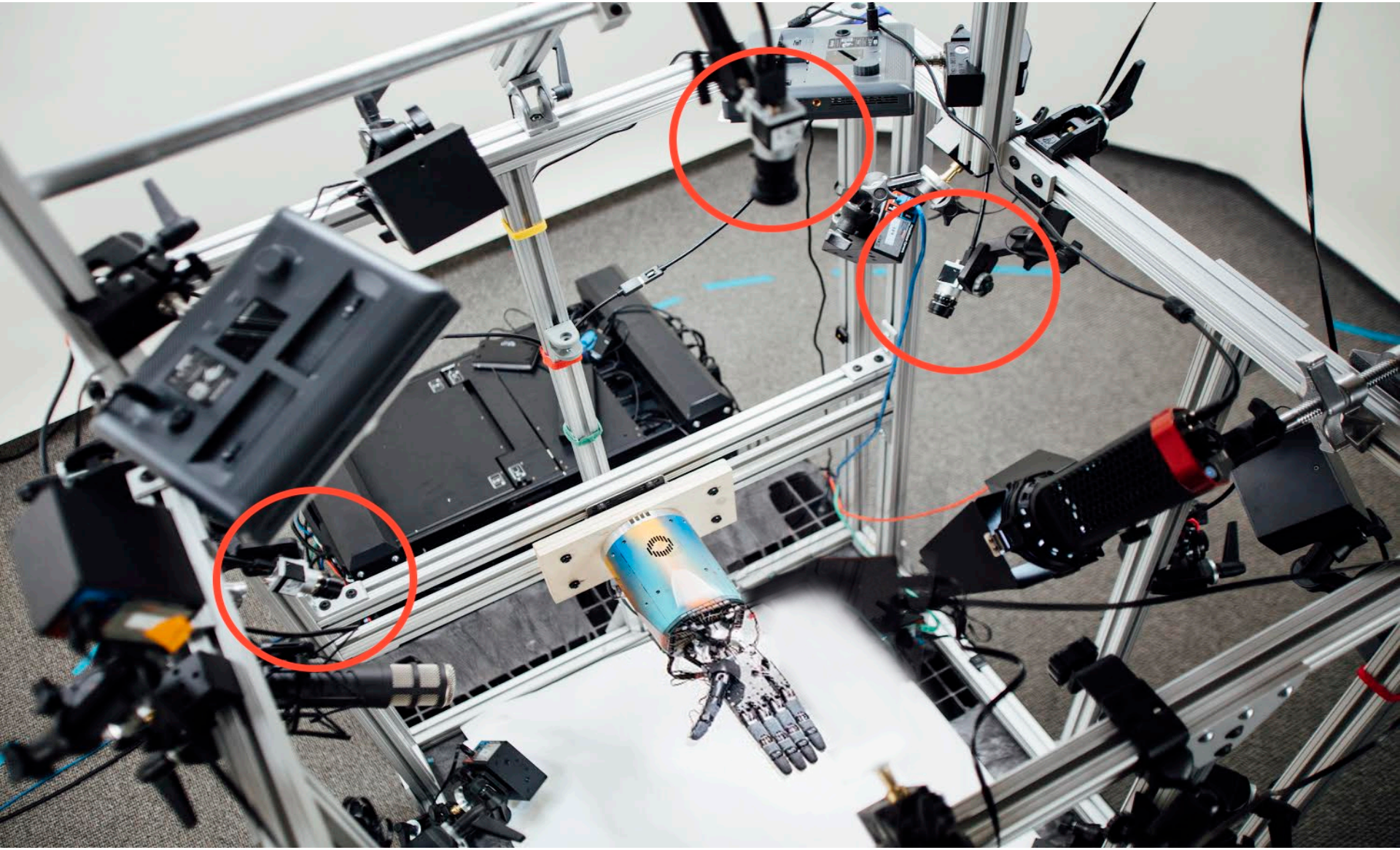
- **Distributed Training**

  We use the same distributed implementation of PPO that was used to train OpenAI Five without modifications. Overall, we found that PPO scales up easily and requires little hyperparameter tuning.

