CS686 Motion Planning Paper Presentation

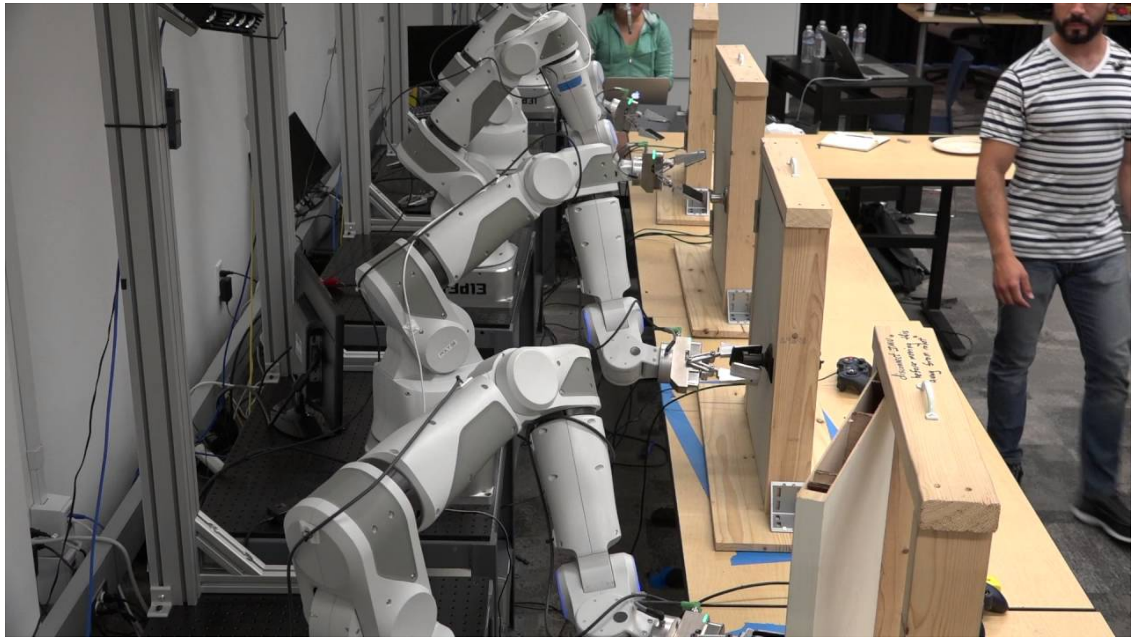
20193085 Min Kim

Paper 1: Deep Reinforcement Learning for Robotic Manipulation with Asynchronous Off-Policy Updates

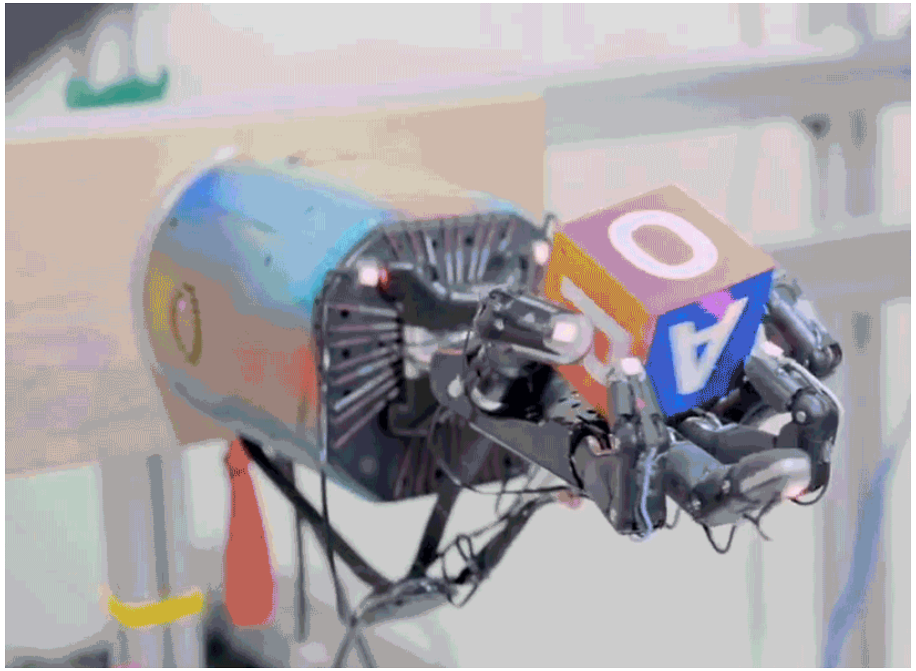
Paper 2: Learning dexterous in-hand manipulation

Paper Overview

Deep Reinforcement Learning for Robotic Manipulation with Asynchronous Off-Policy Updates

