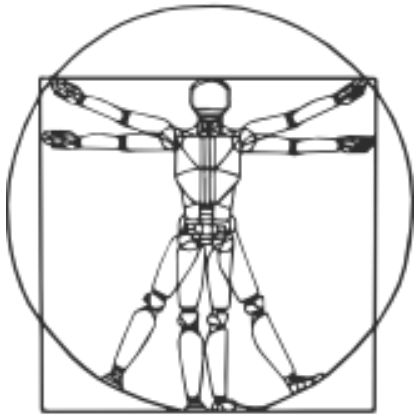


CSC2626
Imitation Learning for Robotics

Florian Shkurti

Week 9: Imitation Learning Combined with RL, Control, and Planning #2

Today's agenda

- Guided policy search (GPS)
- Policy learning from trajectory optimization with side information (PLATO)
- Natural policy gradient with demonstrations for dexterous manipulation

Acknowledgments
Today's slides are based on student presentations from 2019 by: Yuwei Chen, Jienan Yao, and Jason Rebello

Learning Neural Network Policies with Guided Policy Search under Unknown Dynamics

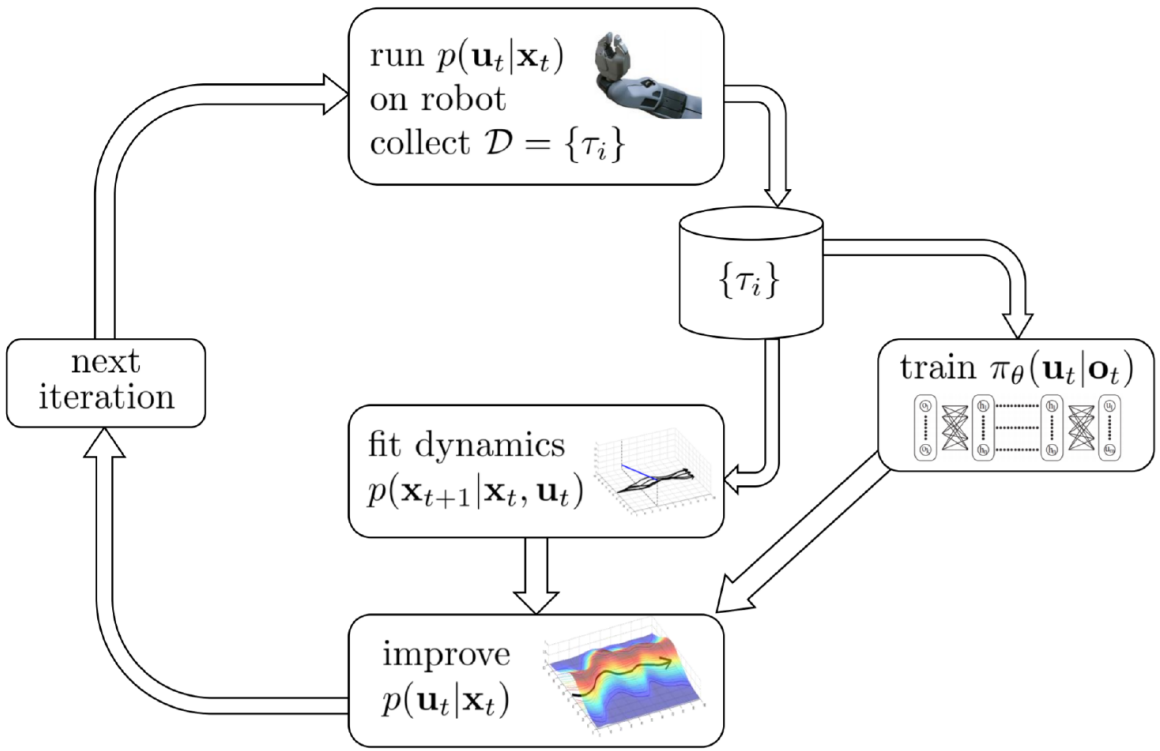
Sergey Levine and Pieter Abbeel
Presenter: Jienan Yao

March 8, 2019

Introduction

- Model-based methods
  - difficult for complex systems
- Model-free methods
  - require carefully designed, low-dimensional parameterizations
- Goal
  - learn dynamics of the system and locally valid optimal control at the same time
  - train a policy that is globally valid using learned local policies

Overview



http://rail.eecs.berkeley.edu/deeprlcourse/static/slides/lec-13.pdf

Preliminaries

Time-varying Gaussian Policy

- $p(\mathbf{u}_t|\mathbf{x}_t) = N(\mathbf{K}_t\mathbf{x}_t + \mathbf{k}_t, \mathbf{C}_t)$
- Could be efficiently optimized when the initial state distribution is narrow and approximately Gaussian

iteratively linear-Gaussian regulator (iLQG)

