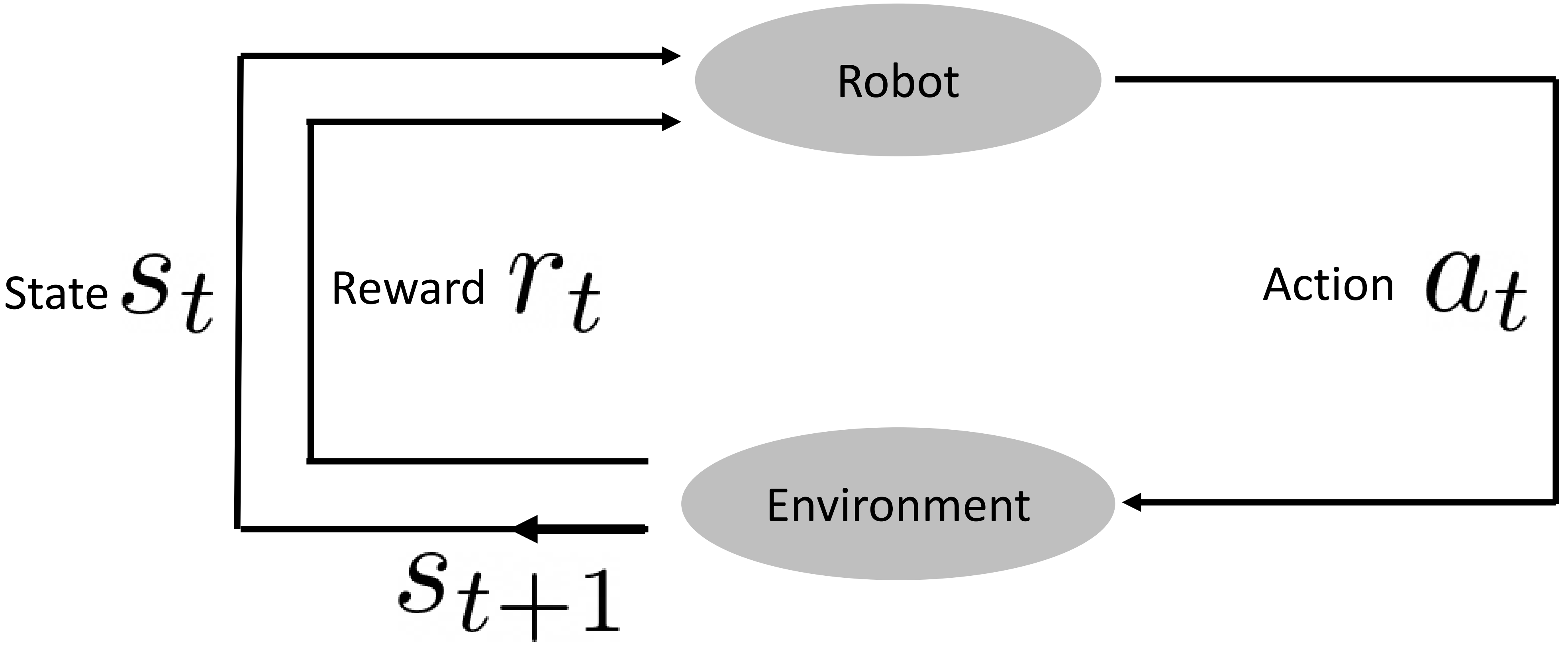
Reinforcement Learning: Actor-Critic

CS 6341 Robotics

Professor Yu Xiang

The University of Texas at Dallas

Reinforcement Learning



Reinforcement Learning:
Imitation Learning:

$$a_t = \pi(s_t)$$

Last Lecture: Policy Optimization

- Maximize expected return

$$J(\pi_\theta) = \mathop{\mathrm{E}}_{\tau\sim\pi_\theta}[R(\tau)]$$

$$R(\tau) = \sum_{t=0}^{T} r_t$$

$$J(\pi_\theta) = \int_\tau P(\tau|\theta)R(\tau)$$

$$P(\tau|\theta) = \rho_0(s_0)\prod_{t=0}^{T} P(s_{t+1}|s_t, a_t)\pi_\theta(a_t|s_t)$$

- Policy gradient

$$\nabla_\theta J(\pi_\theta) = \mathop{\mathrm{E}}_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{T}\nabla_\theta \log \pi_\theta(a_t|s_t)R(\tau)\right]$$

$$\theta_{k+1} = \theta_k + \alpha\, \nabla_\theta J(\pi_\theta)|_{\theta_k}$$

$$\nabla_\theta J(\pi_\theta) = \mathop{\mathrm{E}}_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{T}\nabla_\theta \log \pi_\theta(a_t|s_t)\sum_{t'=t}^{T} R(s_{t'}, a_{t'}, s_{t'+1})\right]$$

reward-to-go

$$\nabla_\theta J(\pi_\theta) = \mathop{\mathrm{E}}_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{T}\nabla_\theta \log \pi_\theta(a_t|s_t)A^{\pi_\theta}(s_t, a_t)\right]$$

Advantage

$$A^\pi(s,a) = Q^\pi(s,a) - V^\pi(s)$$

Q-Learning

- Learn the optimal Q function $$Q^*(s,a) = \max_{\pi} \mathop{\mathrm{E}}_{\tau \sim \pi} [R(\tau) \,|s_0 = s, a_0 = a]$$
- Policy from the Q function $$a^*(s) = \arg\max_a Q^*(s,a)$$
- How to learn the Q function?
  - Bellman Equation $$Q^*(s,a) = \mathop{\mathrm{E}}_{s' \sim P} \left[r(s,a) + \gamma \max_{a'} Q^*(s',a') \right]$$