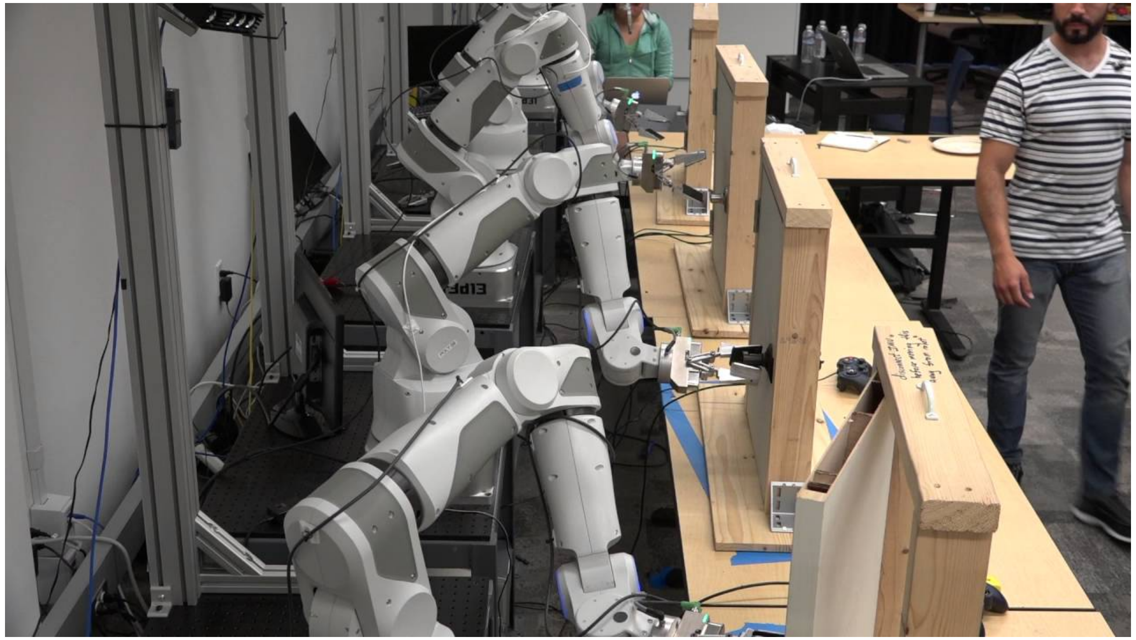


Learning dexterous in-hand manipulation

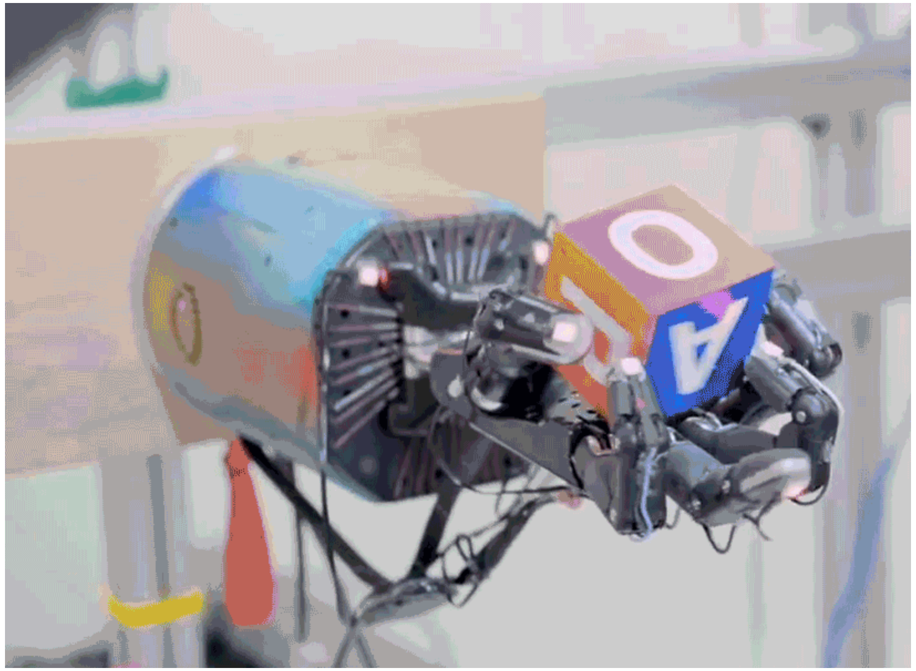


Paper Overview

Opening a door autonomously



Rolling a cube to required position



Contents

Basic Knowledge of Reinforcement Learning

Paper 1:

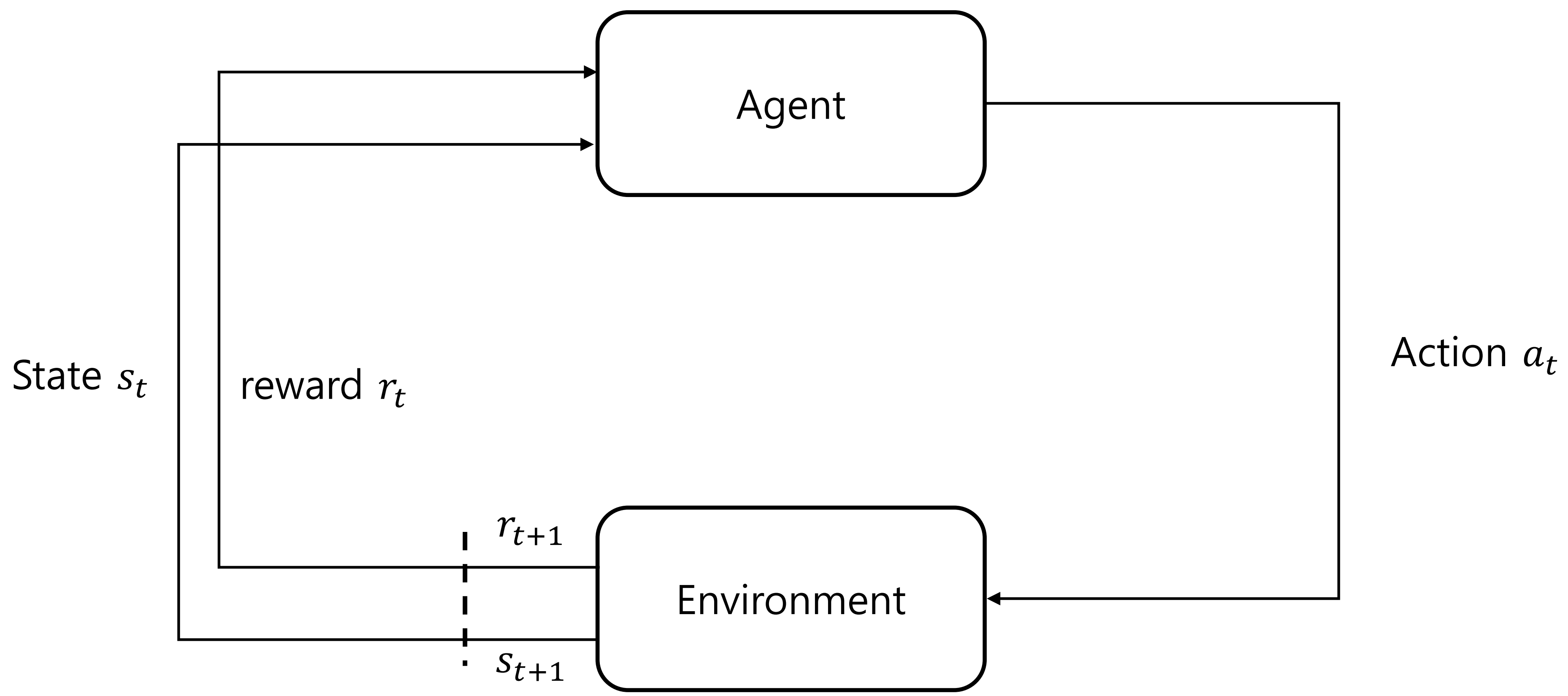
Deep Reinforcement Learning for Robotic Manipulation with Asynchronous Off-Policy Updates

Paper 2:

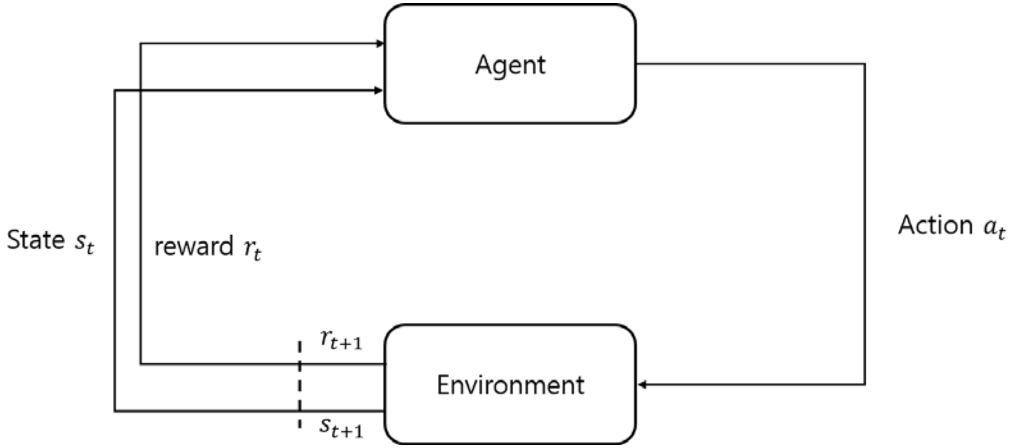
Rolling a cube to required position

Introduction to Reinforcement Learning

Reinforcement Learning



Reinforcement Learning: State-Value Function



Goal: Maximize our total reward

$$\textit{Total Reward} = r_{t+1} + r_{t+2} + r_{t+3} + \ldots$$

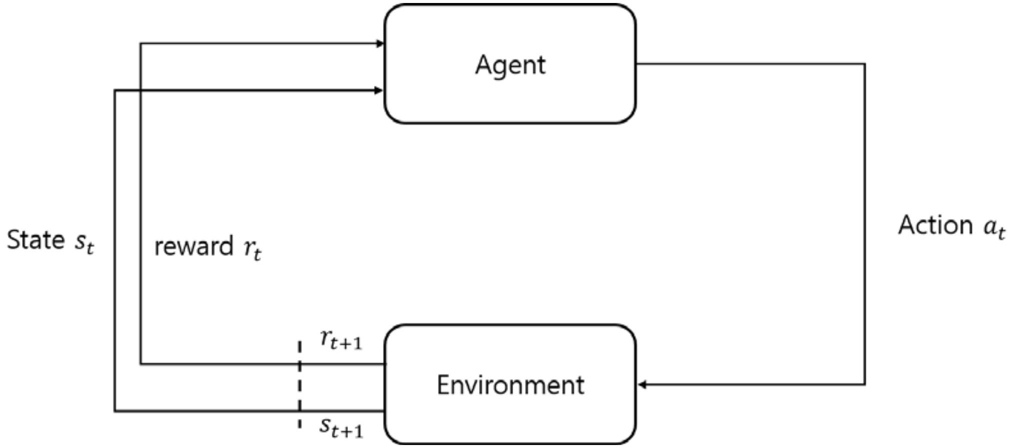
Reward received now would be more valuable than that of future => we apply discount γ < 1

$$\textit{Total Reward} = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots = \sum_{k-1}^{\infty} \gamma^k r_{t+k+1}$$

Rewards depend on current state and action

$$r_t := r_t(s_t, a_t)$$

Reinforcement Learning: State-Value Function



Goal: Find an optimal strategy that maximize total rewards

Policy: π The strategy that our agent will follow

$$a_t = \pi(s_t)$$

What is expectation of total rewards?

$$V^{\pi}(s) = E[\sum_{t=0}^{\infty} \gamma^t r\left(s_t, \pi(s_t)\right) | \, s_0 = s \, ; \, \pi]$$

$$\pi^* \in argmax_{\pi} V^{\pi}$$

Reinforcement Learning: Q-function

State-Value function only determines what is a “good state”, not evaluate “action”

$$V^{\pi}(s) = E[\sum_{t=0}^{\infty} \gamma^t r\big(s_t, \pi(s_t)\big) |\ s_0 = s\ ;\ \pi]$$

Q-function (state-action value function) – Off policy

$$Q^{\pi}(s, a) = E[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) |\ s_0 = s, a_0 = a, a_t = \pi(s_t)]$$

Reinforcement Learning: Bellman Operators

$$Q^{\pi}(s,a) = E[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) |\, s_0 = s, a_0 = a, a_t = \pi(s_t)]$$

Definition: For any W, the Bellman operator T^{π} is defined as

$$T^{\pi}W(x) = r\big(x, \pi(x)\big) + \gamma \sum\nolimits_y p(y|x, \pi(x))W(y)$$

(Monotonicity, Offset, Contraction, Fixed point)

Reinforcement Learning: Bellman Operators

Q-iteration

1. Let Q_0 be any Q-function
2. At each iteration k =1, 2, … ,K

   Compute $Q_{k+1} = TQ_k$
3. Return the greedy policy

   $\pi_K(x) = argmax_{a \in A} Q(s, a)$

Asynchronous VI

1. Let Q_0 be any Q-function
2. At each iteration k =1, 2, … ,K

   Choose a state s_k, a_k

   Compute $Q_{k+1}(s_k, a_k) = TQ_k(s_k, a_k)$
3. Return the greedy policy

   $\pi_K(x) = argmax_{a \in A} Q(s, a)$

Policy Gradient

We assume that policy π is differentiable with respect to some parameter θ that $\frac{d\pi(s,a)}{d\theta}$ exists

Then for total reward ρ:

$$\frac{d\rho}{d\theta} = \sum_s d^{\pi}(s) \sum_a \frac{d\pi(s,a)}{d\theta} Q^{\pi}(s,a)$$

$$= \mathrm{E}[\frac{d}{d\theta} log\pi(s,a) Q^{\pi}(s,a)]$$

But, taking expectation over infinite number of cases is impractical => sampling

$$\mathrm{E}[\frac{d}{d\theta} log\pi(s,a) Q^{\pi}(s,a)] \approx \frac{1}{K+1} \sum_{t=0}^{k} \frac{d}{d\theta} log\pi(s_t,a_t) Q^{\pi}(s_t,a_t)$$

To estimate Q with low bias => we need large K => large variance

Policy Gradient

Actor-critic algorithms:
Use critic to estimate the action-value function $Q^{\pi}(s,a) \approx Q^{\pi}(s,a,w)$