- Iteratively construct locally optimal linear feedback controllers
- Computed by dynamic programming under linearization of dynamics and quadratic expansion of cost
- linear-Gaussian controller

$$p(\mathbf{u}_t|\mathbf{x}_t) = N(\hat{\mathbf{u}}_t + \mathbf{k}_t + \mathbf{K}_t(\mathbf{x}_t - \hat{\mathbf{x}}_t), Q^{-1}_{\mathbf{u},\mathbf{u}t})$$

Trajectory Optimization under Unknown Dynamics

Dynamics $N(f_{\mathbf{x}t}\mathbf{x}_t + f_{\mathbf{u}t}\mathbf{u}_t, \mathbf{F}_t)$ unknown

- Estimated from the samples generated from the real system under previous linear-Gaussian controllers
- Having estimated linear-Gaussian dynamics at each time step, run the preceeding dynamic programming algorithm

Issue:

- Fitted dynamics is only valid in local region around the sample
- New controller could be very different from the old one
- Addressed by imposing KL-Divergence constraints between the old and new trajectory distribution

KL-Divergence Constraints

- Modified cost function

$$\min_{p(\tau)\in N(\tau)} E_p[l(\tau)] \text{ s.t. } D_{\mathrm{KL}}(p(\tau)||\hat{p}(\tau)) \leq \epsilon$$

- Lagrangian of this problem (η dual variable)

$$\mathcal{L}_{\mathrm{traj}}(p(\tau),\eta) = E_p[l(\tau)] + \eta[D_{\mathrm{KL}}(p(\tau)||\hat{p}(\tau)) - \epsilon]$$

- Assuming $p(\mathbf{x}_{t+1}|\mathbf{x}_t,\mathbf{u}_t) = \hat{p}(\mathbf{x}_{t+1}|\mathbf{x}_t,\mathbf{u}_t) = \mathcal{N}(f_{\mathbf{x}t}\mathbf{x}_t + f_{\mathbf{u}t}\mathbf{u}_t, \mathbf{F}_t)$

$$\mathcal{L}_{\mathrm{traj}}(p(\tau),\eta) = \left[\sum_t E_{p(\mathbf{x}_t,\mathbf{u}_t)}[l(\mathbf{x}_t,\mathbf{u}_t) - \eta \log \hat{p}(\mathbf{u}_t|\mathbf{x}_t)]\right] - \eta\mathcal{H}(p(\tau)) - \eta\epsilon$$

- Augmented cost function

$$\tilde{l}(\mathbf{x}_t,\mathbf{u}_t) = \frac{1}{\eta} l(\mathbf{x}_t,\mathbf{u}_t) - \log \hat{p}(\mathbf{u}_t|\mathbf{x}_t)$$

- Solved by dual gradient descent

Dual Gradient Descent

$$\min_{\mathbf{x}} f(\mathbf{x}) \text{ s.t. } C(\mathbf{x}) = 0$$

$$\mathcal{L}(\mathbf{x}, \lambda) = f(\mathbf{x}) + \lambda C(\mathbf{x})$$

$$g(\lambda) = \mathcal{L}(\mathbf{x}^\star(\lambda), \lambda)$$

$$\mathbf{x}^\star = \arg\min_{\mathbf{x}} \mathcal{L}(\mathbf{x}, \lambda)$$

$$\frac{dg}{d\lambda} = \frac{d\mathcal{L}}{d\lambda}(\mathbf{x}^\star, \lambda)$$

1. Find $\mathbf{x}^\star \leftarrow \arg\min_{\mathbf{x}} \mathcal{L}(\mathbf{x}, \lambda)$
2. Compute $\frac{dg}{d\lambda} = \frac{d\mathcal{L}}{d\lambda}(\mathbf{x}^\star, \lambda)$
3. $\lambda \leftarrow \lambda + \alpha \frac{dg}{d\lambda}$

http://rail.eecs.berkeley.edu/deeprlcourse/static/slides/lec-13.pdf

Background Dynamics Distribution

- Use priors to greatly reduce the number of samples required
- Gaussian Mixture Model (GMM) is a good choice for physical systems such as robots
  - dynamics reasonably approximated with piecewise linear functions
  - not necessarily good forward model but obtain prior for dynamics
- Refit the GMM at each iteration
- Infer the cluster weights for the samples
- Use the weighted mean and covariance of these clusters as the prior parameters to estimate a time-varying linear dynamics

