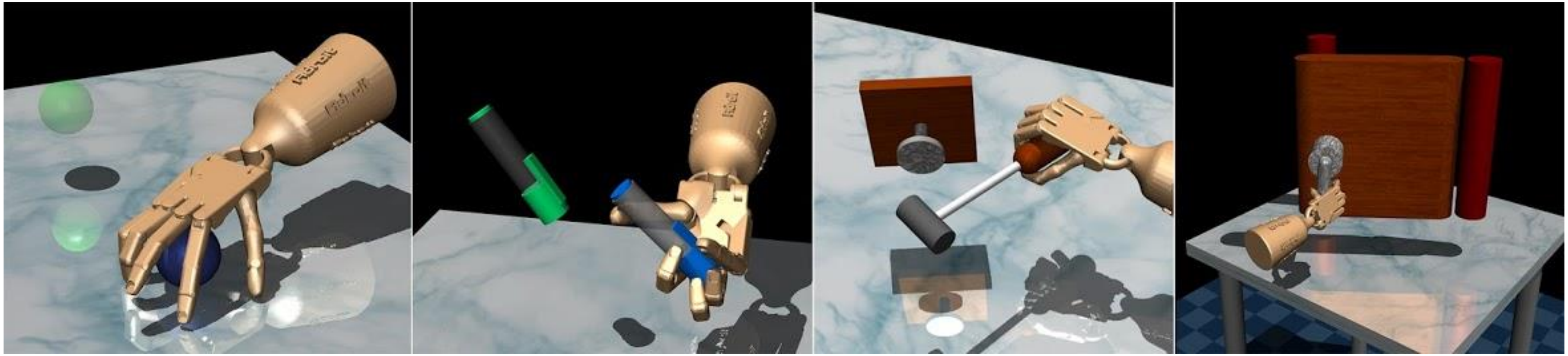
Learning Complex Dexterous Manipulation with Deep Reinforcement Learning and Demonstrations



Rajeswaran et al.

Robotics: Science and Systems (RSS) 2018

Deep RL in Robotics

- Dexterous manipulation at a joint level requires manual reward shaping and lots of samples
  - Requires training in simulation due to sample complexity
  - Prior on-policy RL methods: limited to manipulators with 7-10 DoF on simpler tasks

- **Demo Augmented Policy Gradients (DAPG)**: imitation learning + deep RL to reduce samples required to solve high-dimensional control problems

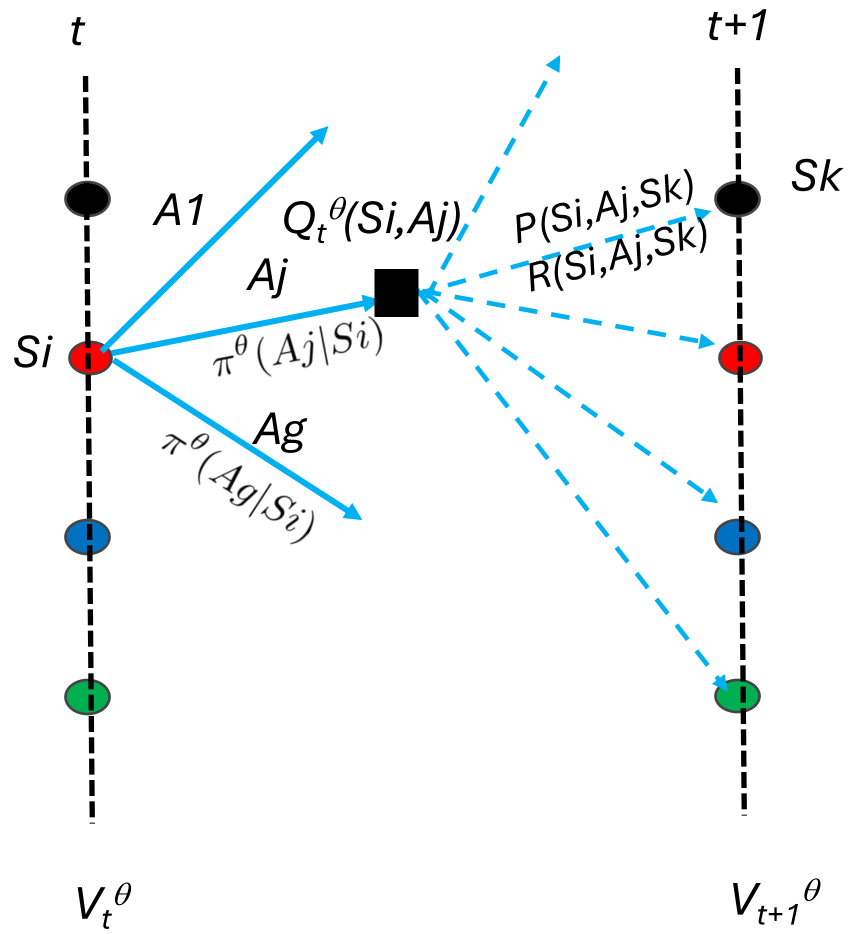
Policies in RL

- Deterministic control:
  - **Policy:** State → Action

$$\pi : S \to A$$

- Probabilistic control (**policy gradient methods**):
  - **Policy network:** maps current state to action distribution, parameterized by Θ

$$\pi_\theta : S \to \mathbb{P}(A) (\text{written as } \pi_\theta(Aj|Si))$$



Probabilistic control

Policy distribution: $\pi^\theta(Aj|Si)$

$Q_t^\theta(Si,Aj) = \sum_{k=0}^{|S|-1}[V_{t+1}^\theta(Sk)+R(Si,Aj,Sk)] * P(Si,Aj,Sk)$