Q-Learning

- For discrete states and actions

- Dynamic programming (Q-table)
  - Initialize Q values arbitrarily $Q_0(s,a) = 0$
  - Then iterate

$$Q_{k+1}(s,a) = r(s,a) + \gamma \sum_{s'} P(s'|s,a) \max_{a'} Q_k(s',a')$$

$$a^*(s) = \arg\max_a Q^*(s,a)$$

0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0 R: 1	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0

https://mohitmayank.com/blog/interactive-q-learning

Q-Learning

$$Q^*(s,a) = \mathop{\mathrm{E}}_{s' \sim P}\left[r(s,a) + \gamma \max_{a'} Q^*(s',a')\right]$$

- What if the state and action space is large?
  - We cannot store a table
- Use parameterization $Q_\phi(s,a)$
- Collect a set of transitions $\mathcal{D} = \{(s_i, a_i, r_i, s_i')\}$ Replay Buffer
- TD target $y_i = r_i + \gamma \max_{a'} Q_{\phi^-}(s_i', a')$
- Loss function $L(\phi) = \frac{1}{N}\sum_i (Q_\phi(s_i, a_i) - y_i)^2$ $\phi \leftarrow \phi - \alpha \nabla_\phi L(\phi)$
- Update the target network $\phi^- \leftarrow \phi$

Q-Learning

- TD target $y_i = r_i + \gamma \max_{a'} Q_{\phi^-}(s'_i, a')$

- How to compute this max?
  - Discretize actions $Q_{\phi^-}(s') = \left[Q_{\phi^-}(s', a_1),\ Q_{\phi^-}(s', a_2),\ \ldots,\ Q_{\phi^-}(s', a_K) \right]$

Playing Atari with Deep Reinforcement Learning

Volodymyr Mnih **Koray Kavukcuoglu** **David Silver** **Alex Graves** **Ioannis Antonoglou**
Daan Wierstra **Martin Riedmiller**
DeepMind Technologies

We now describe the exact architecture used for all seven Atari games. The input to the neural network consists is an $84 \times 84 \times 4$ image produced by ϕ. The first hidden layer convolves 16 8×8 filters with stride 4 with the input image and applies a rectifier nonlinearity [10, 18]. The second hidden layer convolves 32 4×4 filters with stride 2, again followed by a rectifier nonlinearity. The final hidden layer is fully-connected and consists of 256 rectifier units. The output layer is a fully-connected linear layer with a single output for each valid action. The number of valid actions varied between 4 and 18 on the games we considered. We refer to convolutional networks trained with our approach as Deep Q-Networks (DQN).

Volodymyr Mnih et al., 2013 (arXiv preprint)

Q-Learning

- TD target $y_i = r_i + \gamma \max_{a'} Q_{\phi^-}(s'_i, a')$

- How to compute this max?
  - Discretize actions $Q_{\phi^-}(s') = \left[Q_{\phi^-}(s', a_1),\ Q_{\phi^-}(s', a_2),\ \ldots,\ Q_{\phi^-}(s', a_K) \right]$

  - Continuous actions: actor-critic methods
    - Learn a policy (actor) $\pi_\theta(s') \approx \arg\max_a Q_\phi(s', a)$

$$y = r + \gamma Q_{\phi^-}(s', \pi_\theta(s'))$$

Deep Deterministic Policy Gradient (DDPG)

- DDPG currently learns a Q-function and a policy
  - Uses off-policy data and the Bellman equation to learn the Q-function
  - Uses the Q-function to learn the policy
- Q-learning $$Q^*(s,a) = \mathop{\mathrm{E}}_{s' \sim P}\left[r(s,a) + \gamma \max_{a'} Q^*(s',a') \right]$$

Approximator $Q_\phi(s,a)$ Collect a set of transitions (s,a,r,s',d)

mean-squared Bellman error (MSBE)

$$L(\phi, \mathcal{D}) = \mathop{\mathrm{E}}_{(s,a,r,s',d) \sim \mathcal{D}} \left[\left(Q_\phi(s,a) - \left(r + \gamma(1-d) \max_{a'} Q_\phi(s',a') \right) \right)^2 \right]$$

a policy $\mu(s)$ $\max_a Q(s,a) \approx Q(s,\mu(s))$

$$Q_\phi(s', \mu(s'))$$

Deep Deterministic Policy Gradient (DDPG)

- Trick one: replay buffers
  - Large enough to contain a wide range of experiences
- Trick two: target networks
  - The term is called target $r + \gamma(1-d)\max_{a'} Q_{\phi}(s', a')$
  - The target depends on the same parameters ϕ but with a time delay
  - Target network ϕ_{targ}

$$\phi_{\text{targ}} \leftarrow \rho\phi_{\text{targ}} + (1-\rho)\phi$$

  - Target policy network $\mu_{\theta_{\text{targ}}}$

Deep Deterministic Policy Gradient (DDPG)

- Q-learning in DDPG

$$L(\phi, \mathcal{D}) = \underset{(s,a,r,s',d)\sim\mathcal{D}}{\mathrm{E}}\left[\left(Q_\phi(s,a) - \left(r + \gamma(1-d)Q_{\phi_{\mathrm{targ}}}(s', \mu_{\theta_{\mathrm{targ}}}(s'))\right)\right)^2\right]$$

- Policy learning in DDPG

$$\max_\theta \underset{s\sim\mathcal{D}}{\mathrm{E}}\left[Q_\phi(s, \mu_\theta(s))\right]$$

Gradient Ascent

Deep Deterministic Policy Gradient (DDPG)

Algorithm 1 Deep Deterministic Policy Gradient

Input: initial policy parameters θ, Q-function parameters ϕ, empty replay buffer $\mathcal{D}$

Set target parameters equal to main parameters $\theta_{\text{targ}} \leftarrow \theta$, $\phi_{\text{targ}} \leftarrow \phi$

repeat

  Observe state s and select action $a = \text{clip}(\mu_\theta(s) + \epsilon, a_{Low}, a_{High})$, where $\epsilon \sim \mathcal{N}$