General Parameterized Policies

Algorithm 1 Guided policy search with unknown dynamics

1: **for** iteration $k = 1$ to K **do**
2: Generate samples $\{\tau_i^j\}$ from each linear-Gaussian controller $p_i(\tau)$ by performing rollouts
3: Fit the dynamics $p_i(\mathbf{x}_{t+1}|\mathbf{x}_t, \mathbf{u}_t)$ to the samples $\{\tau_i^j\}$
4: Minimize $\sum_{i,t} \lambda_{i,t} D_{\mathrm{KL}}(p_i(\mathbf{x}_t)\pi_\theta(\mathbf{u}_t|\mathbf{x}_t) \| p_i(\mathbf{x}_t, \mathbf{u}_t))$ with respect to θ using samples $\{\tau_i^j\}$
5: Update $p_i(\mathbf{u}_t|\mathbf{x}_t)$ using the algorithm in Section 3 and the supplementary appendix
6: Increment dual variables $\lambda_{i,t}$ by $\alpha D_{\mathrm{KL}}(p_i(\mathbf{x}_t)\pi_\theta(\mathbf{u}_t|\mathbf{x}_t) \| p_i(\mathbf{x}_t, \mathbf{u}_t))$
7: **end for**
8: **return** optimized policy parameters θ

General Parameterized Policies

- Objective

$$\min_{\theta, p(\tau)} E_{p(\tau)}[l(\tau)] \text{ s.t. } D_{\mathrm{KL}}(p(\mathbf{x}_t)\pi_\theta(\mathbf{u}_t|\mathbf{x}_t)||p(\mathbf{x}_t, \mathbf{u}_t)) = 0, \forall t$$

- Lagrangian of the problem

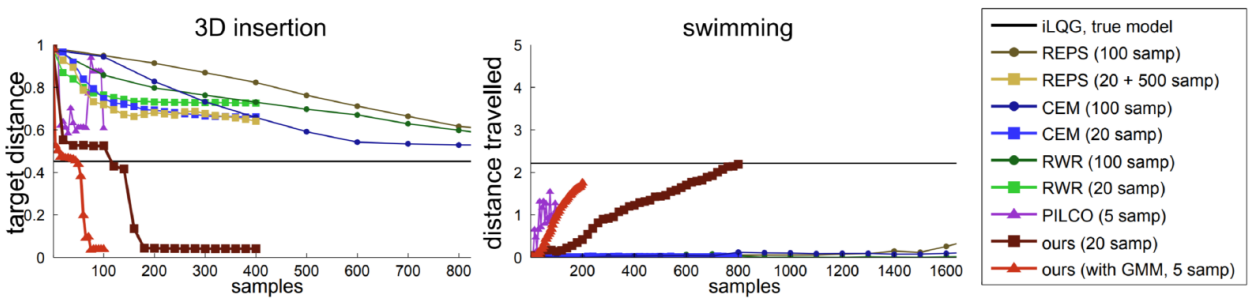
$$\mathcal{L}_{\mathrm{GPS}}(\theta, p, \lambda) = E_{p(\tau)}[l(\tau)] + \sum_{t=1}^{T} \lambda_t D_{\mathrm{KL}}(p(\mathbf{x}_t)\pi_\theta(\mathbf{u}_t|\mathbf{x}_t)||p(\mathbf{x}_t, \mathbf{u}_t))$$

- Parameterized policy trained in a supervised fashion
- Trajectory optimization exploits structure of linear-Gaussian controllers, trained with fewer samples

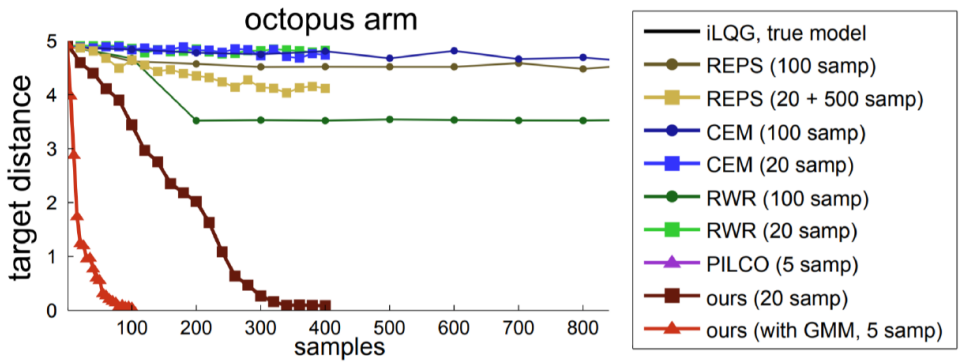
Experiments Conducted

- 2D, 3D peg insertion (discontinuous dynamics)
- Octopus arm control (high-dimensional state and action space)
- Planar swimming (three-link snake)
- Walking (seven-link biped to maintain a target velocity)

