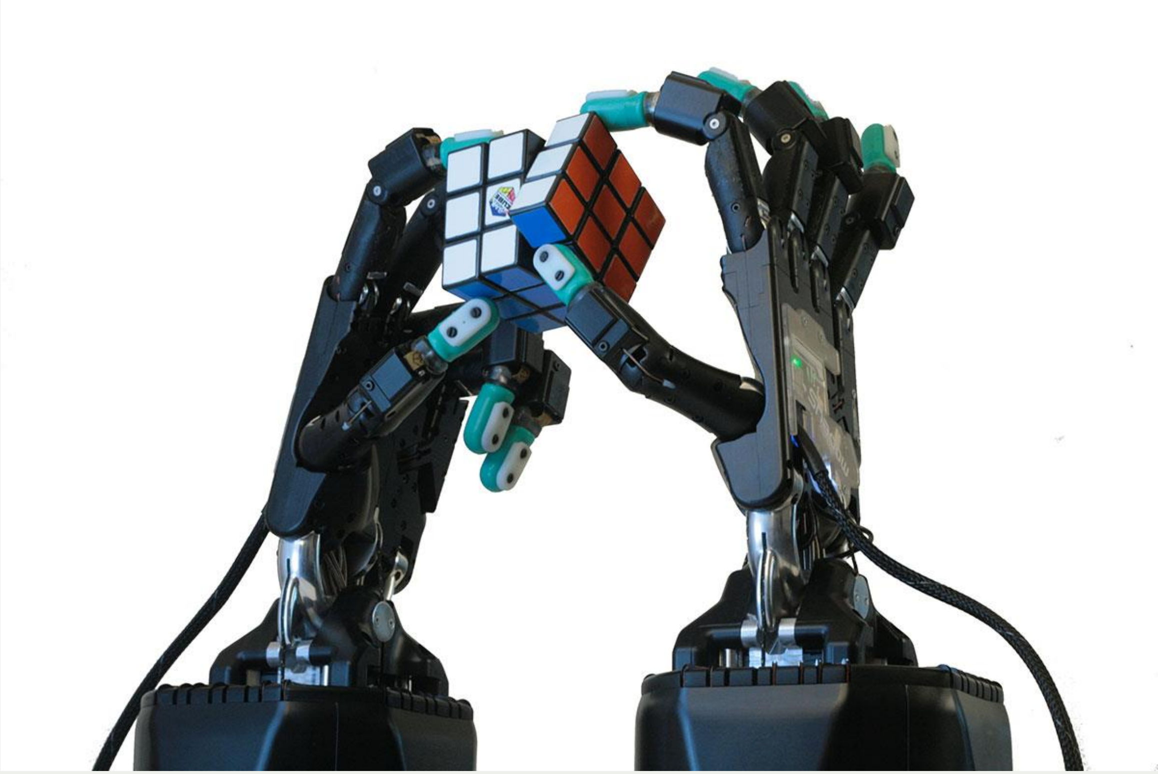
Learning dexterous in-hand manipulation

Open AI et al. 2019

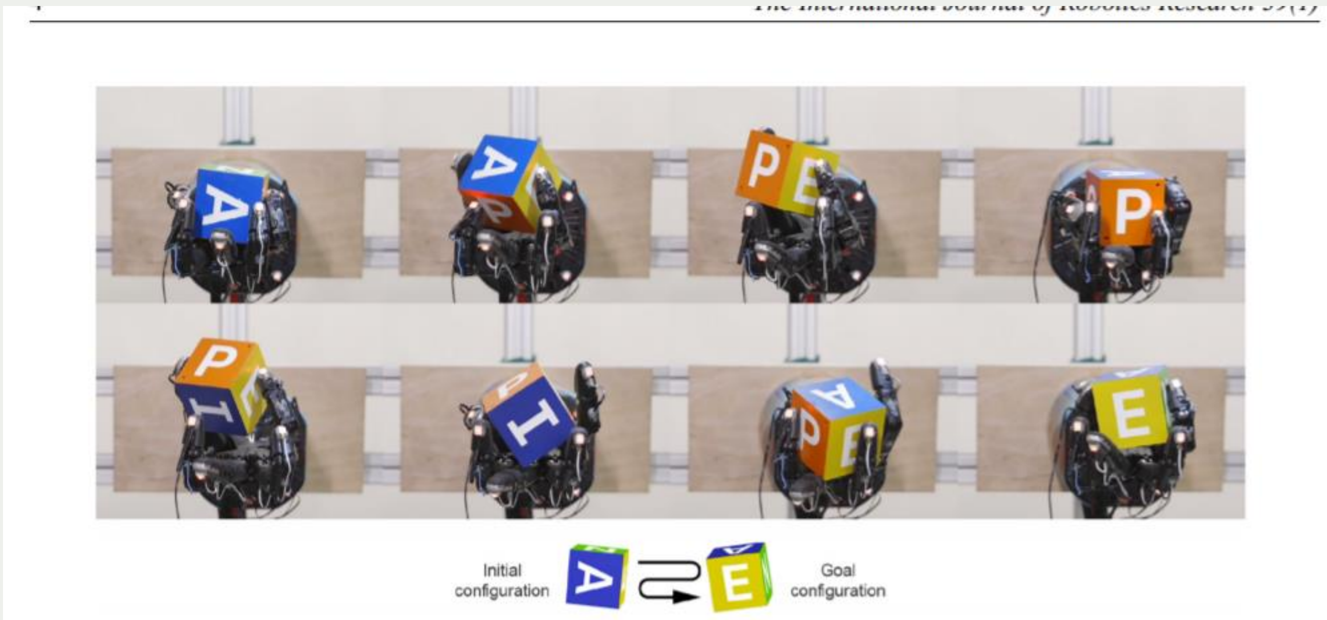
Ingyun Ahn

Summary



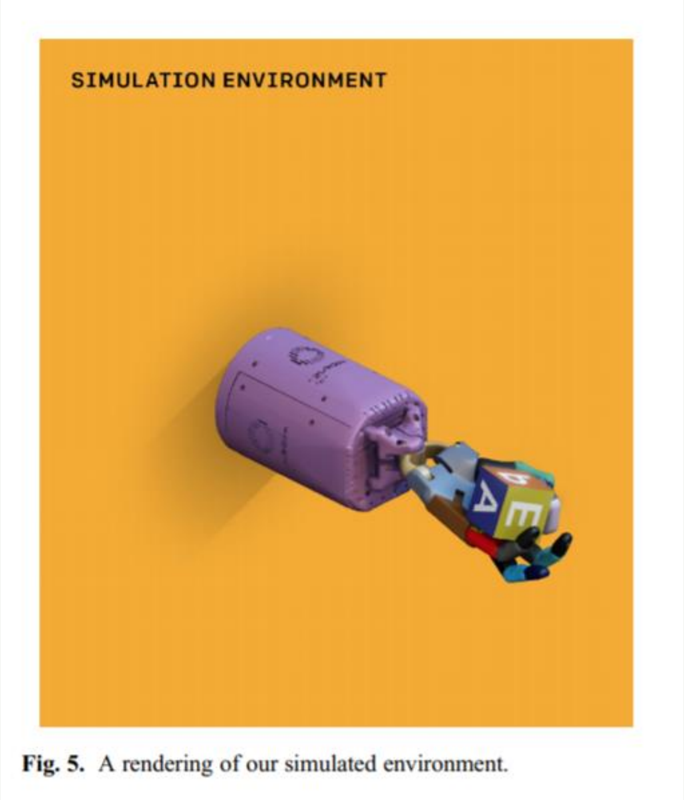
The Shadow Dexterous Hand

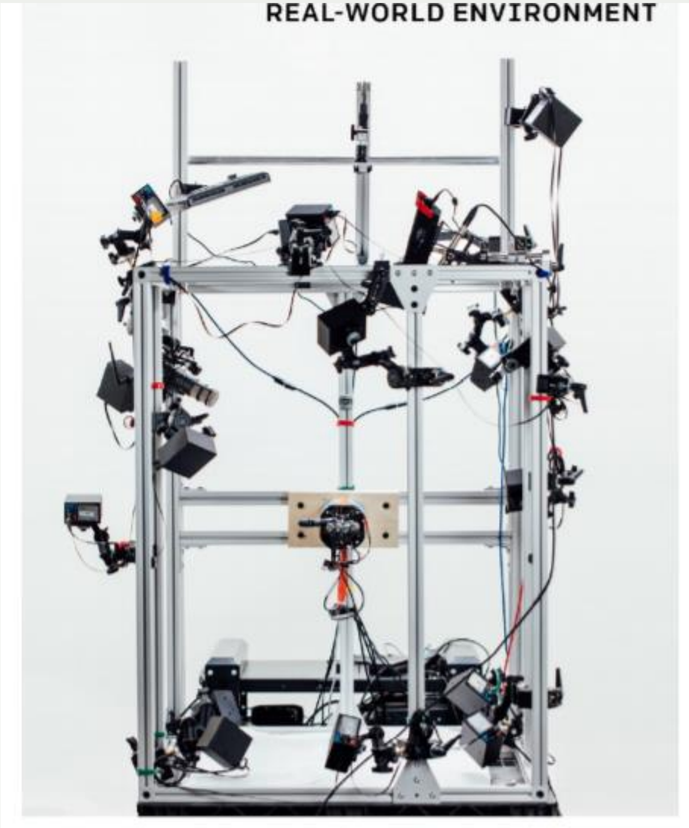
Summary



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

Summary





Simulation

Real World

Background-GAE

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

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

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

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

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

Background-PPO

Algorithm 1 PPO-Clip

1: 입력 : 초기 파라미터 θ_0, 초기 value function 파라미터 ϕ_0

2: **for** k = 0, 1, 2, ... **do**

3: 정책 $\pi_k = \pi(\theta_k)$으로 trajectory $D_k = \{\tau_i\}$를 모읍니다.

4: rewards-to-go $\hat{R}_t$를 계산합니다.

5: 현재 value function V_{ϕ_k}으로 advantage $\hat{A}_t$를 계산합니다.
PPO-Clip을 최대화하여 정책을 업데이트합니다.

$$maximize_{\theta} L^{CLIP}(\theta) = \hat{E}_t[min(r_t(\theta)\hat{A}_t), clip(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t]$$

6: $$\theta_{k+1} = \arg\max_{\theta} \frac{1}{|\mathcal{D}_k|T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \min\left(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_k}(a_t|s_t)} A^{\pi_{\theta_k}}(s_t, a_t), \;\; g(\epsilon, A^{\pi_{\theta_k}}(s_t, a_t)) \right)$$

보통 Adam같은 stochastic gradient ascent를 사용합니다.
mean-squared error를 통해 regression해서 value function을 학습합니다.