  Execute a in the environment

  Observe next state s', reward r, and done signal d to indicate whether s' is terminal

  Store (s, a, r, s', d) in replay buffer $\mathcal{D}$

  If s' is terminal, reset environment state.

  if it's time to update **then**

    for however many updates **do**

      Randomly sample a batch of transitions, $B = \{(s, a, r, s', d)\}$ from $\mathcal{D}$

      Compute targets

$$y(r, s', d) = r + \gamma(1-d)Q_{\phi_{\text{targ}}}(s', \mu_{\theta_{\text{targ}}}(s'))$$

      Update Q-function by one step of gradient descent using

$$\nabla_\phi \frac{1}{|B|} \sum_{(s,a,r,s',d)\in B} \left(Q_\phi(s,a) - y(r,s',d)\right)^2$$

      Update policy by one step of gradient ascent using

$$\nabla_\theta \frac{1}{|B|} \sum_{s \in B} Q_\phi(s, \mu_\theta(s))$$

      Update target networks with

$$\phi_{\text{targ}} \leftarrow \rho\phi_{\text{targ}} + (1-\rho)\phi$$
$$\theta_{\text{targ}} \leftarrow \rho\theta_{\text{targ}} + (1-\rho)\theta$$

    end for

  end if

until convergence

Twin Delayed DDPG (TD3)

- Trick one: clipped double-Q learning
  - TD3 learns two Q functions
  - uses the smaller of the two Q-values to form the targets in the Bellman error loss functions

Q-learning suffers from **overestimation bias**

$$y(r, s', d) = r + \gamma(1 - d) \min_{i=1,2} Q_{\phi_{i,\text{targ}}}(s', a'(s'))$$

- Trick two: “delayed” policy updates
  - Updates the policy (and target networks) less frequently than the Q-function
- Trick three: target policy smoothing
  - Adds noise to the target action, to make it harder for the policy to exploit Q-function errors by smoothing out Q along changes in action

$$a'(s') = \text{clip}\left(\mu_{\theta_{\text{targ}}}(s') + \text{clip}(\epsilon, -c, c), a_{Low}, a_{High}\right), \qquad \epsilon \sim \mathcal{N}(0, \sigma)$$

Soft Actor-Critic (SAC)

$$\log \pi_\theta(a|s) = \log \left[P_\theta(s)\right]_a$$

$$\log \pi_\theta(a|s) = -\frac{1}{2}\left(\sum_{i=1}^{k}\left(\frac{(a_i-\mu_i)^2}{\sigma_i^2} + 2\log\sigma_i\right) + k\log 2\pi\right)$$

- An algorithm that optimizes a stochastic policy in an off-policy way
- Entropy-regularized RL

$$\pi^* = \arg\max_{\pi} \underset{\tau\sim\pi}{\mathrm{E}}\left[\sum_{t=0}^{\infty}\gamma^t\Big(R(s_t,a_t,s_{t+1}) + \alpha H\left(\pi(\cdot|s_t)\right)\Big)\right]$$

Entropy $H(P) = \underset{x\sim P}{\mathrm{E}}\left[-\log P(x)\right]$

increasing entropy results in more exploration, which can accelerate learning later on

$$V^\pi(s) = \underset{\tau\sim\pi}{\mathrm{E}}\left[\sum_{t=0}^{\infty}\gamma^t\Big(R(s_t,a_t,s_{t+1}) + \alpha H\left(\pi(\cdot|s_t)\right)\Big)\middle| s_0 = s\right]$$

$$V^\pi(s) = \underset{a\sim\pi}{\mathrm{E}}\left[Q^\pi(s,a)\right] + \alpha H\left(\pi(\cdot|s)\right)$$

$$Q^\pi(s,a) = \underset{\tau\sim\pi}{\mathrm{E}}\left[\sum_{t=0}^{\infty}\gamma^t R(s_t,a_t,s_{t+1}) + \alpha\sum_{t=1}^{\infty}\gamma^t H\left(\pi(\cdot|s_t)\right)\middle| s_0 = s, a_0 = a\right]$$

Soft Actor-Critic (SAC)

- SAC learns a policy and two Q-functions
  - Uses entropy regularization
  - Train a stochastic policy

$$
\begin{aligned}
Q^{\pi}(s,a) &= \underset{\substack{s' \sim P \\ a' \sim \pi}}{\mathrm{E}} \left[R(s,a,s') + \gamma \left(Q^{\pi}(s',a') + \alpha H\left(\pi(\cdot|s') \right) \right) \right] \\
&= \underset{\substack{s' \sim P \\ a' \sim \pi}}{\mathrm{E}} \left[R(s,a,s') + \gamma \left(Q^{\pi}(s',a') - \alpha \log \pi(a'|s') \right) \right]
\end{aligned}
$$

Approximate expectation with samples $Q^{\pi}(s,a) \approx r + \gamma \left(Q^{\pi}(s',\tilde{a}') - \alpha \log \pi(\tilde{a}'|s') \right), \qquad \tilde{a}' \sim \pi(\cdot|s')$

Soft Actor-Critic (SAC)

- Q-learning

$$L(\phi_i, \mathcal{D}) = \underset{(s,a,r,s',d)\sim\mathcal{D}}{\mathrm{E}}\left[\left(Q_{\phi_i}(s,a) - y(r,s',d)\right)^2\right]$$

$$y(r,s',d) = r + \gamma(1-d)\left(\min_{j=1,2} Q_{\phi_{\mathrm{targ},j}}(s',\tilde{a}') - \alpha \log \pi_\theta(\tilde{a}'|s')\right), \qquad \tilde{a}' \sim \pi_\theta(\cdot|s')$$