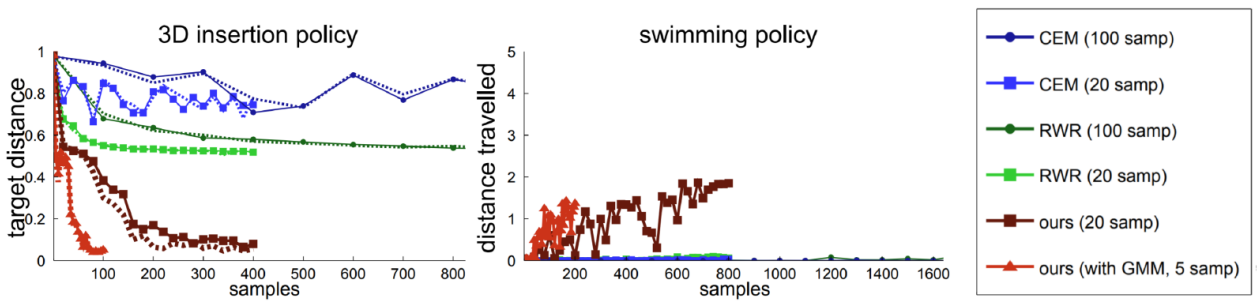
Trajectory Optimization



Trajectory Optimization



Neural Network Policy Learning with GPS



Conclusion and Discussion

- Optimize linear-Gaussian controllers under unknown dynamics
  - hybrid model-based and model-free approach
  - rely on a stronger assumption that time-varying linear-Gaussians are reasonable local approximation for the dynamics
- Train arbitrary parameterized policies (e.g. Neural Networks) within GPS framework
  - experiments show intelligent performance in partially observed environments, even for tasks that cannot be solved with direct model-free policy search
- Future directions
  - incorporate sensory information that is difficult to simulate but useful in partially observed domains