Critic: Update action-value function parameters w

Actor update policy parameters θ, in direction suggested by critic

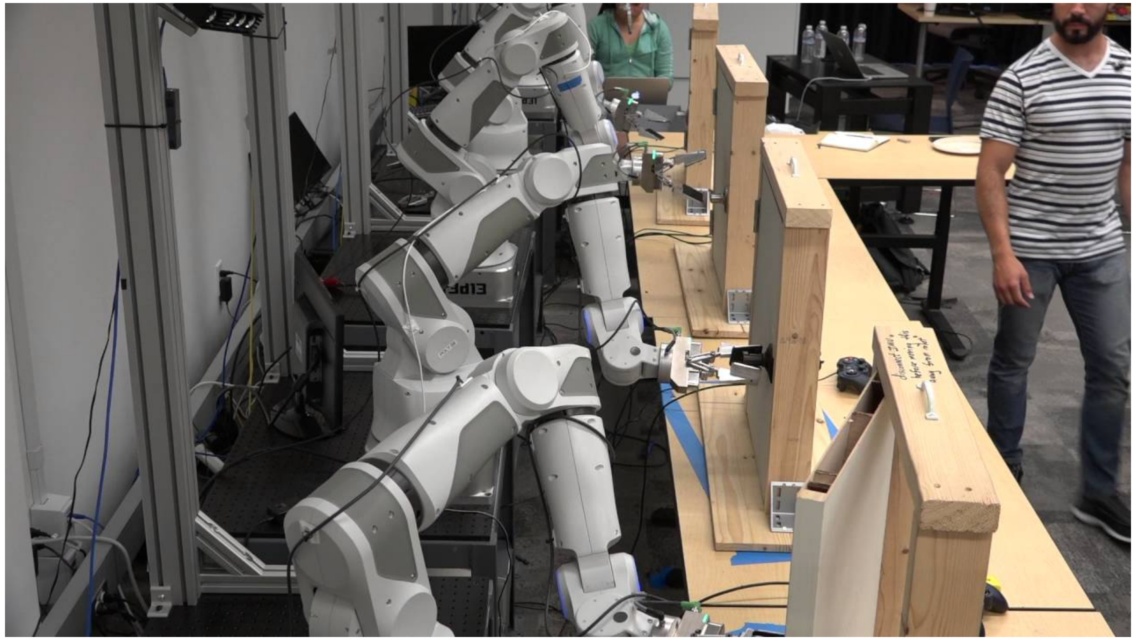
Deep Reinforcement Learning for Robotic Manipulation with Asynchronous Off-Policy Updates

Paper 1: Discrete to Continuous

Robot opens door in a continuous space, but original methods are designed for Discrete space

Require large amount of training time with multiple trial and error.

Discrete



Continuous

NAF (Normalized Advantage Function)

Represent Q-function by value function V and advantage term A produced by neural network

$$Q(s, a|\theta^Q) = A(s, a|\theta^A) + V(s|\theta^V)$$

Training Process:

1. Initialize state s_0,
2. Iteratively select action $a_t = \pi(s_t|\theta^\pi)$
3. Generate transition (s_t, a_t, r_t, s_{t+1}) and store in the buffer — Collect
4. sample random minibatch from buffer and do below for multiple times — Train

   set target $y_i = r_i + \gamma V'(s_{i+1}|\theta^{Q'})$

   update θ^Q by minimize loss L = $\frac{1}{N}\sum_i(y_i - Q(s_i, a_i|\theta^Q))^2$

   update the target network: $\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$

Gu, Shixiang, et al. "Continuous deep q-learning with model-based acceleration." International Conference on Machine Learning. 2016.

Paper Title

Deep Reinforcement Learning for Robotic Manipulation
with Asynchronous Off-Policy Updates

Asynchronous

Robotic Manipulation

Deep Reinforcement Learning

Off-Policy

Asynchronous NAF

Algorithm 1 Asynchronous NAF - N collector threads and 1 trainer thread

// trainer thread
Randomly initialize normalized Q network $Q(\boldsymbol{x}, \boldsymbol{u}|\theta^Q)$, where $\theta^Q = \{\theta^\mu, \theta^P, \theta^V\}$ as in Eq. 1
Initialize target network Q' with weight $\theta^{Q'} \leftarrow \theta^Q$
Initialize shared replay buffer $R \leftarrow \emptyset$
for iteration=1, I **do**
  Sample a random minibatch of m transitions from R
  Set $y_i = \begin{cases} r_i + \gamma V'(\boldsymbol{x}'_i|\theta^{Q'}) & \text{if} \quad t_i < T \\ r_i & \text{if} \quad t_i = T \end{cases}$
  Update the weight θ^Q by minimizing the loss: $L = \frac{1}{m}\sum_i (y_i - Q(\boldsymbol{x}_i, \boldsymbol{u}_i|\theta^Q))^2$
  Update the target network: $\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$
end for
// collector thread n, $n = 1...N$
Randomly initialize policy network $\boldsymbol{\mu}(\boldsymbol{x}|\theta_n^\mu)$
for episode=1, M **do**
  Sync policy network weight $\theta_n^\mu \leftarrow \theta^\mu$
  Initialize a random process $\mathcal{N}$ for action exploration
  Receive initial observation state $\boldsymbol{x}_1 \sim p(\boldsymbol{x}_1)$
  for t=1, T **do**
    Select action $\boldsymbol{u}_t = \boldsymbol{\mu}(\boldsymbol{x}_t|\theta_n^\mu) + \mathcal{N}_t$
    Execute $\boldsymbol{u}_t$ and observe r_t and $\boldsymbol{x}_{t+1}$
    Send transition $(\boldsymbol{x}_t, \boldsymbol{u}_t, r_t, \boldsymbol{x}_{t+1}, t)$ to R
  end for
end for

Asynchronous NAF

Trainer

Randomly initialize normalized Q network $Q(\boldsymbol{x}, \boldsymbol{u}|\theta^Q)$, where $\theta^Q = \{\theta^\mu, \theta^P, \theta^V\}$ as in Eq. 1
Initialize target network Q' with weight $\theta^{Q'} \leftarrow \theta^Q$
Initialize shared replay buffer $R \leftarrow \emptyset$
for iteration=1, I **do**
  Sample a random minibatch of m transitions from R
  Set $y_i = \begin{cases} r_i + \gamma V'(\boldsymbol{x}'_i|\theta^{Q'}) & \text{if } t_i < T \\ r_i & \text{if } t_i = T \end{cases}$
  Update the weight θ^Q by minimizing the loss: $L = \frac{1}{m}\sum_i (y_i - Q(\boldsymbol{x}_i, \boldsymbol{u}_i|\theta^Q))^2$
  Update the target network: $\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$
end for