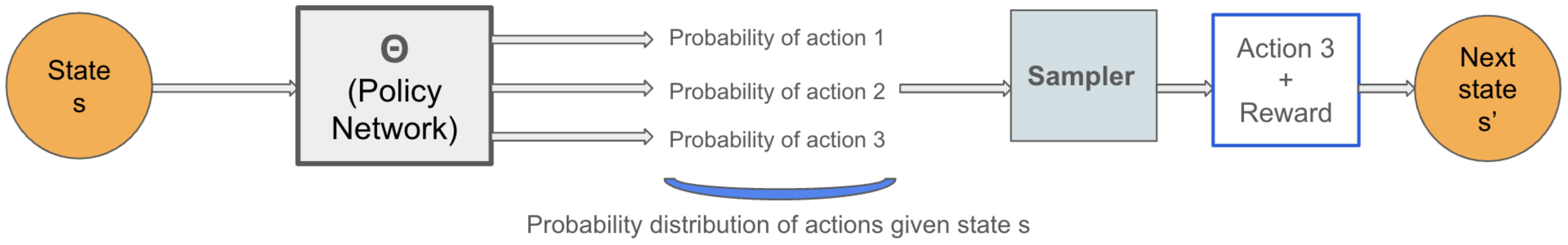
$V_t^\theta(Si) = \sum_{j=0}^{|A|-1} \pi^\theta(Aj|Si) Q_t^\theta(Si,Aj)$

$V_T^\theta(Si) = 0$

Policy improvement: change θ to promote actions with higher Q-values

Policy Networks

- Policy networks implement policies to sample actions/rewards given a state

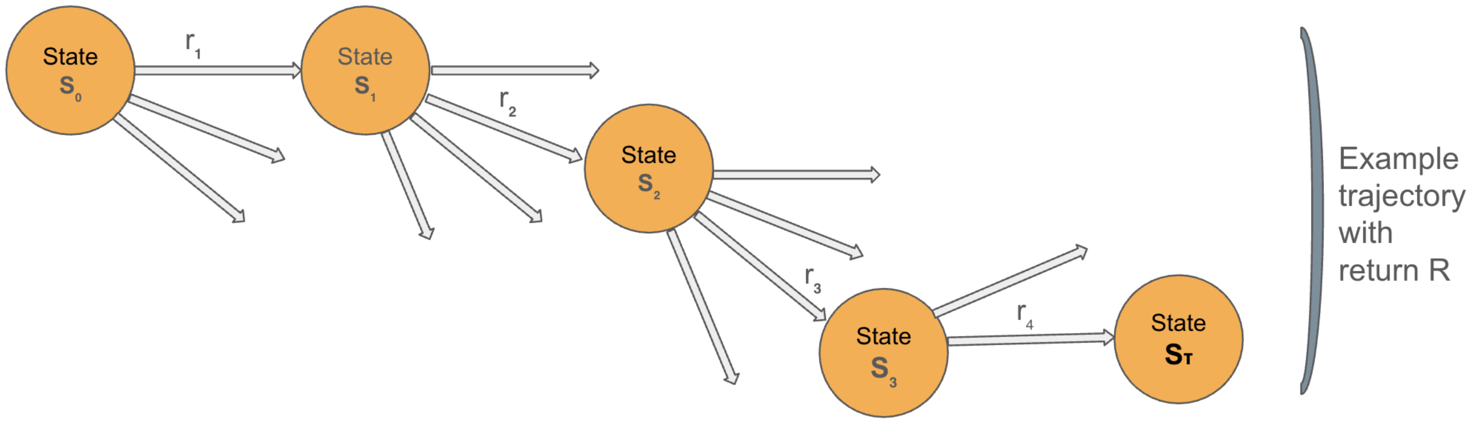


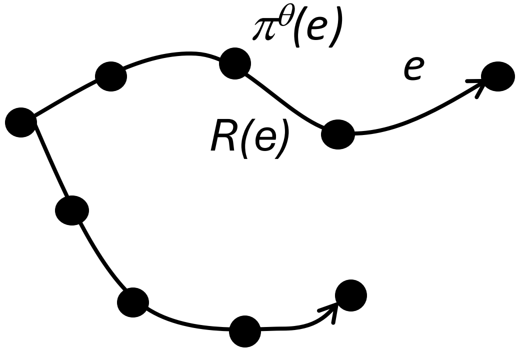
- **On-policy** if *current* policy is used for sampling, **off-policy** if *any* policy is used for sampling
  - On-policy methods discard old policy samples and are sample inefficient; off-policy methods require a buffer to store samples

Training Policy Networks

- **How do we update Θ based on observed trajectories?**
  - This is similar to **supervised learning** methods