PLATO: Policy Learning using Adaptive Trajectory Optimization

Yuwei Chen

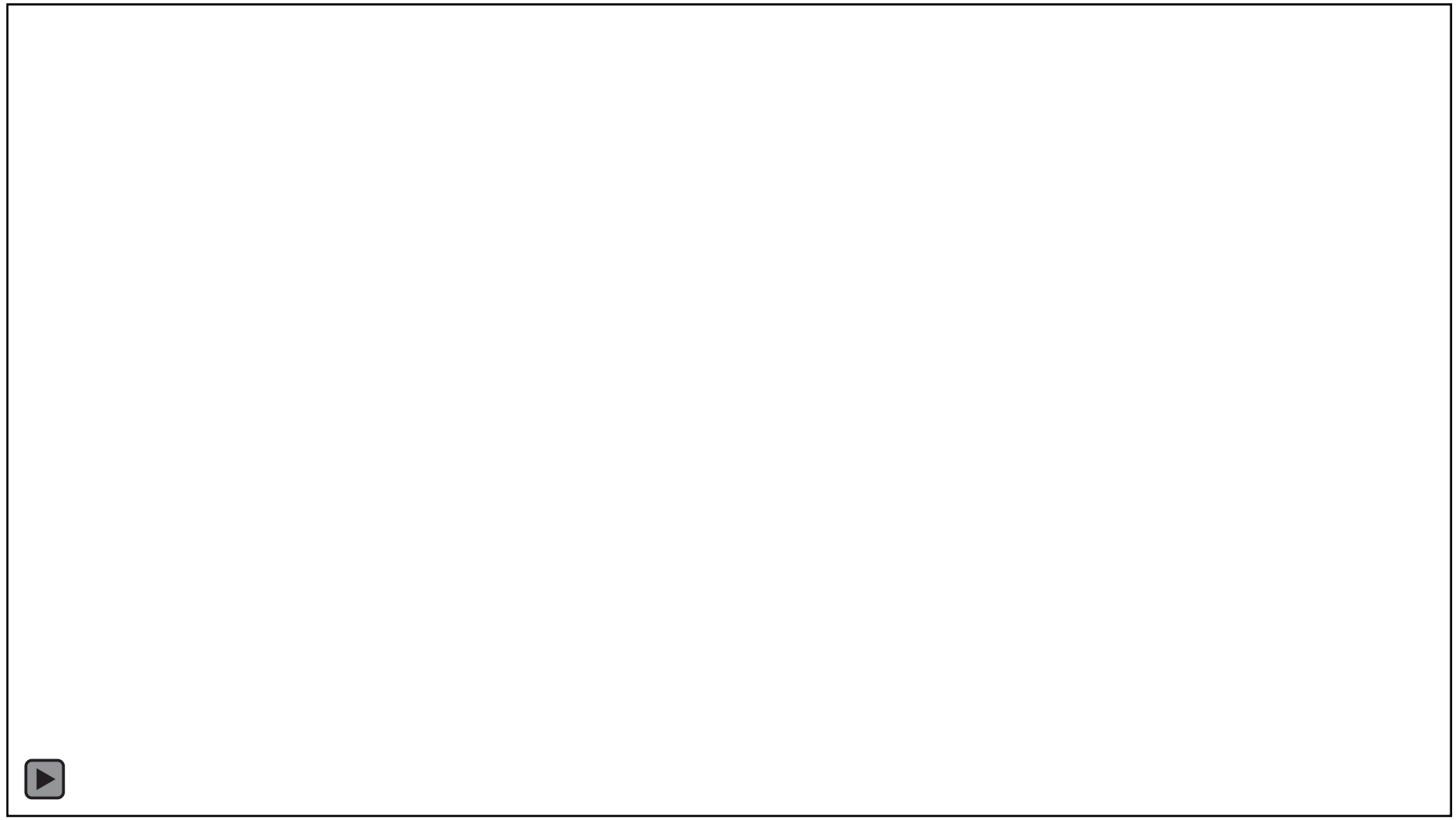
March 5th, 2019

Introduction

PLATO algorithm tries to imitate MPC teacher which could overcome following challenges:

- RL-based training of large NN needs large amount of experience (data). In contract, supervised-learning methods, such as DAgger and GPS require a viable source of supervision.
- IN RL-based training, partially trained controller will perform unreasonable and unsafe actions which can cause the destruction of robot or damage to its surroundings.

Note that MPC teacher has access to all underlying states while learner policy could only act on observations.



Problem Set-up

- states $\mathbf{x}$, actions $\mathbf{u}$.
- The policy could only control the system from observations $\mathbf{o}$.
- The policy $\pi_\theta(\mathbf{u}|\mathbf{o}_t)$, parametrized by θ.
- At test time, the agent chooses actions according to $\pi_\theta(\mathbf{u}|\mathbf{o}_t)$ at each time step t, and experiences a loss $c(\mathbf{x}_t|\mathbf{o}_t) \in [0, 1]$.
- The next state is distributed by dynamics $p(\mathbf{x}_{t+1}|\mathbf{x}_t, \mathbf{u}_t)$.

The objective is to learn policy $\pi_\theta(\mathbf{u}|\mathbf{o}_t)$ s.t.

$$\arg\min_\pi J(\pi) = \mathbf{E}_\pi[\sum_{t=1}^{T} c(\mathbf{x}_t, \mathbf{u}_t)].$$

At the same time, let's define expected cost from state $\mathbf{x}_t$ at time t as

$$J(\pi|\mathbf{x}_t) = \mathbf{E}_\pi[\sum_{t=1}^{T} c(\mathbf{x}_t, \mathbf{u}_t)|\mathbf{x}_t].$$

Adaptive MPC teacher

One naive way is to train the policy with supervised learning from data generated from MPC teacher. However, because state distribution for the teacher and leaner are different. Learned policy might fail.
In order to overcome this challenge, an adaptive MPC teacher is used which generates actions from controller obtained by:

$$\pi_\lambda^t(\mathbf{u}|\mathbf{x}_t, \theta) \leftarrow \arg\min_\pi J_t(\pi|\mathbf{x}_t) + \lambda D_{KL}(\pi(\mathbf{u}|\mathbf{x}_t)||\pi_\theta(\mathbf{u}|\mathbf{o}_t)) \qquad (1)$$

where λ determines the relative importance of matching the leaner policy versus optimizing the expected return. Note that the particular MPC algorithm is based on iLQG.

Algorithm

Algorithm 1 PLATO algorithm

Initialize data $D \leftarrow \emptyset$
for $i = 1$ to N **do**
  for $t = 1$ to T **do**
    $\pi_\lambda^t(\mathbf{u}_t|\mathbf{x}_t, \theta) \leftarrow \arg\min_\pi J_t(\pi|\mathbf{x}_t) + \lambda D_{KL}(\pi(\mathbf{u}|\mathbf{x}_t)||\pi_\theta(\mathbf{u}|\mathbf{o}_t))$.
    Sample $\mathbf{u}_t \sim \pi_\lambda^t(\mathbf{u}|\mathbf{x}_t, \theta)$.
    $\pi^*(\mathbf{u}_t|\mathbf{x}_t) \leftarrow \arg\min_\pi J(\pi)$.
    Sample $\mathbf{u}_t^* \sim \pi^*(\mathbf{u}|\mathbf{x}_t)$.
    Append $(\mathbf{o}_t, \mathbf{u}_t^*)$ to dataset D.
    State evolves $\mathbf{x}_{t+1} \sim p(\mathbf{x}_{t+1}|\mathbf{x}_t, \mathbf{u}_t)$.
  end for
  Train π_θ on D.
end for