- Policy learning

maximize

$$\begin{aligned} V^\pi(s) &= \underset{a\sim\pi}{\mathrm{E}}\left[Q^\pi(s,a)\right] + \alpha H\left(\pi(\cdot|s)\right) \\ &= \underset{a\sim\pi}{\mathrm{E}}\left[Q^\pi(s,a) - \alpha \log \pi(a|s)\right] \end{aligned}$$

The policy is learned by maximizing the **soft value function**

Soft Actor-Critic (SAC)

- Policy learning

reparameterization trick $\tilde{a}_\theta(s, \xi) = \tanh\left(\mu_\theta(s) + \sigma_\theta(s) \odot \xi\right), \qquad \xi \sim \mathcal{N}(0, I)$

$$\underset{a \sim \pi_\theta}{\mathrm{E}}\left[Q^{\pi_\theta}(s, a) - \alpha \log \pi_\theta(a|s)\right] = \underset{\xi \sim \mathcal{N}}{\mathrm{E}}\left[Q^{\pi_\theta}(s, \tilde{a}_\theta(s, \xi)) - \alpha \log \pi_\theta(\tilde{a}_\theta(s, \xi)|s)\right]$$

$$\max_\theta \underset{\substack{s \sim \mathcal{D} \\ \xi \sim \mathcal{N}}}{\mathrm{E}}\left[\min_{j=1,2} Q_{\phi_j}(s, \tilde{a}_\theta(s, \xi)) - \alpha \log \pi_\theta(\tilde{a}_\theta(s, \xi)|s)\right]$$

Soft Actor-Critic (SAC)

Algorithm 1 Soft Actor-Critic

Input: initial policy parameters θ, Q-function parameters ϕ_1, ϕ_2, empty replay buffer $\mathcal{D}$
Set target parameters equal to main parameters $\phi_{\text{targ},1} \leftarrow \phi_1$, $\phi_{\text{targ},2} \leftarrow \phi_2$
repeat
  Observe state s and select action $a \sim \pi_\theta(\cdot|s)$
  Execute a in the environment
  Observe next state s', reward r, and done signal d to indicate whether s' is terminal
  Store (s, a, r, s', d) in replay buffer $\mathcal{D}$
  If s' is terminal, reset environment state.
  if it's time to update **then**
    for j in range(however many updates) **do**
      Randomly sample a batch of transitions, $B = \{(s, a, r, s', d)\}$ from $\mathcal{D}$
      Compute targets for the Q functions:

$$y(r, s', d) = r + \gamma(1-d)\left(\min_{i=1,2} Q_{\phi_{\text{targ},i}}(s', \tilde{a}') - \alpha \log \pi_\theta(\tilde{a}'|s')\right), \quad \tilde{a}' \sim \pi_\theta(\cdot|s')$$

      Update Q-functions by one step of gradient descent using

$$\nabla_{\phi_i} \frac{1}{|B|} \sum_{(s,a,r,s',d)\in B} \left(Q_{\phi_i}(s,a) - y(r,s',d)\right)^2 \qquad \text{for } i = 1, 2$$

      Update policy by one step of gradient ascent using

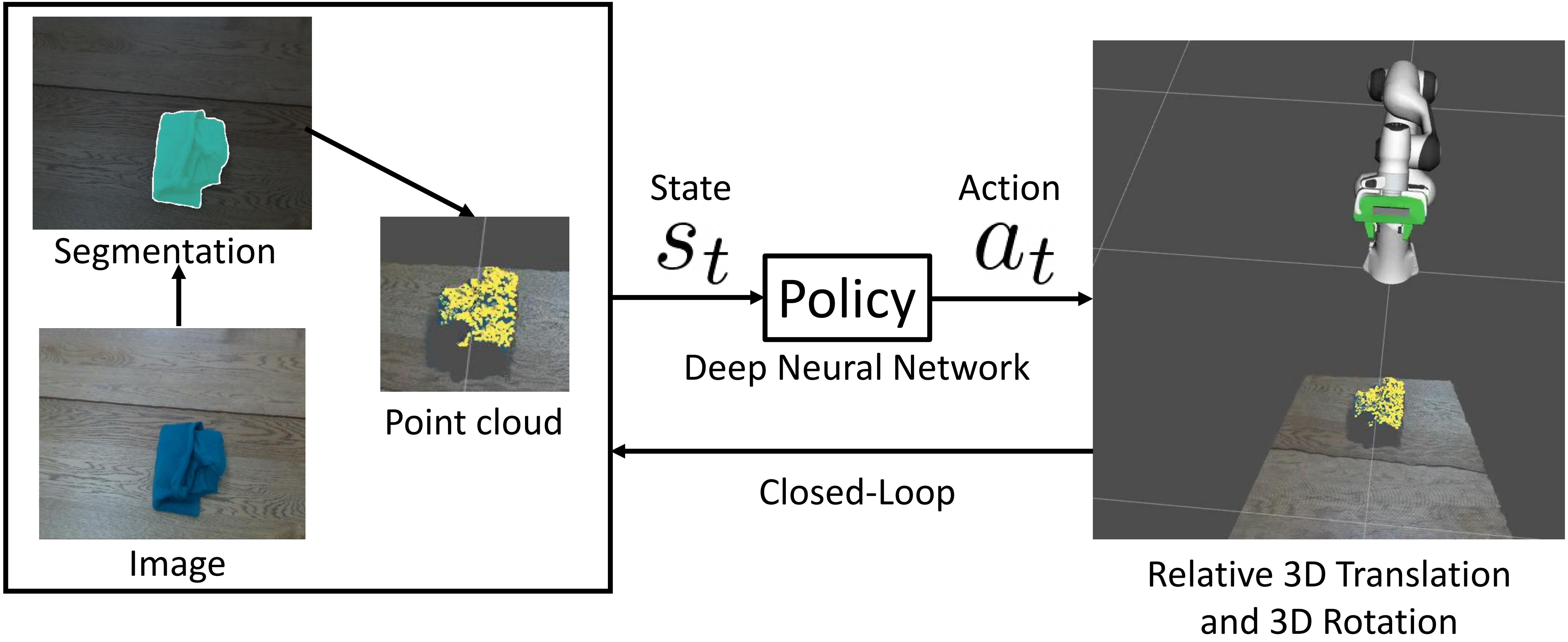
$$\nabla_\theta \frac{1}{|B|} \sum_{s\in B} \left(\min_{i=1,2} Q_{\phi_i}(s, \tilde{a}_\theta(s)) - \alpha \log \pi_\theta\left(\tilde{a}_\theta(s)|\, s\right)\right),$$