- Sample multi-hop trajectories for a policy π and treat sampled trajectories as training data
  - If $R(e) > 0$, update to Θ increases probability of $\pi^{\Theta}(e)$ – path becomes more probable
  - If $R(e) < 0$, update to Θ decreases probability of $\pi^{\Theta}(e)$ – path becomes less probable





Policy Objective Function

- Use **gradient ascent** to iteratively adjust policy parameters θ to maximize the expected return, **policy objective function J(θ)**

$$\max_{\theta} J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta}[R(\tau)] = \sum_{\underset{\text{All possible trajectories } \tau \text{ consistent with policy } \pi}{\boxed{\tau}}} \underbrace{P(\tau;\theta)}_{\text{Probability of trajectory}} \underbrace{R(\tau)}_{\text{Cumulative discounted return from trajectory}}$$

- Find the gradient of the policy objective function:

$$\nabla_\theta J(\theta) = \nabla_\theta \sum_{\tau} P(\tau;\theta) R(\tau)$$

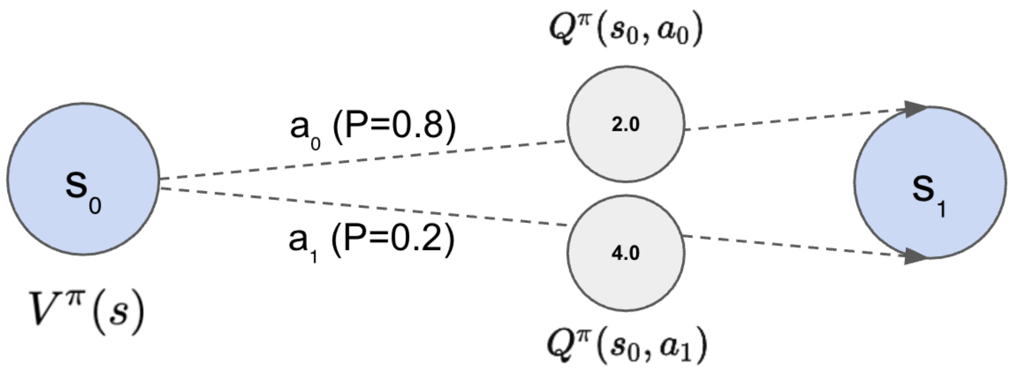
- **Gradient ascent update,** where α is the step size:

$$\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$$

Advantage

- **Key idea:** What is the relative *advantage* of an action compared to other actions we've seen given a state?
- **Goal:** Take actions with more rewards than previous actions

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



Assuming $Q^\pi(s_0, a_0) = 2.0$ and $Q^\pi(s_0, a_1) = 4.0$

$$V^\pi(s_0) = 2 * 0.8 + 4 * 0.2 = 2.4$$

$$A^\pi(s_0, a_0) = 2.0 - 2.4 = -0.4$$

$$A^\pi(s_0, a_1) = 4.0 - 2.4 = 1.6$$

$\rightarrow$ We want to increase the likelihood of a_1

Policy Gradient Methods

- Policy gradient with advantage:

$$g = \frac{1}{NT} \sum_{i=1}^{N} \sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(a_t^i | s_t^i) \hat{A}^\pi(s_t^i, a_t^i, t).$$

- **Problem:** Sample complexity
  - Complex, multi-step tasks require extensive exploration to discover high-reward actions

- **One solution:** Incorporate human priors to "kickstart" learning
  - Reward shaping (instead of sparse task completion rewards)
    Manual and labor intensive
  - Use demonstrations

Demonstrations

- Abstraction of demonstrations:

$$\boxed{\rho_D} = \left\{ \left(\boxed{s_t^{(i)}, a_t^{(i)}, s_{t+1}^{(i)}}, \boxed{r_t^{(i)}} \right) \right\}$$