Adaptive MPC teacher revisited

Our iLQG-based MPC algorithm produces linear-Gaussian local controller $\pi_\lambda^t(\mathbf{u}_t|\mathbf{x}_t) = N(\mu_\lambda(\mathbf{x}_t), \Sigma_t)$ where $\mu_\lambda = \mathbf{K}_t\mathbf{x}_t + \mathrm{k}_t$.
Assume our learner policy is conditionally Gaussian i.e.
$\pi_\theta(\mathbf{u}|\mathbf{o}_t) = N(\mu_\theta(\mathbf{o}_t), \Sigma_{\pi_\theta})$ where $\mu_\theta(\mathbf{o}_t)$ is the output of nonlinear function, e.g. NN. Then the MPC objective can be expressed in closed form:

$$\min_\pi J_t(\pi|\mathbf{x}_t) + \frac{1}{2}\lambda\left[\ln\left(\frac{|\Sigma_{\pi_\theta}|}{|\Sigma_t|}\right) + tr(\Sigma_{\pi_\theta}^{-1}\Sigma_t) + ||\mu_\lambda^*(\mathbf{x}_t) - \mu_\theta(\mathbf{o}_t)||^2_{\Sigma_{\pi_\theta}^{-\frac{1}{2}}} + \text{const}\right].$$

Training the learner's policy

During the supervised learning phase, we minimize the KL-divergence between the learner policy π_θ and precomputed near-optimal policies π^* which is estimated by iLQG:

$$\theta \leftarrow \arg\min_\theta \sum_{(\mathbf{x}_t, \mathbf{o}_t) \in D} D_{KL}\left(\pi_\theta(\mathbf{u}|\mathbf{o}_t) || \pi^*(\mathbf{u}|\mathbf{x}_t)\right).$$

Since both π_θ and π^* are conditionally Gaussian, the KL divergence could be expressed in closed-form if ignoring the terms not involving the learner policy means $\mu_\theta(\mathbf{o}_t)$.

$$\min_\theta \sum_{(\mathbf{x}_t, \mathbf{o}_t) \in D} ||\mu^*(\mathbf{x}_t) - \mu_\theta(\mathbf{o}_t)||^2_{\Sigma_{\pi^*}^{-\frac{1}{2}}}.$$

In this paper, μ_θ is represented by a NN, and solved by SGD.

Theoretical Analysis

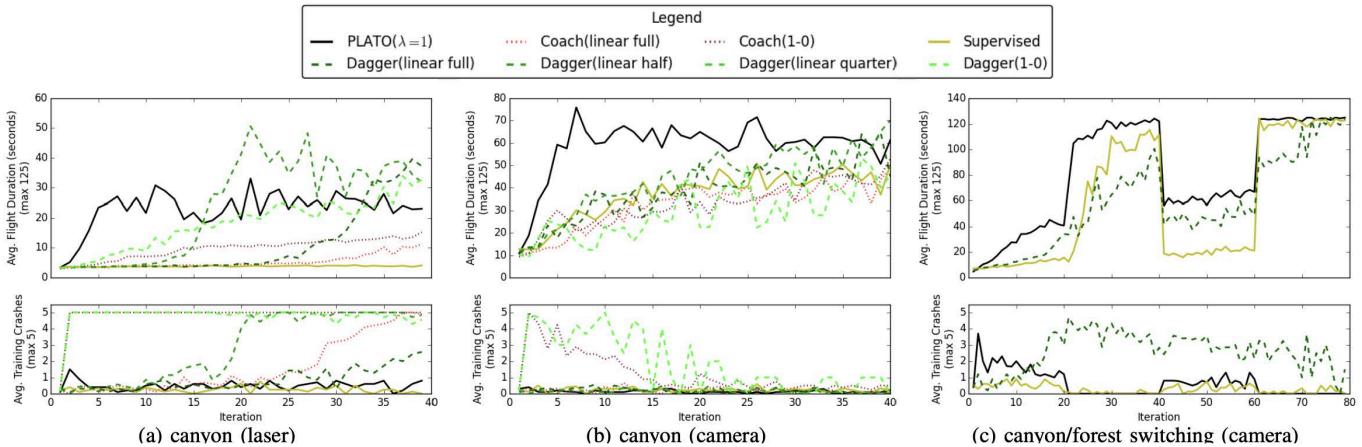
Let $Q_t(\mathbf{x}, \pi, \tilde{\pi})$ denote the cost of executing π for one time step starting from initial state, and then executing $\tilde{\pi}$ for the remaining $t-1$ time steps. We assume the cost-to-go difference between the learned policy and the optimal policy is bounded $Q_t(\mathbf{x}, \pi, \pi^*) - Q_t(\mathbf{x}, \pi^*, \pi^*) \leq \delta$

Theorem

Let the cost-to-go $Q_t(\mathbf{x}, \pi, \pi^*) - Q_t(\mathbf{x}, \pi^*, \pi^*) \leq \delta$ for all $t \in \{1, \ldots, T\}$. Then for PLATO, $J(\pi_\theta) \leq J(\pi^*) + \delta\sqrt{\epsilon_{\theta^*}} O(T) + O(1)$.