      where $\tilde{a}_\theta(s)$ is a sample from $\pi_\theta(\cdot|s)$ which is differentiable wrt θ via the reparametrization trick.
      Update target networks with

$$\phi_{\text{targ},i} \leftarrow \rho\phi_{\text{targ},i} + (1-\rho)\phi_i \qquad \text{for } i = 1, 2$$

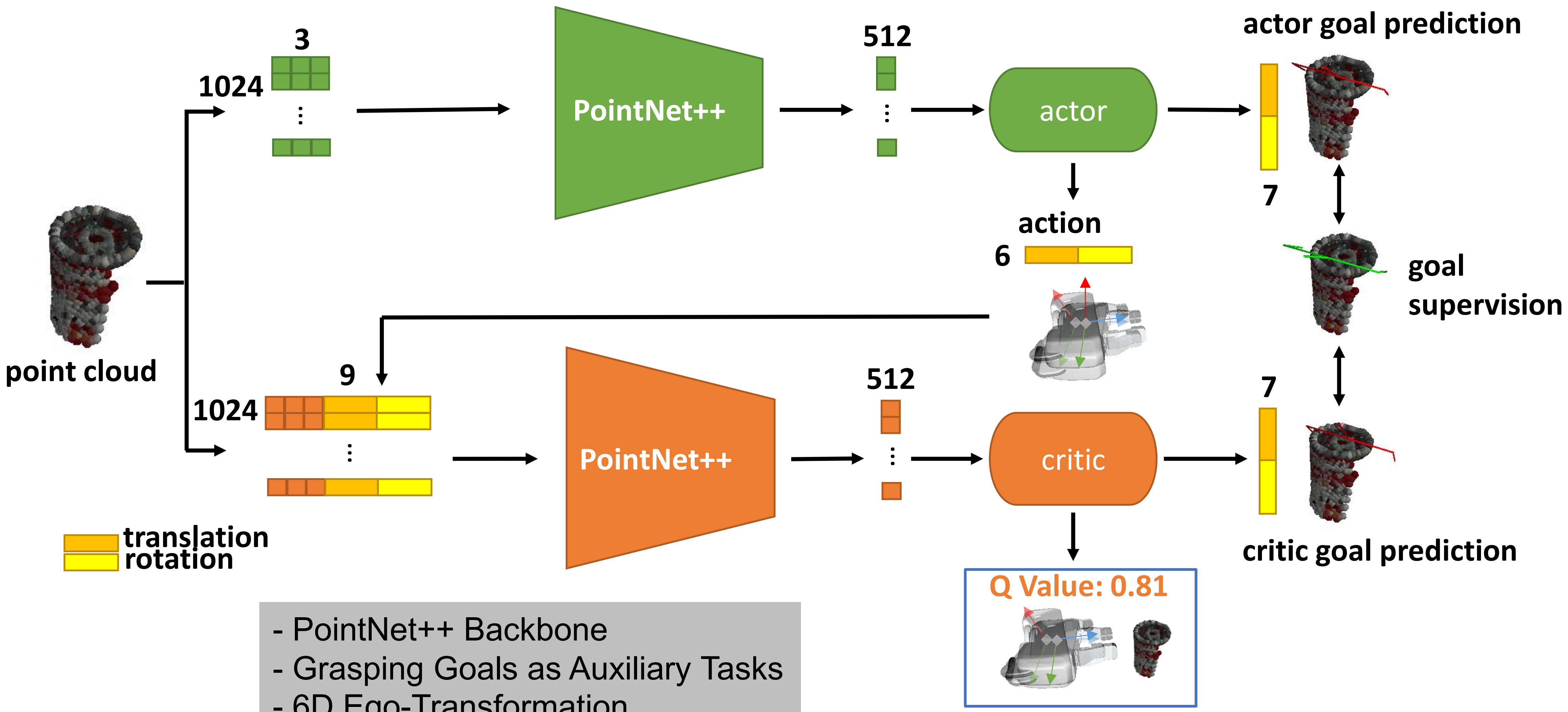
    end for
  end if
until convergence

Learning Closed-Loop Control Polices for 6D Grasping



Goal-Auxiliary Actor-Critic for 6D Robotic Grasping with Point Clouds. Wang-Xiang-Yang-Mousavian-Fox, CoRL'21