Dataset of i demonstrations

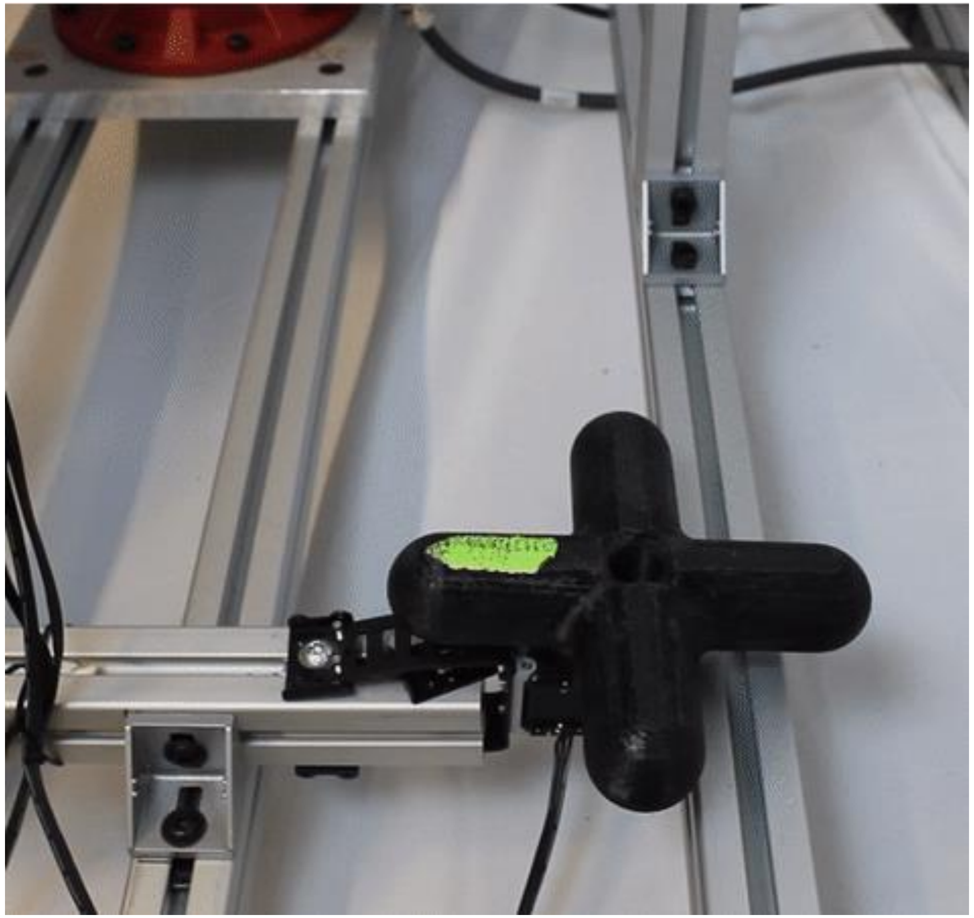
Transition from s_t to s_{t+1}

Shaped reward

- Given an expert trajectory

$\tau_i = \{(s_1, a_1), (s_2, a_2), (s_3, a_3), (s_4, a_4)\} \in \rho_D$

we want to learn a policy $\pi_\theta(a \mid s)$



Behavior Cloning

- How do we incorporate information from demonstrations into training?
  - **Behavior Cloning:** $\boxed{\underset{\theta}{\text{maximize}} \sum_{(s,a)\in\rho_D} \ln \pi_\theta(a|s)}$

The likelihood of the trajectory under policy π_θ is:

$$L(\theta) = \prod_{t=1}^{T} \pi_{\theta(}a_t|s_{\ t})$$

Take log to convert $L(\theta)$ into a sum:

$$log\ \ L(\theta)\ = \sum_{t=1}^{T} log\ \pi_{\theta(}a_t|s_t)$$

Expert trajectory $\boldsymbol{\tau_i = \{(s_1, a_1), (s_2, a_2), (s_3, a_3), (s_4, a_4)\} \in \rho_D}$

$$\boldsymbol{s_1} \xrightarrow[\boldsymbol{r_1}]{\boldsymbol{a_1}} \boldsymbol{s_2} \xrightarrow[\boldsymbol{r_2}]{\boldsymbol{a_2}} \boldsymbol{s_3} \xrightarrow[\boldsymbol{r_3}]{\boldsymbol{a_3}} \boldsymbol{s_4}$$

$$log\ L(\theta) = log\,\pi_{\theta(}a_1|s_1) + log\,\pi_{\theta(}a_2|s_2) +\ \ log\,\pi_{\theta(}a_3|s_3)$$
$$= [\pi_{\theta(}a_1 \mid s_1) \cdot \pi_\theta(a_2 \mid s_2) \cdot \pi_\theta(a_3 \mid s_3)]$$

Using Demonstrations

- Pre-train policy with BC
  - **Problem:** Cloned policies are *not* successful without further training
- Augment PG objective with BC objective to make use of expert data later in training

→ Demo Augmented Policy Gradients (DAPG)

Demo Augmented Policy Gradients (DAPG)

Key Idea: use imitation learning to bootstrap and guide deep RL

- Augment the original objective with a weighted Behavior Cloning term

$$g_{aug} = \underbrace{\sum_{\boxed{(s,a)\in\rho_\pi}} \nabla_\theta \ln \pi_\theta(a|s) A^\pi(s,a)}_{\text{Policy Gradient}} + \underbrace{\sum_{\boxed{(s,a)\in\rho_D}} \nabla_\theta \ln \pi_\theta(a|s) \boxed{w(s,a)}}_{\text{Behavior Cloning Gradient}}$$

Data collected by the policy

Demonstration data

Policy Gradient

Behavior Cloning Gradient

- **Heuristic weighting scheme** $w(s,a)$ to decay BC contributions over time as our policy improves

$$\boxed{w(s,a) = \lambda_0 \boxed{\lambda_1^k} \boxed{\max_{(s',a')\in\rho_\pi} A^\pi(s',a')} \; \forall (s,a) \in \rho_D}$$