  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 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. This setup allows us to generate about 2 years of simulated experience per hour.

State Estimation from Vision



3-camera setup for vision-based state estimation.

State Estimation from Vision

The policy that we describe 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.
[MOSTLY will be explained by me after removing most of this TEXT above]

So, the State Estimation from Vision, we need to now understand the

- Model Architecture

AND

- Training

State Estimation from Vision

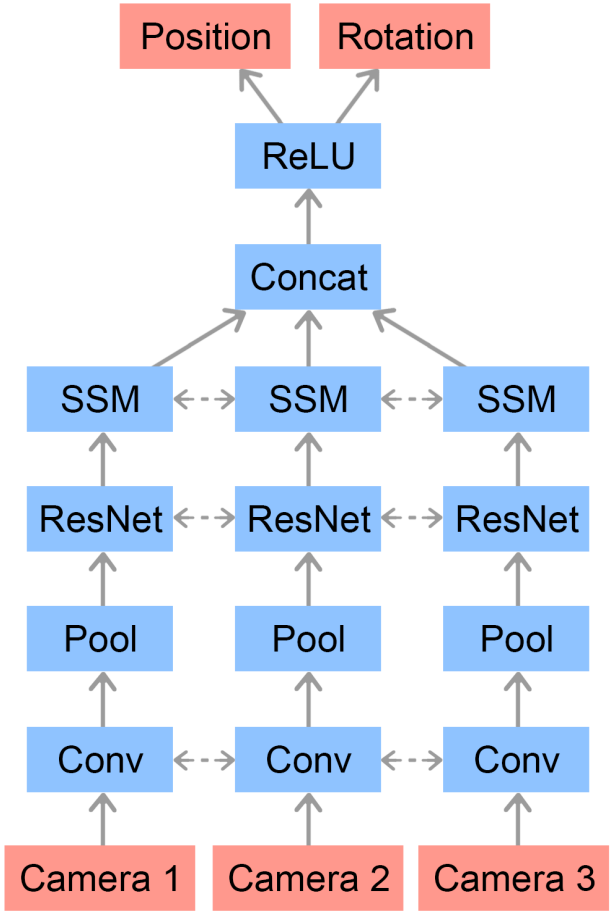
- **Model Architecture**

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, which is depicted in Figure next slide. 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.

State Estimation from Vision

- **Model Architecture**

Vision network architecture. Camera images are passed through a convolutional feature stack that consists of two convolutional layers, max-pooling, ResNet blocks, and spatial softmax(SSM) 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 activation function. Linear outputs from the last layer form the estimates of the position and orientation of the object.



State Estimation from Vision

- **Training**

We run the trained policy in the simulator until we gather one million states. We then train the vision network by minimizing the mean squared error between the normalized prediction and the ground truth with mini-batch gradient descent.

For each mini-batch, we render the images with randomized appearance before feeding them to the network. Moreover, we augment the data by modifying the object pose. We use 2 GPUs for rendering and 1 GPU for running the network and training.

Results

- Quantitative Results

- Qualitative Results

(Should I include these three below?)

- Effect of Memory?
- Complexity and Scale?
- Vision Performance?

Results

- **Quantitative Results:**

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 is measured.