Off-policy, continuous

$$L = \frac{1}{N}\sum_i (y_i - Q(x_i, u_i|\theta^Q))^2$$

R

Collector

$(x_t, u_t, r_r, x_{t+1}, t)$ $(x_t, u_t, r_r, x_{t+1}, t)$ $(x_t, u_t, r_r, x_{t+1}, t)$ $(x_t, u_t, r_r, x_{t+1}, t)$

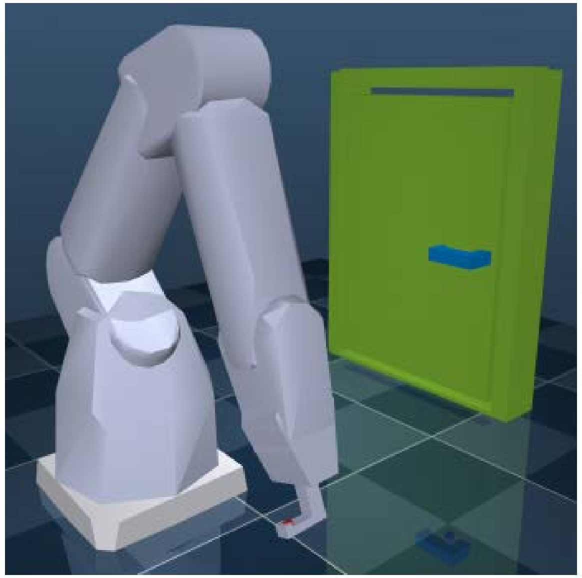
Randomly initialize policy network $\boldsymbol{\mu}(\boldsymbol{x}|\theta_n^\mu)$
for episode=1, M **do**
  Sync policy network weight $\theta_n^\mu \leftarrow \theta^\mu$
  Initialize a random process $\mathcal{N}$ for action exploration
  Receive initial observation state $\boldsymbol{x}_1 \sim p(\boldsymbol{x}_1)$
  for t=1, T **do**
    Select action $\boldsymbol{u}_t = \boldsymbol{\mu}(\boldsymbol{x}_t|\theta_n^\mu) + \mathcal{N}_t$
    Execute $\boldsymbol{u}_t$ and observe r_t and $\boldsymbol{x}_{t+1}$
    Send transition $(\boldsymbol{x}_t, \boldsymbol{u}_t, r_t, \boldsymbol{x}_{t+1}, t)$ to R
  end for
end for

Tasks

1. Simulation Tasks:

For showing effectiveness of Asynchronous NAF with multiple collecting threads

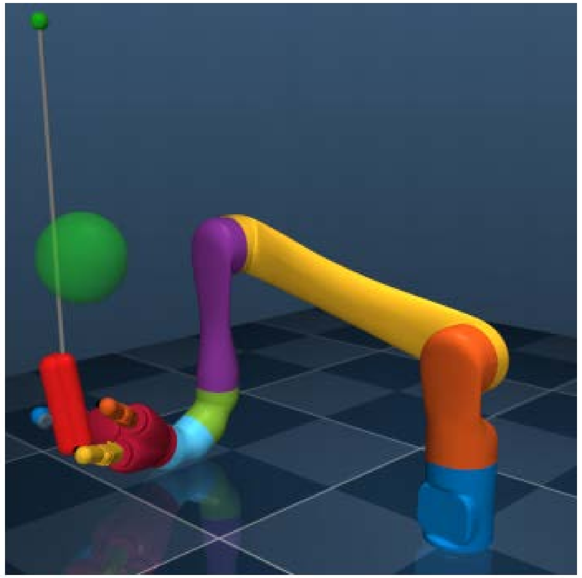
Door Pushing and Pulling



IMU sensor => door angle, positions

$$r(\boldsymbol{x},\boldsymbol{u}) = -c_1 d(\boldsymbol{h},\boldsymbol{e}(\boldsymbol{x})) + c_2(-d(\boldsymbol{q}_o,\boldsymbol{q}(\boldsymbol{x})) + d_i) - c_3 \boldsymbol{u}^T \boldsymbol{u}$$

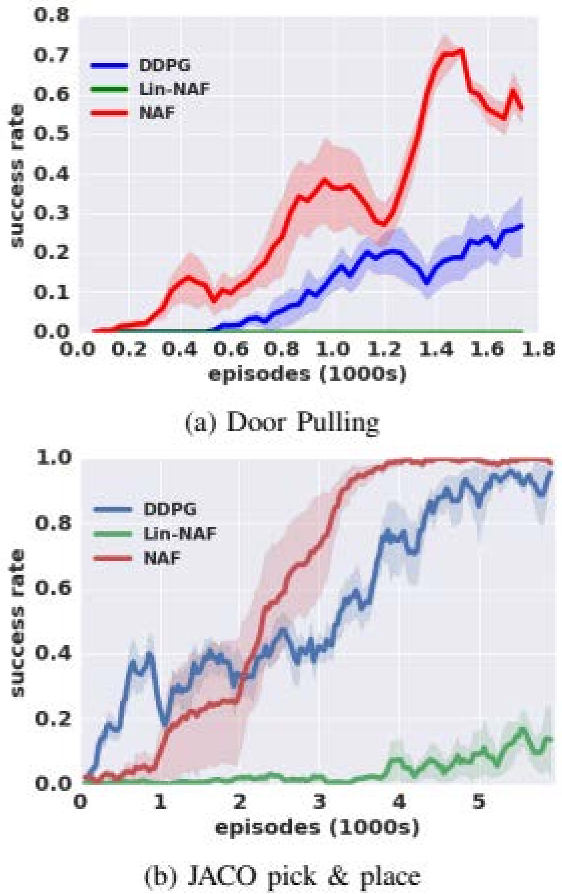
Reaching, Pick and Place



$$r(\boldsymbol{x},\boldsymbol{u}) = -c_1 d(\boldsymbol{y},\boldsymbol{e}(\boldsymbol{x})) - c_2 \boldsymbol{u}^T \boldsymbol{u}$$

$$r(\boldsymbol{x},\boldsymbol{u}) = -c_1 d(\boldsymbol{s}(\boldsymbol{x}),\boldsymbol{g}(\boldsymbol{x})) - c_2 \sum_{i=1}^{3} d(\boldsymbol{s}(\boldsymbol{x}),\boldsymbol{f}_i(\boldsymbol{x})) - c_3 d(\boldsymbol{y},\boldsymbol{s}(\boldsymbol{x})) - c_4 \boldsymbol{u}^T \boldsymbol{u}$$