k = number of PG iterations

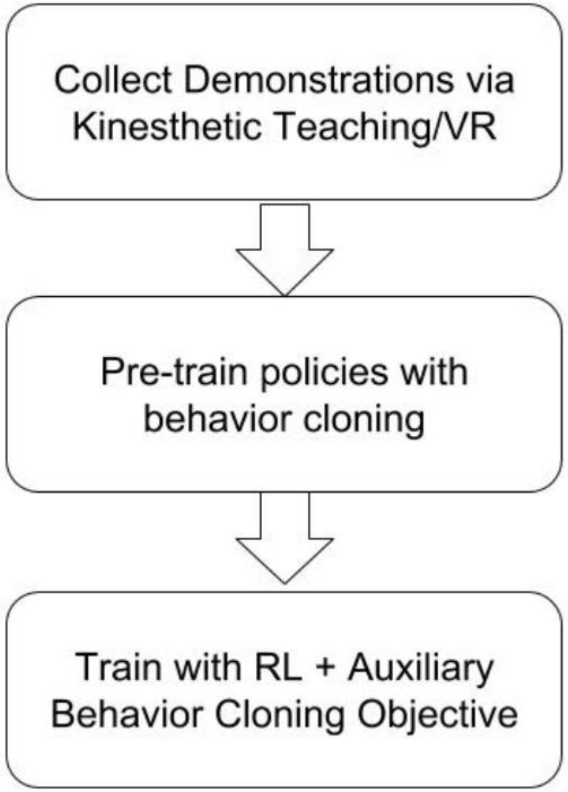
Highest advantage in data collected by PG → *approximation* to $A^\pi(s',a')$ for ρ_D

- $\lambda_0 = 0$, $w(s,a) = 1$ → **Behavior Cloning;** $\lambda_0 > 0$, $w(s,a) = 0$ → **RL**

Why DAPG?

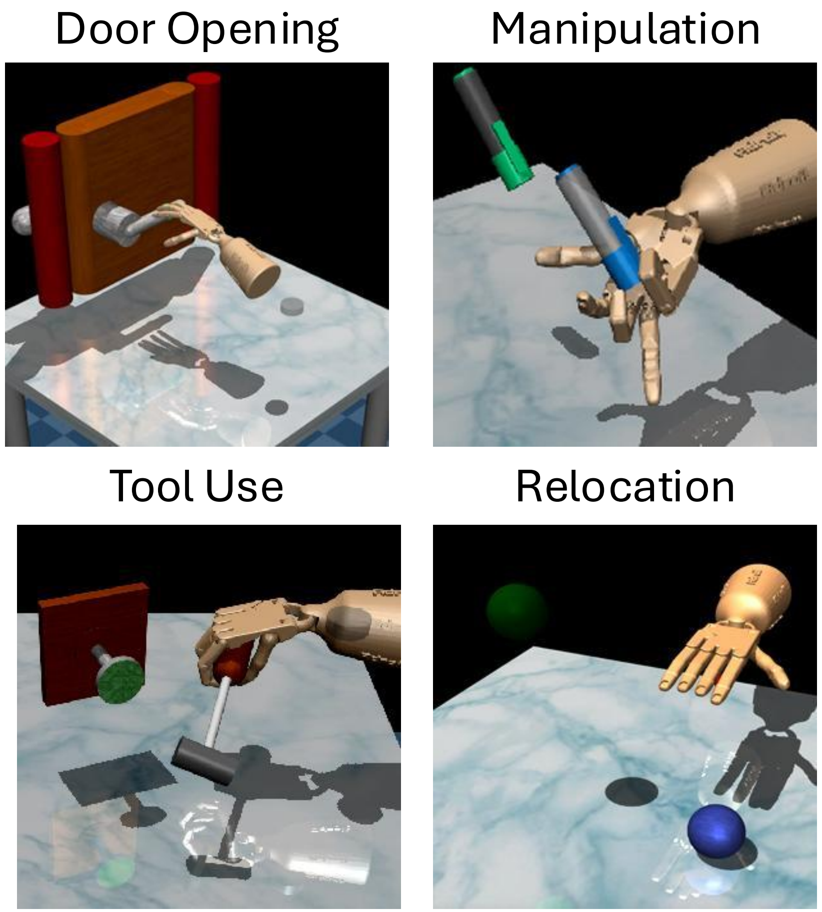
- Why BC + RL?
  - Eliminates reward shaping
  - Discovers more natural looking behaviors
  - Guides exploration
  - Decrease sample complexity

- Demonstrations are collected with VR in simulation
  - 25 demonstrations per task