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

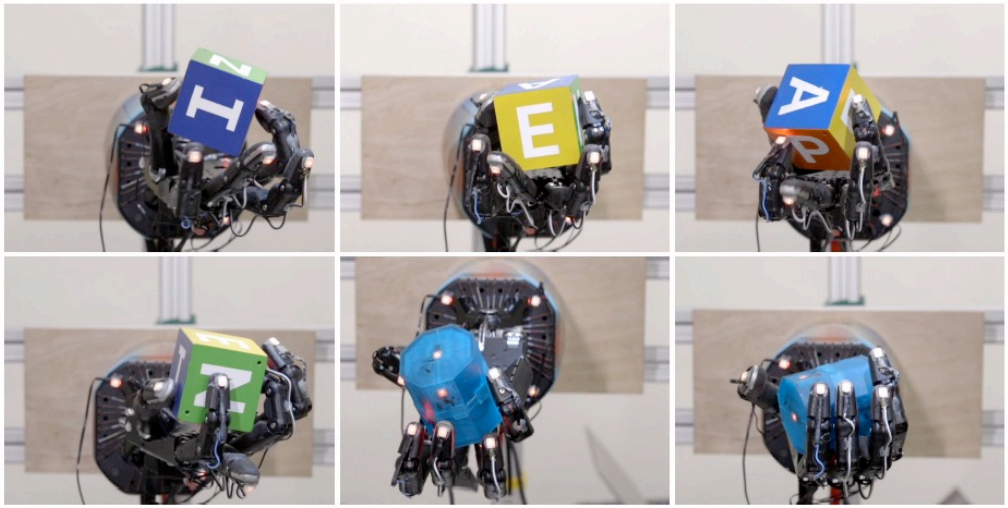
Results

- **Quantitative Results:**

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

Results

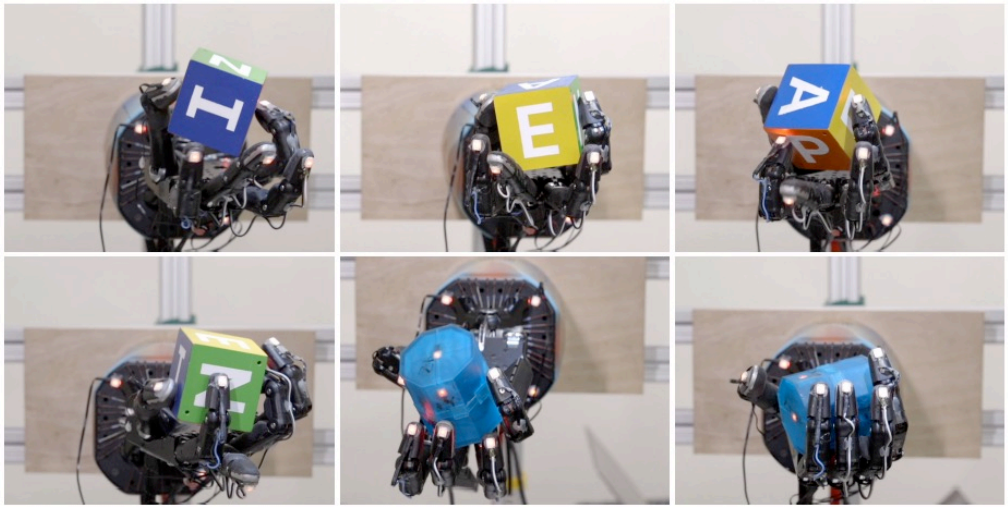
- **Qualitative Results**



- Many of the grasps found in humans (see Figure above). Furthermore, the policy also naturally discovers many strategies for dexterous in-hand manipulation described by the robotics community 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.

Results

- **Qualitative Results**



- For precision grasps, our policy tends to use the little finger instead of the index or middle finger. 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.

Other Experimentation

We also evaluate the performance on a second type of object, an octagonal prism.

To do so, we fine-tuned 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.

What would be the Performance comparison?

(SEE below for answer/discussion)

Surprising Observations

- Tactile sensing is not necessary to manipulate real-world objects.
- Randomizations developed for one object generalize to others with similar properties.
- With physical robots, good systems engineering is as important as good algorithms
- Using real data to train our vision policies didn't make a difference
- Decreasing reaction time did not improve performance.

CONCLUSION

- 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 which can be modelled in the simulator.

- Contemporary deep RL algorithms can be applied to solving complex real-world robotics problems which are beyond the reach of existing non-learning-based approaches.