GA-DDPG Network Architecture

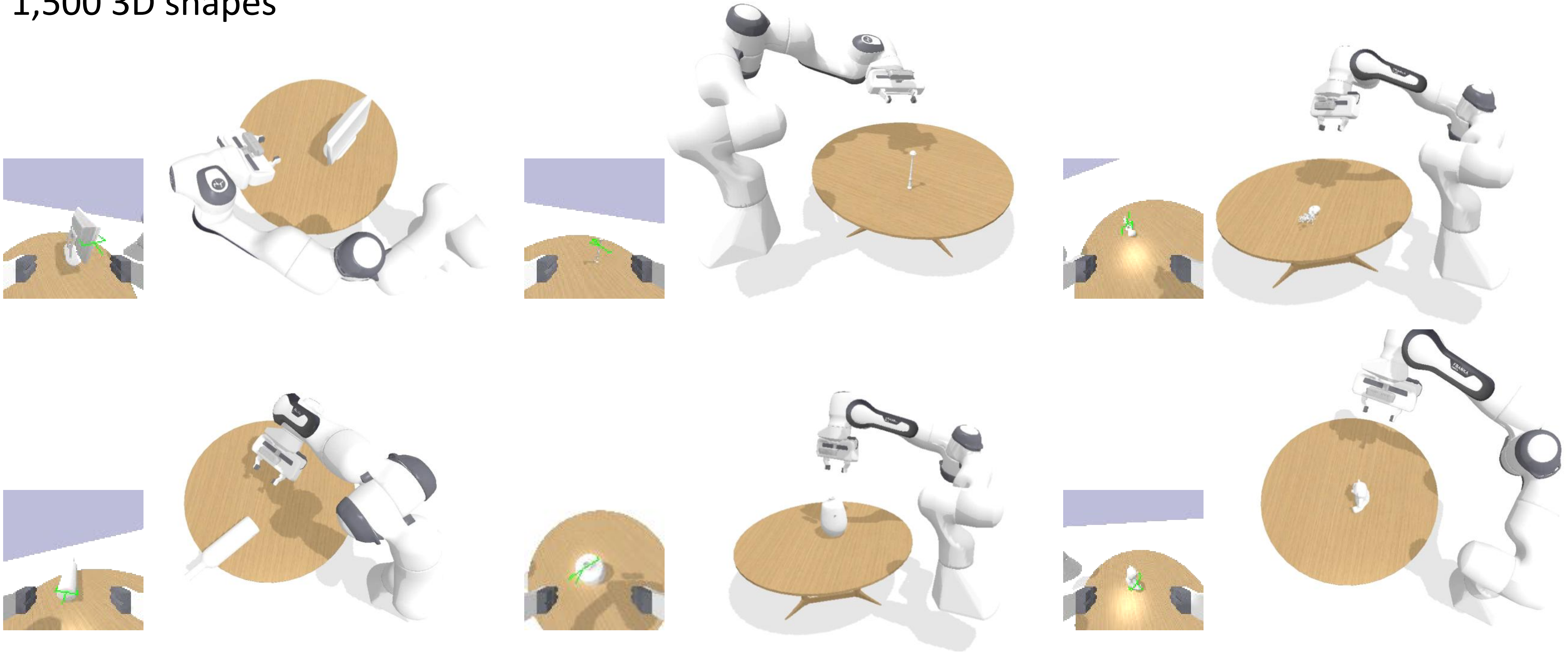


- PointNet++ Backbone
- Grasping Goals as Auxiliary Tasks
- 6D Ego-Transformation