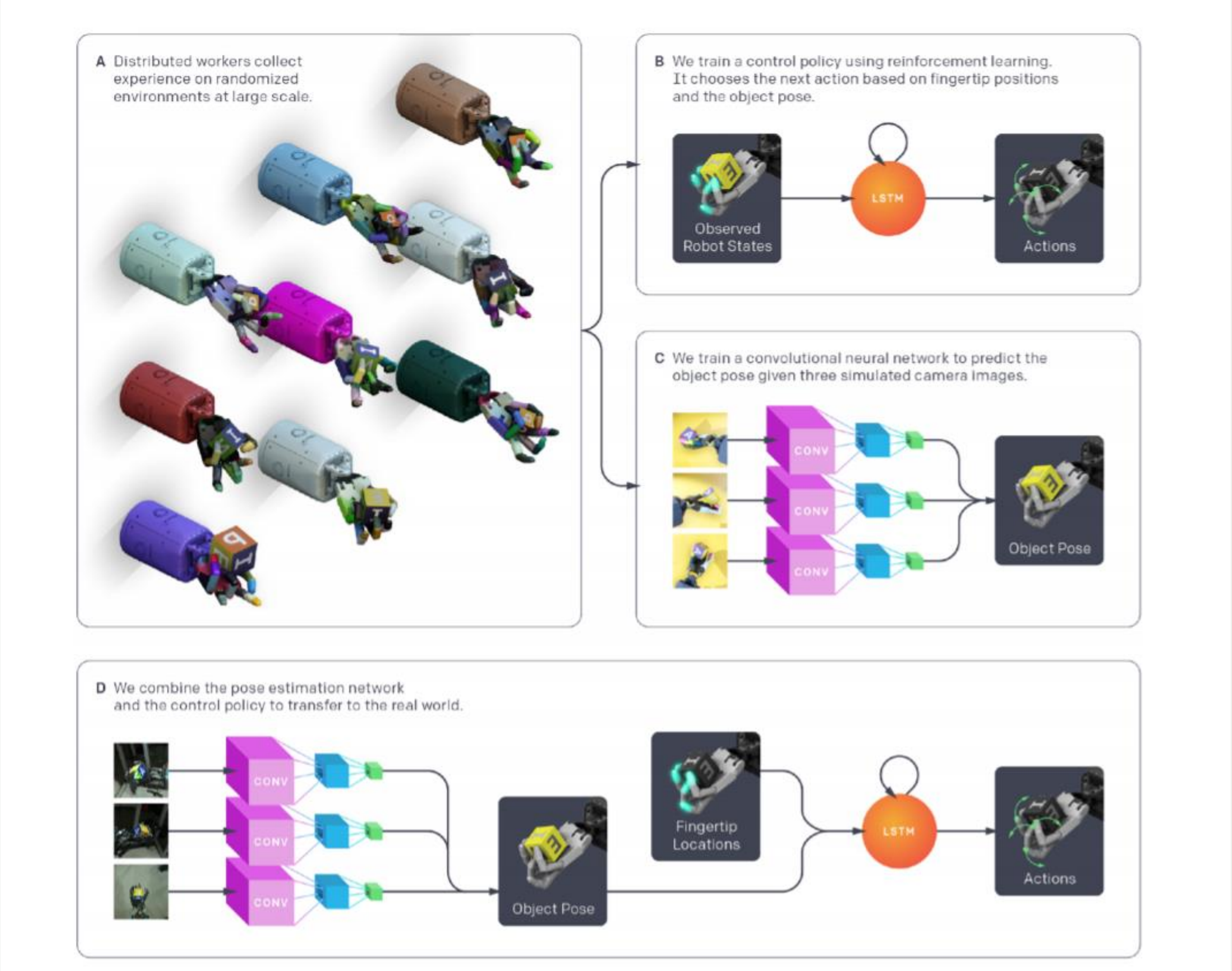
7: $$\phi_{k+1} = \arg\min_{\phi} \frac{1}{|\mathcal{D}_k|T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \left(V_\phi(s_t) - \hat{R}_t \right)^2$$

보통은 gradient descent 알고리즘을 사용합니다.

8: **end for**

https://www.youtube.com/watch?v=CKaN5PgkSBc&t=90s

Task&System


A Distributed workers collect experience on randomized environments at large scale.
B We train a control policy using reinforcement learning. It chooses the next action based on fingertip positions and the object pose.
Observed Robot States
LSTM
Actions
C We train a convolutional neural network to predict the object pose given three simulated camera images.
CONV
CONV
CONV
Object Pose
D We combine the pose estimation network and the control policy to transfer to the real world.
CONV
CONV
CONV
Object Pose
Fingertip Locations
LSTM
Actions

Task&System

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).

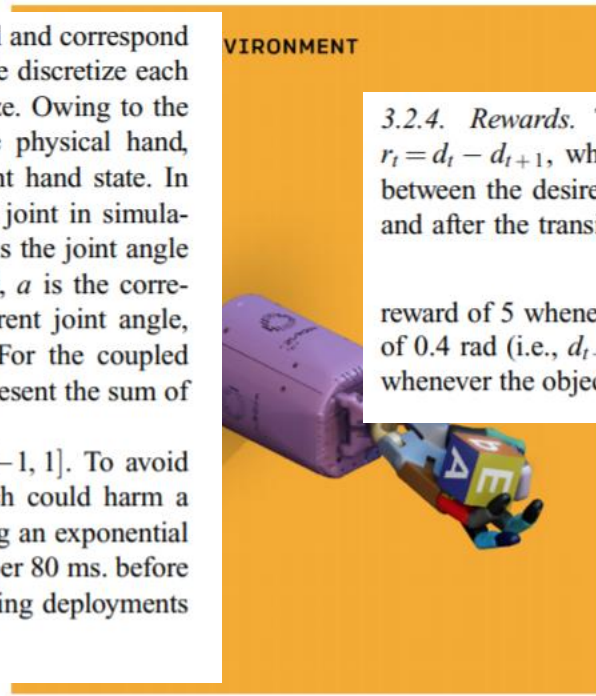


Fig. 5. A rendering of our simulated environment.

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.

Result

Updating..