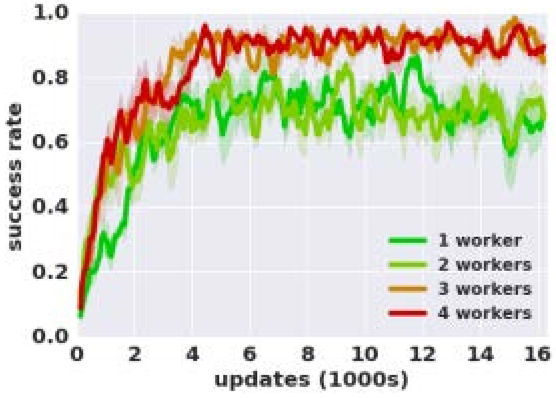
Results: Comparing different models



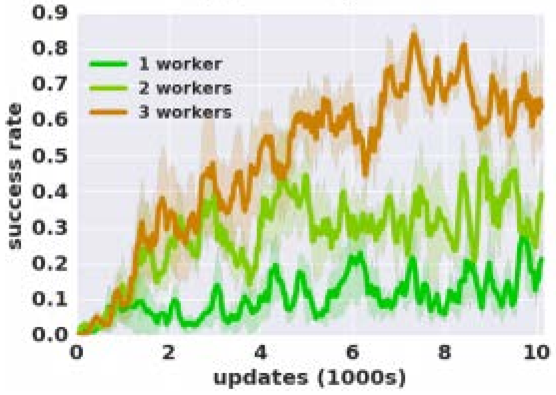
Even when only one robot arms are used

NAF performs much better than other types of networks

Results: Comparing Number of Workers



(a) Reaching



(b) Door Pushing

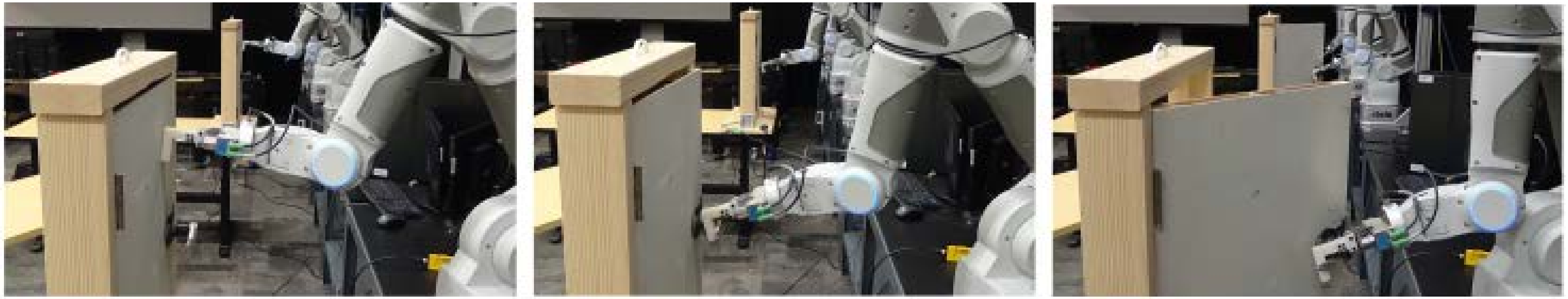
Using Asynchronous Training:

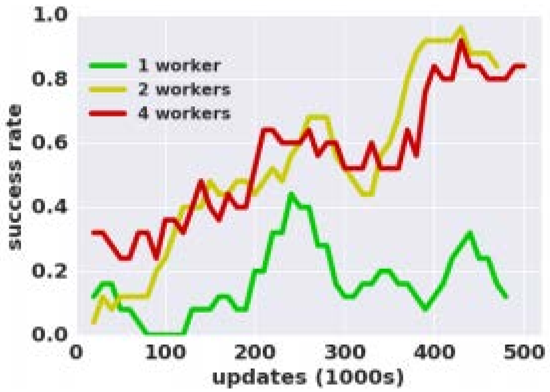
As the number of worker increase, success rate increase

As the task gets more complicated,
more workers => high success rate

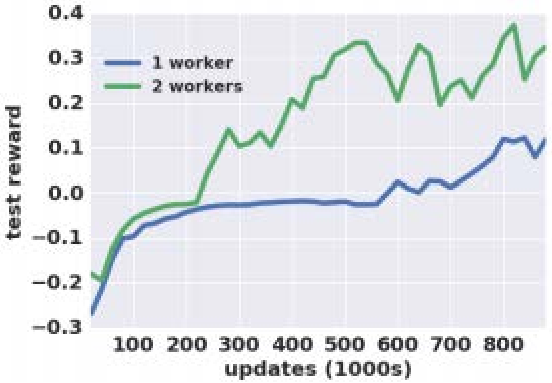
Results: Real World Experiments

1. Real world opening door





Target Reaching Task



Door Opening Task

Target Reaching:
2, 4 workers significantly improves learning speed over 1 worker

Door Opening:
2 workers:
needed 2.5 hours to learn to 100% success rate across 20 consecutive trials
1 workers:
needed more than 4 hours