Learning from Demonstration with the OMG-Planner

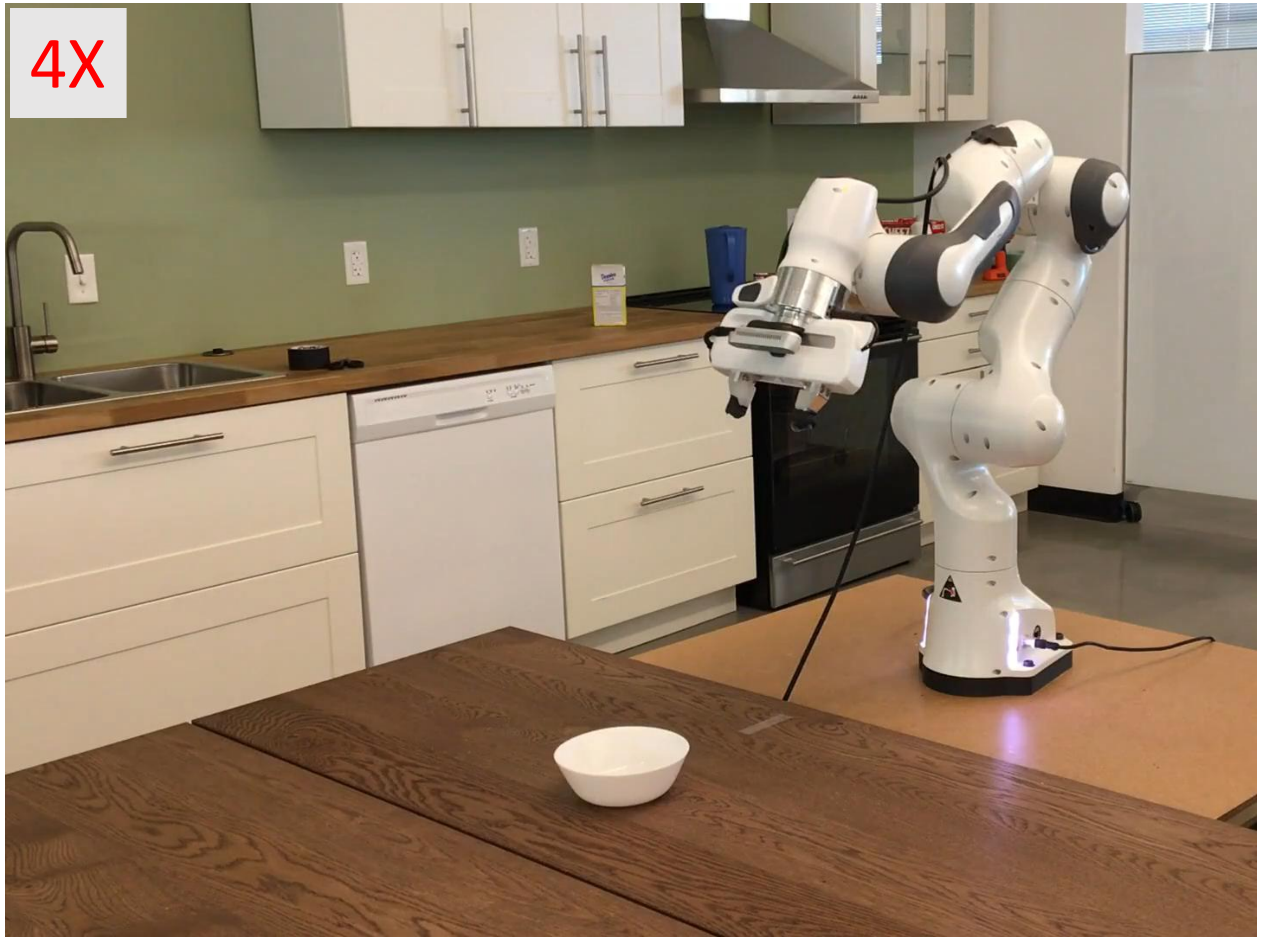
50,000 trajectories

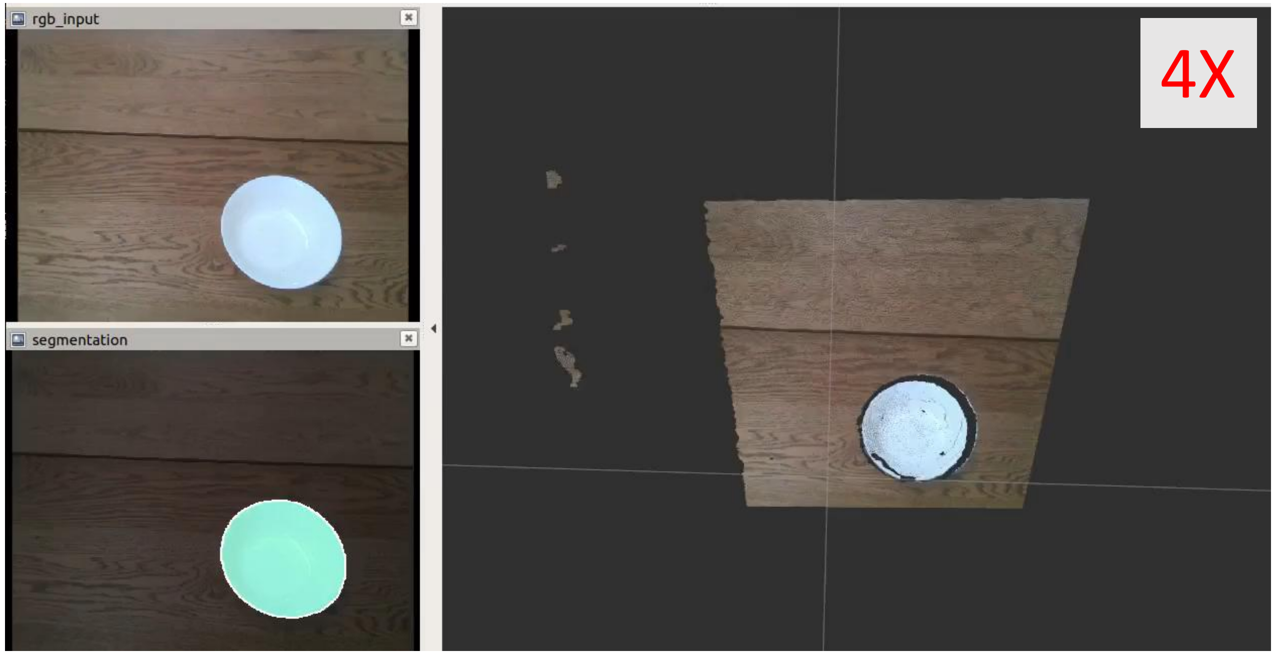
1,500 3D shapes



Goal-Auxiliary Actor-Critic for 6D Robotic Grasping with Point Clouds. Wang-Xiang-Yang-Mousavian-Fox, CoRL’21