Therefore, the policy learned by PLATO converges to a policy with bounded cost.

Experiment



(a) canyon (laser) (b) canyon (camera) (c) canyon/forest switching (camera)

Comparison to DAgger

PLATO could be viewed as a generalization of DAgger, which samples from mixture policy

$$\pi_{mix,i} = \beta_i \pi^* + (1 - \beta_i)\pi_{\theta_i}.$$

Differences with the DAgger:
(1) The training data is labelled with actions from π^*.
(2) PLATO uses adaptive MPC policy to select actions at each time step, rather than the mixture policy $\pi_{mix,i}$ used.

Advantages

- The learned policy does not need to be executed during training, because of the robustness of MPC. It minimizes the catastrophic failures.
- Learned policy can use a different set of observations than MPC because the policy is directly trained on raw input from onboard sensor.

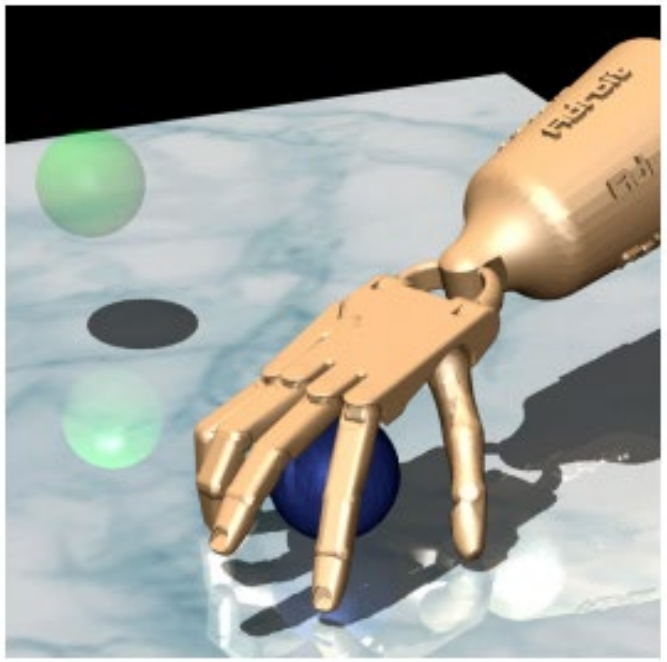
Learning Complex Dexterous Manipulation with Deep Reinforcement Learning and Demonstrations

Aravind Rajeswaran , Vikash Kumar , Abhishek Gupta, Giulia Vezzani , John Schulman , Emanuel Todorov , Sergey Levine

Jason Rebello
UTIAS

Motivation

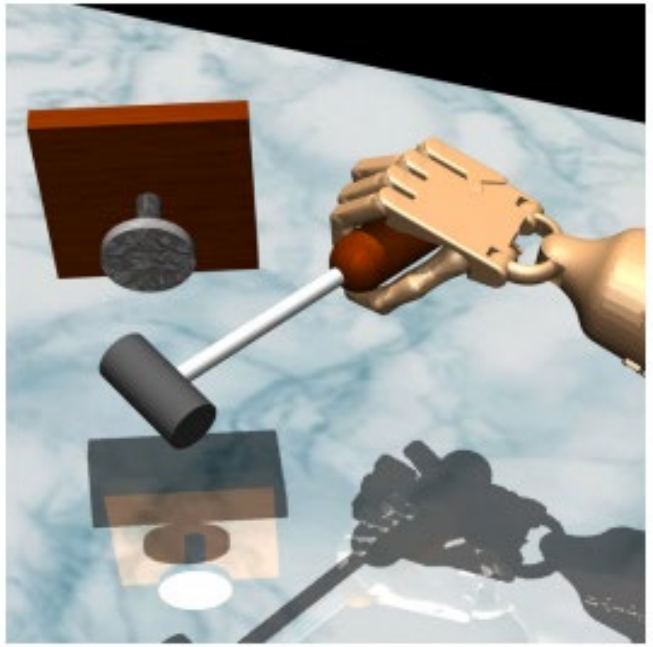
- Dexterous multi-fingered hands are ***extremely versatile***
- ***Control*** is challenging due to high dimensionality, complex contact patterns
- Previous methods require ***reward shaping***
- DRL limited to ***simpler manipulators and simple tasks***
- Lack of physical systems due to ***sample inefficiency***