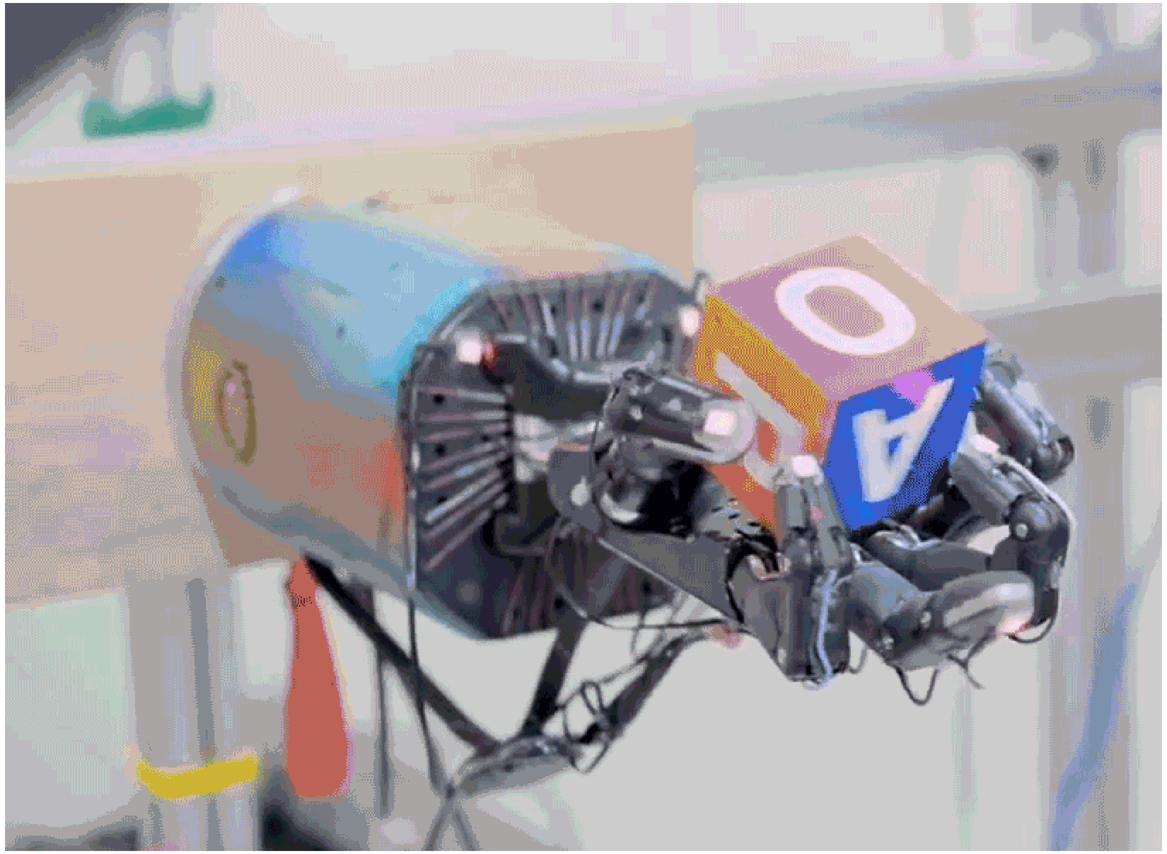
Ways to Improve

1. They specified reward function to guide learning algorithm, limiting exploration and learning speed.

2. Incapable of dealing with multiple situations such as door with different types.

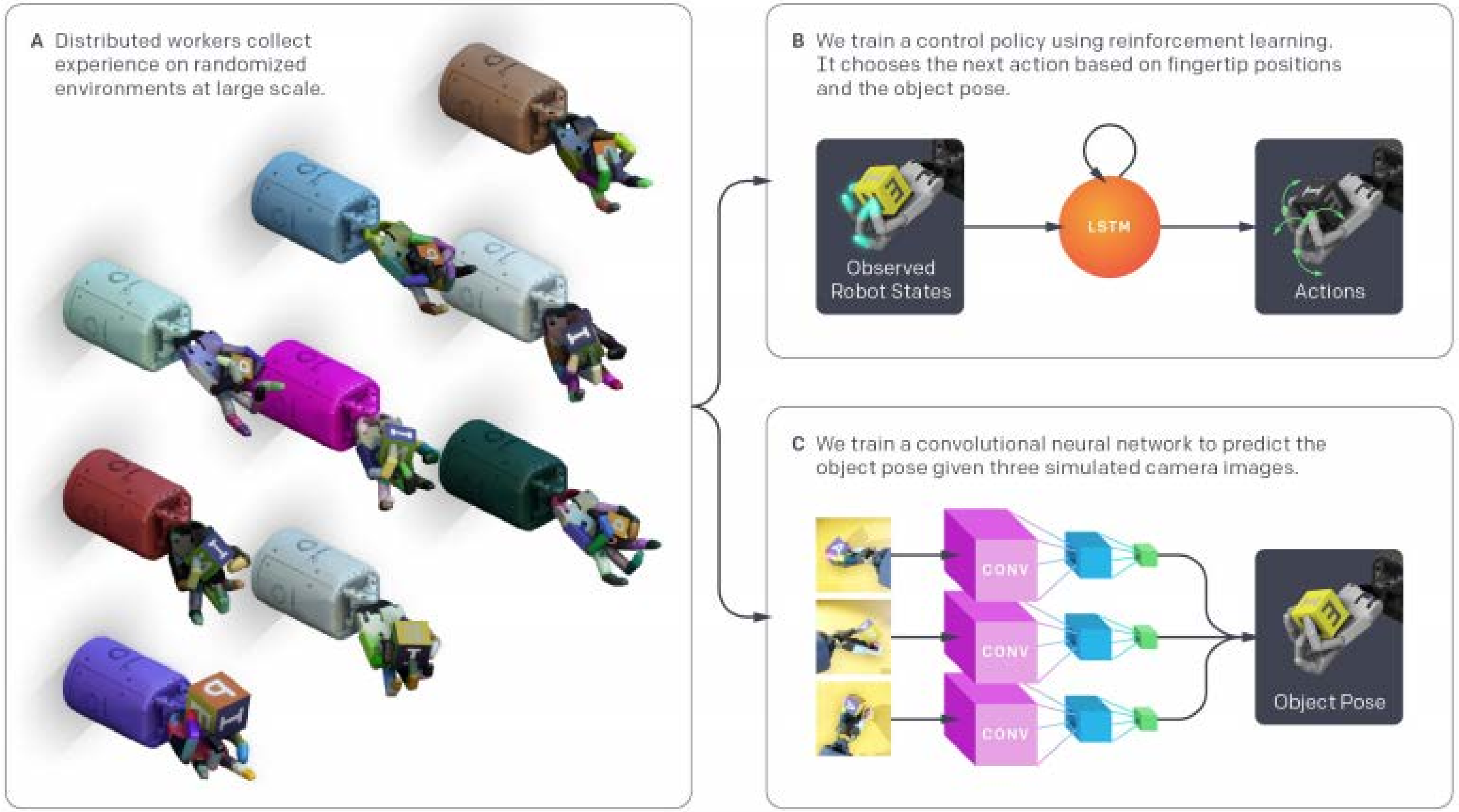
3. Require multiple expensive robot arms to train network to work in a real world