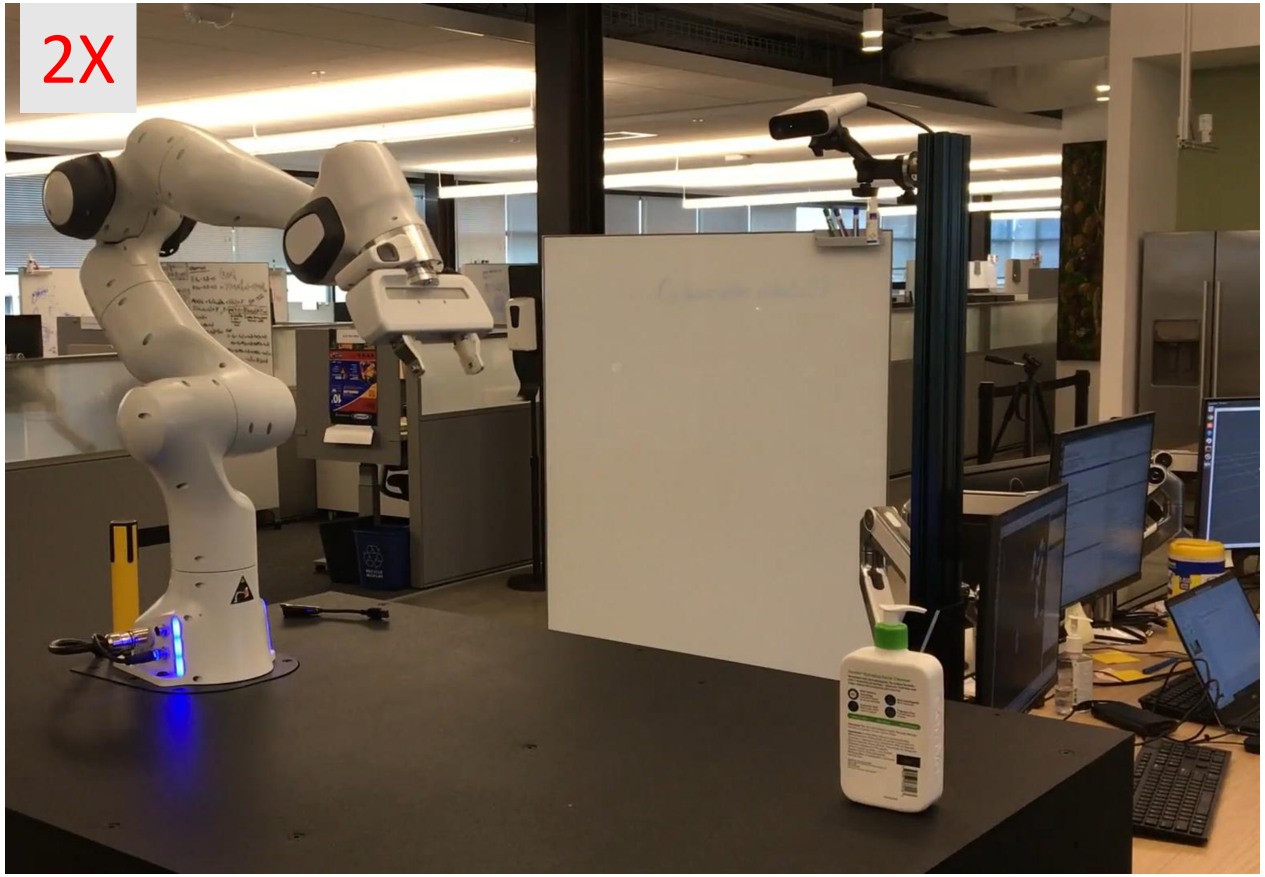
Our Learned Policy in the Real World

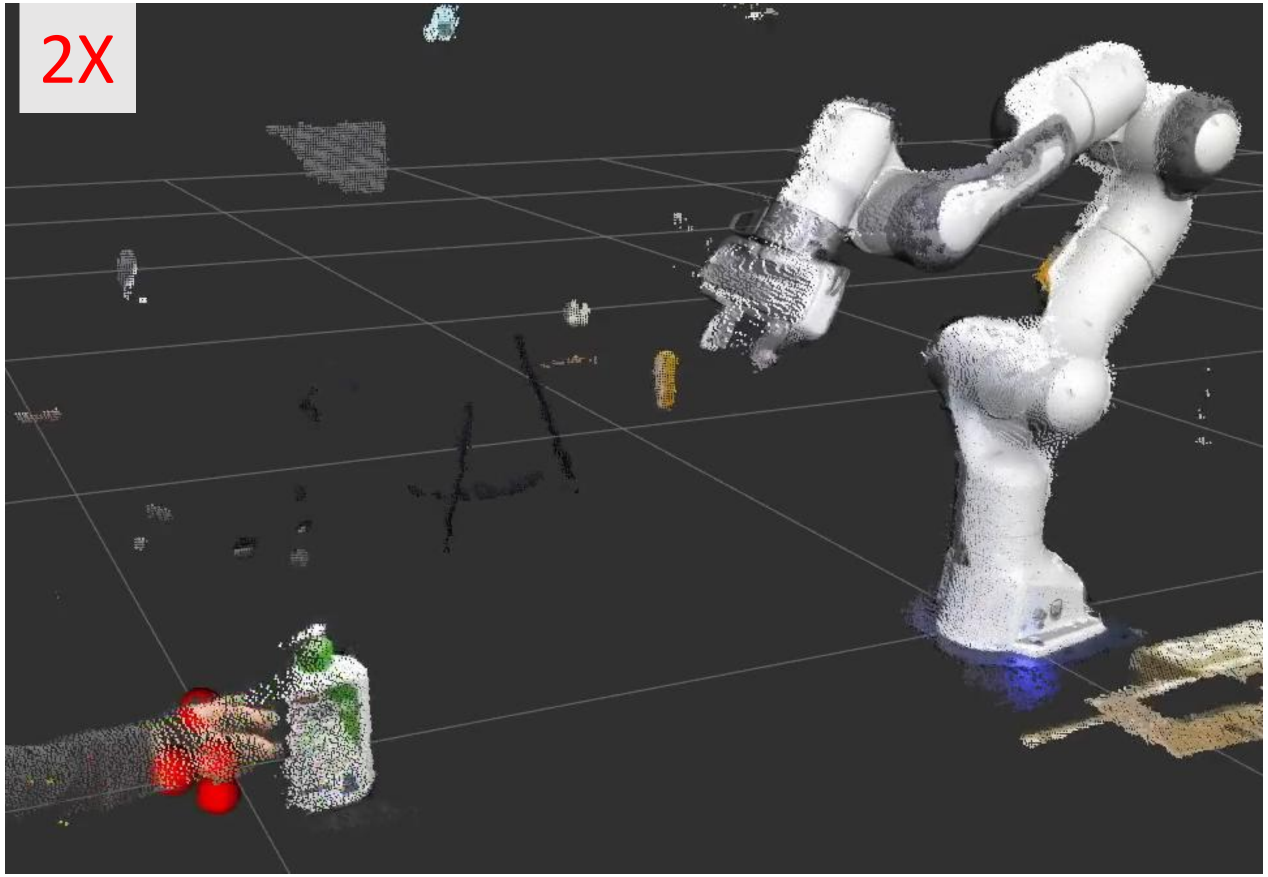




Goal-Auxiliary Actor-Critic for 6D Robotic Grasping with Point Clouds. Wang-Xiang-Yang-Mousavian-Fox, CoRL’21

Closed-Loop Human-Robot Handover





Goal-Auxiliary Actor-Critic for 6D Robotic Grasping with Point Clouds. Wang-Xiang-Yang-Mousavian-Fox, CoRL'21

Summary

- Model-free RL
  - Deep Deterministic Policy Gradient (DDPG)
  - Twin Delayed DDPG (TD3)
  - Soft Actor-Critic (SAC)

Further Reading

- OpenAI Spinning Up in Deep RL
  https://spinningup.openai.com/en/latest/index.html