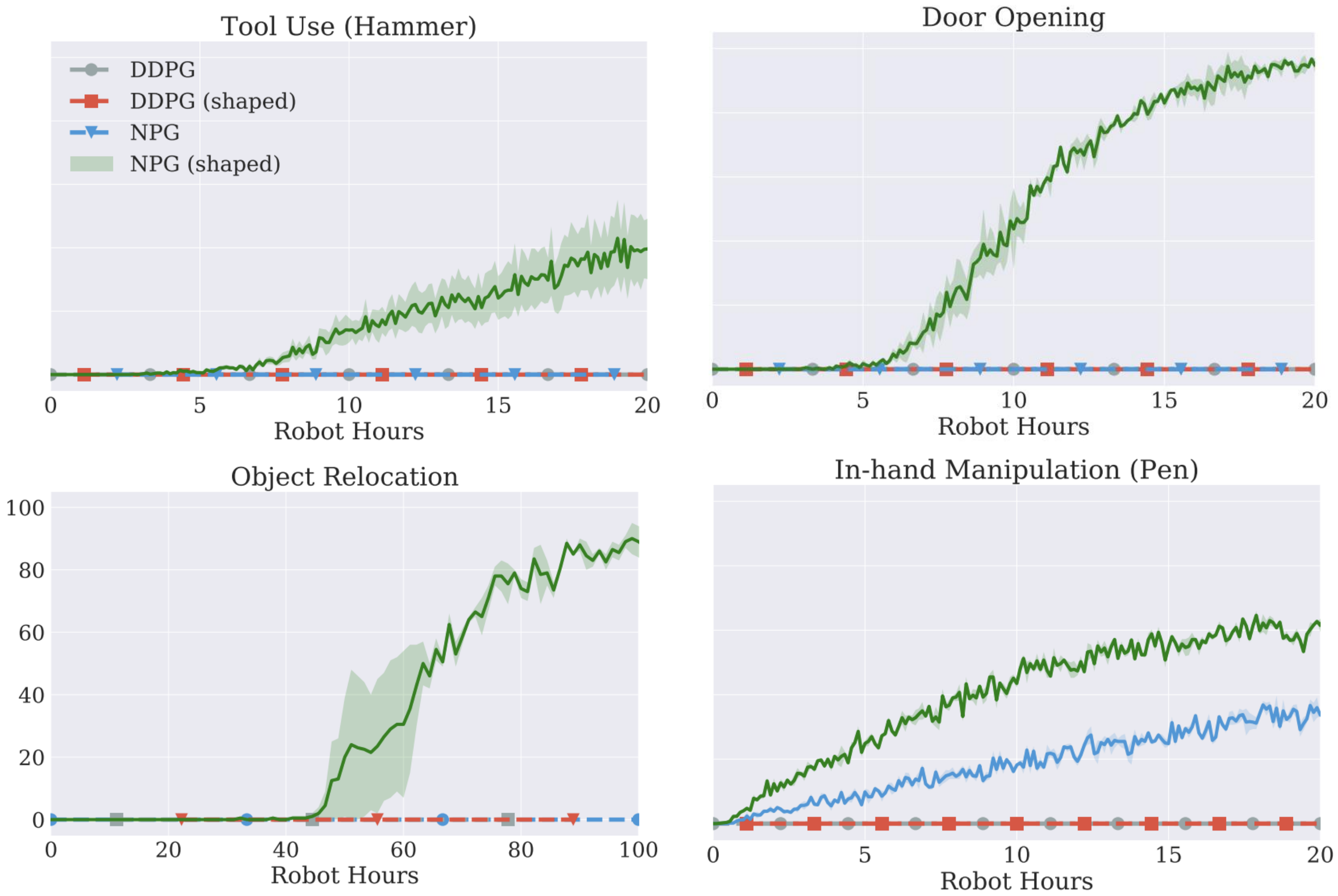


Demonstration Augmented Policy Gradient

Experiments

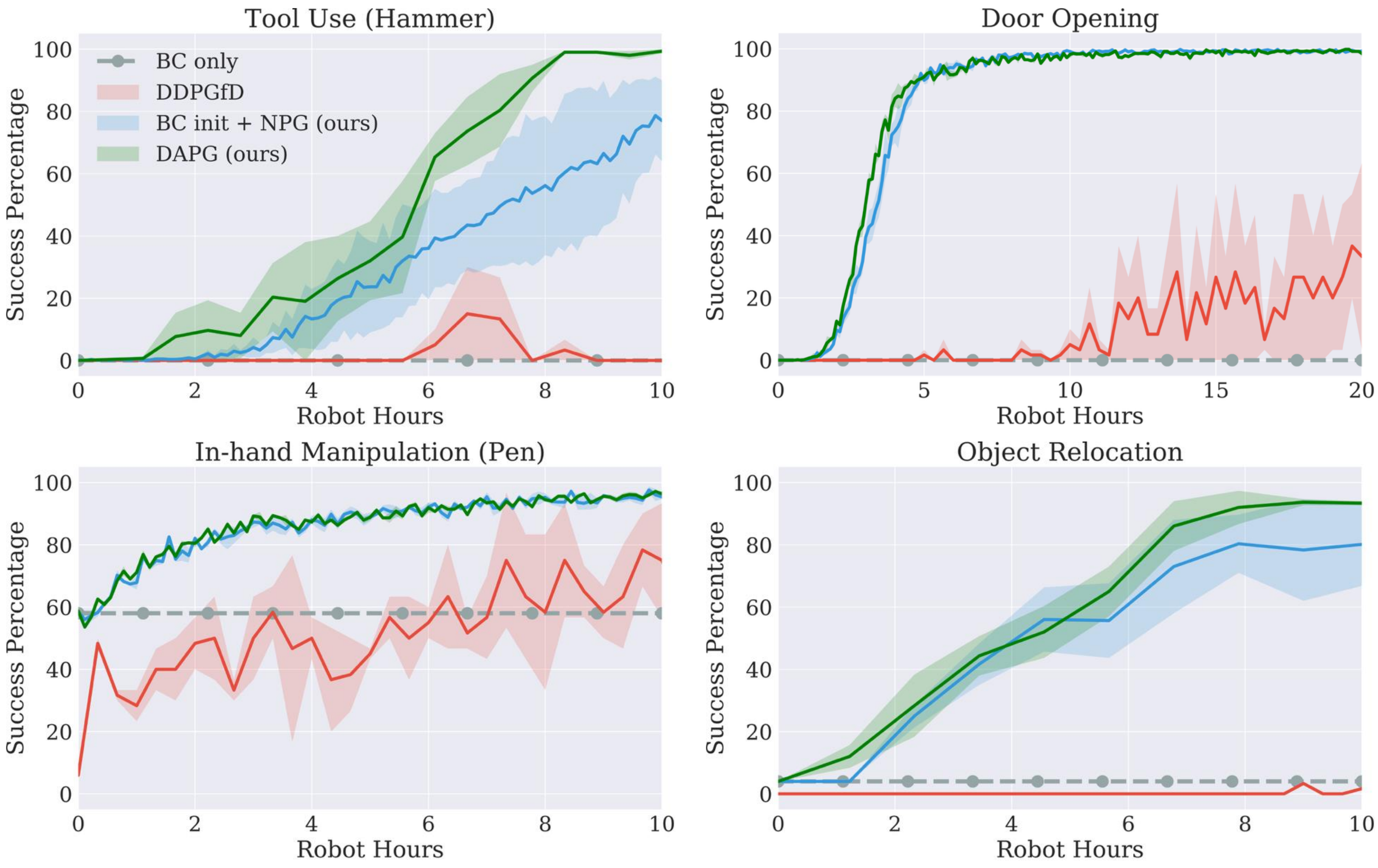


Solving tasks from *sparse* rewards is a hard **exploration** problem
- RL learns from the differences between outcomes
- If we never succeed, there will be nothing to learn
  - *Shaped* rewards can help



Experiments

- **Baseline:** DDPG from Demonstrations (DDPGfD)
  - Off-policy RL with demonstrations