Learning Dexterous In-Hand Manipulation

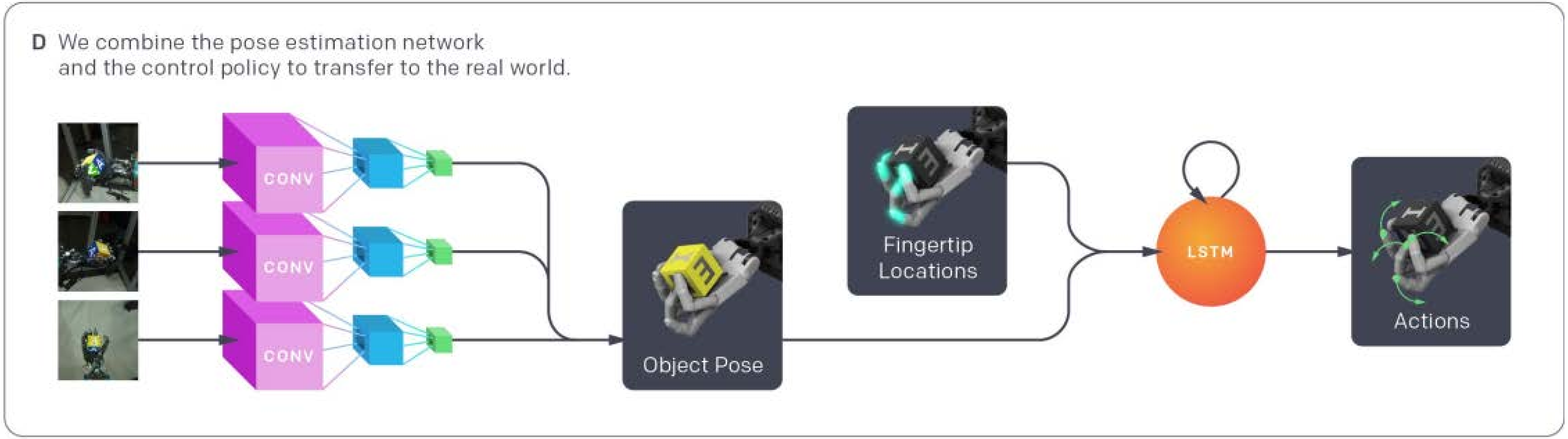


Experiment Environment

System Overview



System Overview



Transferable Simulations

Randomization:

1. Observation Noise:

   Gaussian noise to policy observation

2. Physics randomization:

   Physical parameters like friction are randomized for each episode

3. Unmodeled effects

   To demonstrate unexpected effects in real-world, added random motor backlash and action delays.

4. Visual appearance randomization

   Randomize camera positions, lighting conditions, hand and object poses

Policy Architecture

Policy model: LSTM

Required memory augmented policy to identify properties of the current environment and adapt its behavior accordingly

Training: Proximal Policy Optimization (PPO)

on-policy RL algorithm that uses 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.

Encourages the policy to take actions which are better than average while discouraging bigger changes to the policy

Rewards:

$r_t = d_t - d_{t-1}$, where d is the rotation angles between the desired and current orientations

+5 when a goal is achieve, -20 whenever the object is dropped

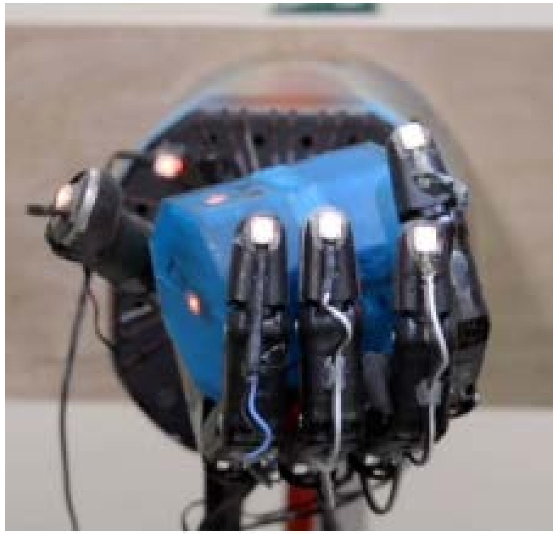
Results

Qualitative Results:

Without Human demonstration, many different grasp types are learned by the policy:

Tip Pinch Grasp

Tripod Grasp



Power Grasp

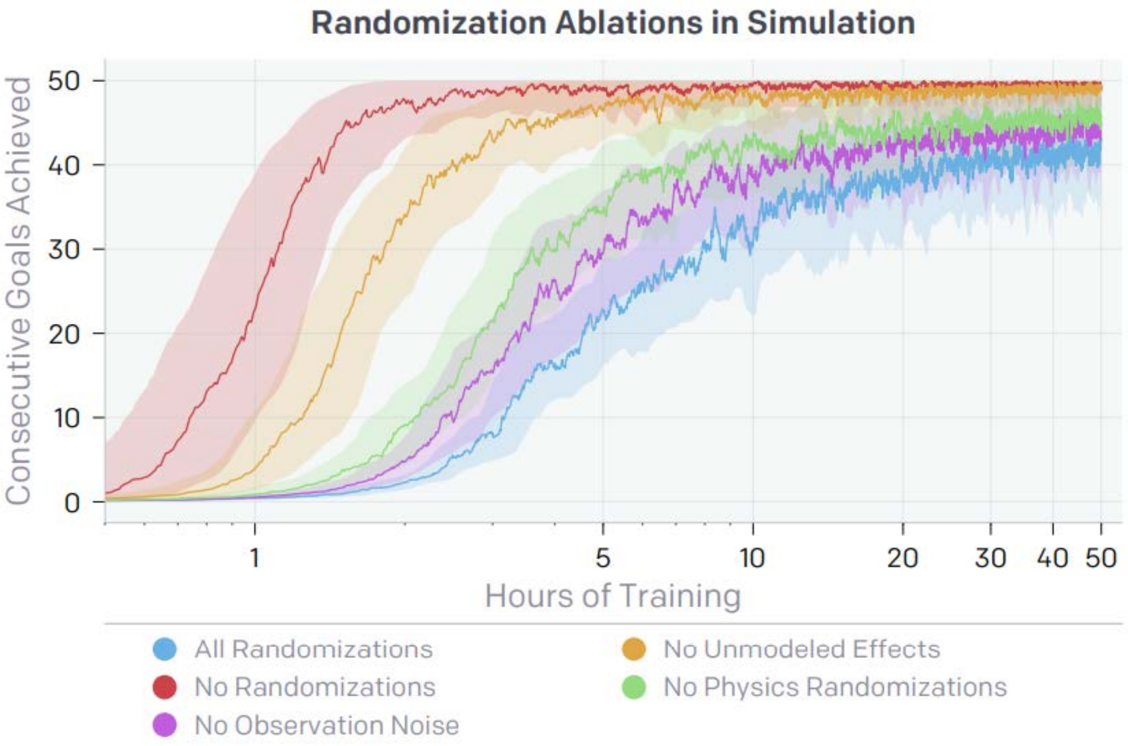
Results

Quantitative Results:

Number of successful consecutive rotations in simulation and physical world

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

Effect of Randomization



Number of successful consecutive rotations with various randomization in physical environment

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

Limitations

1. Even with randomization: there are still gaps between performance in simulation and physical worlds

2. Cannot accomplish more dexterous motions such as rotating a pan around fingers only with vision data

Thank you