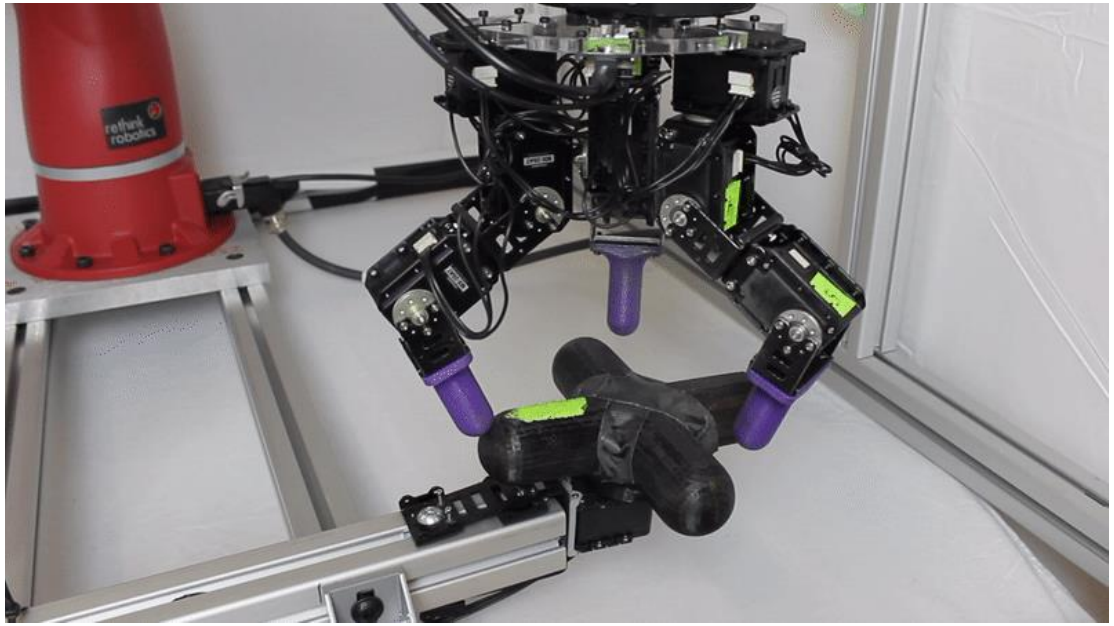
Results

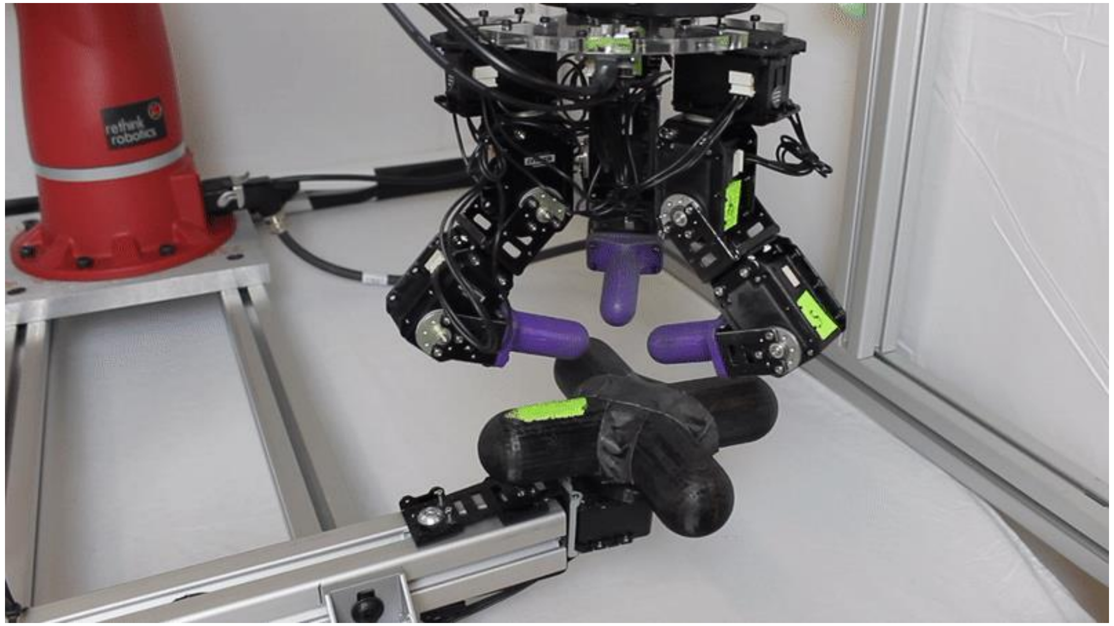
Method	DAPG	(sp)	RL	(sh)	RL	(sp)
Task	N	Hours	N	Hours	N	Hours
Relocation	52	5.77	880	98	∞	∞
Hammer	55	6.1	448	50	∞	∞
Door	42	4.67	146	16.2	∞	∞
Pen	30	3.33	864	96	2900	322

- PG methods take too long (on-policy NPG) or can't learn policies at all (off-policy DDPG) even when reward shaping is used to incorporate priors; BC-only policies are unsuccessful

- DAPG is 30x more sample efficient and 30x faster to train than PG alone
  - A small number of demonstrations can *significantly* reduce the sample complexity of PG methods
  - Initializing with BC alone increases performance, but using demonstration data for initialization *and* training performs best

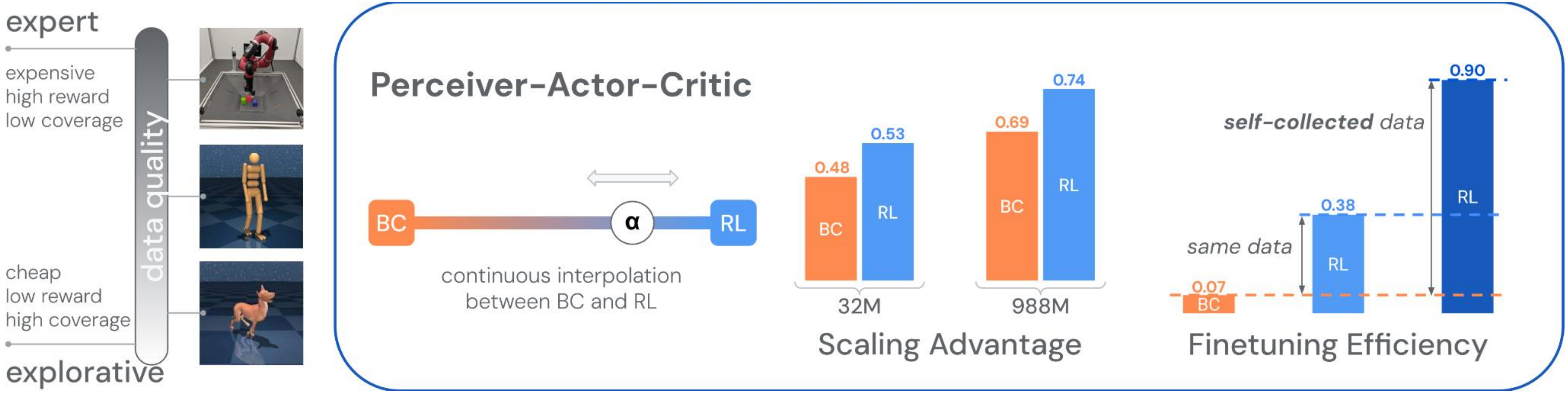
Sim-to-Real





Similar Work

Behavior-constrained policy updates turn BC into a lower bound to improve upon in an offline actor-critic RL setting



"*Offline Actor-Critic Reinforcement Learning Scales to Large Models*", Springenberg et al., 2024

Imitation Learning

- Use **demonstrations** to mimic expert behavior with supervised learning
- IL learns how to copy the expert, RL learns actions with high rewards
  - Problem: IL methods are only as good as the demonstrations

Imitation Learning

Online IL

Expert demonstrations are augmented with real-time expert interactions for dynamic error correction (e.g. **DAgger**)

Offline IL

The agent only has access to a fixed dataset of demonstrations, learning passively (e.g. **Behavior Cloning**)

Inverse IL

Infer the reward function from expert demonstrations instead of manually designing rewards