Object relocation task

Contributions

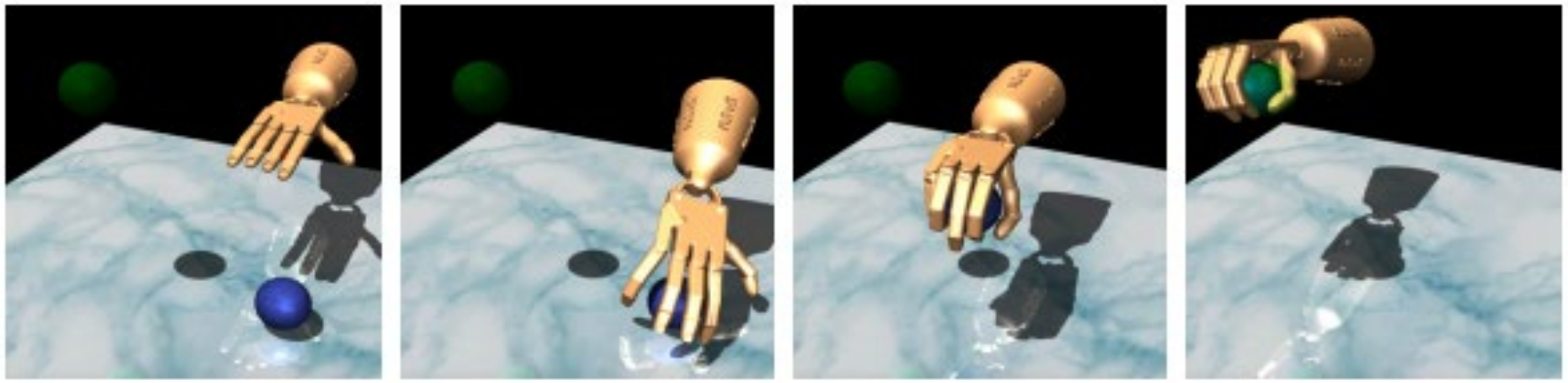
- Manipulation with ***24-DOF*** hand
- ***Model Free*** DRL
- Used in ***complex tasks with variety of tools***
- Small number of ***human demonstrations*** reduces sample complexity
- ***Reduces learning time***
- ***Robust*** and natural movements



Tool use task

Manipulation Task 1

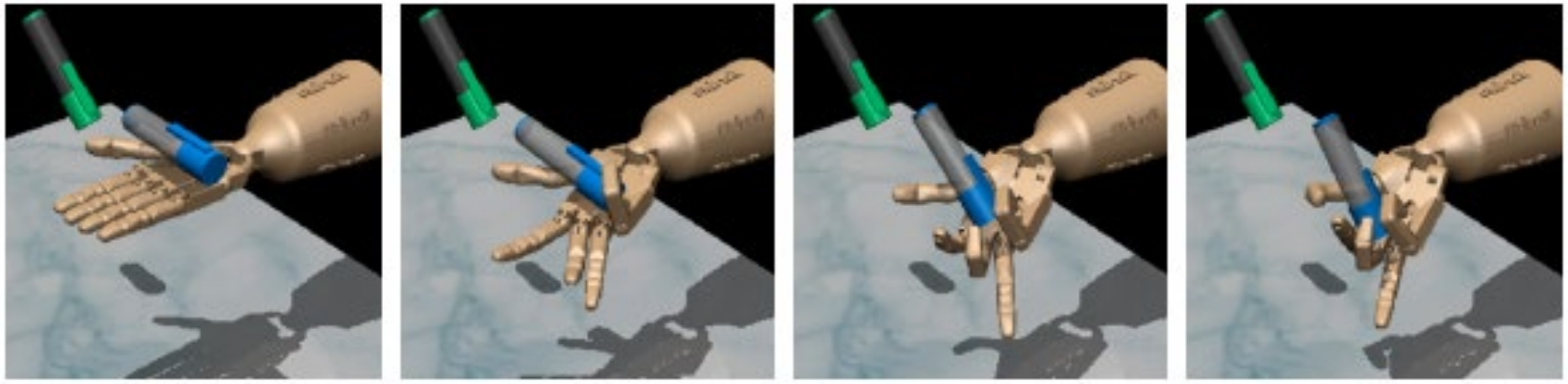
Object Relocation



- Move Blue ball to green position
- Task complete when ball is epsilon ball away from target
- Positions of ball and target are randomized
- Main challenge is exploration (reach object, grab and move to target location)

Manipulation Task 2

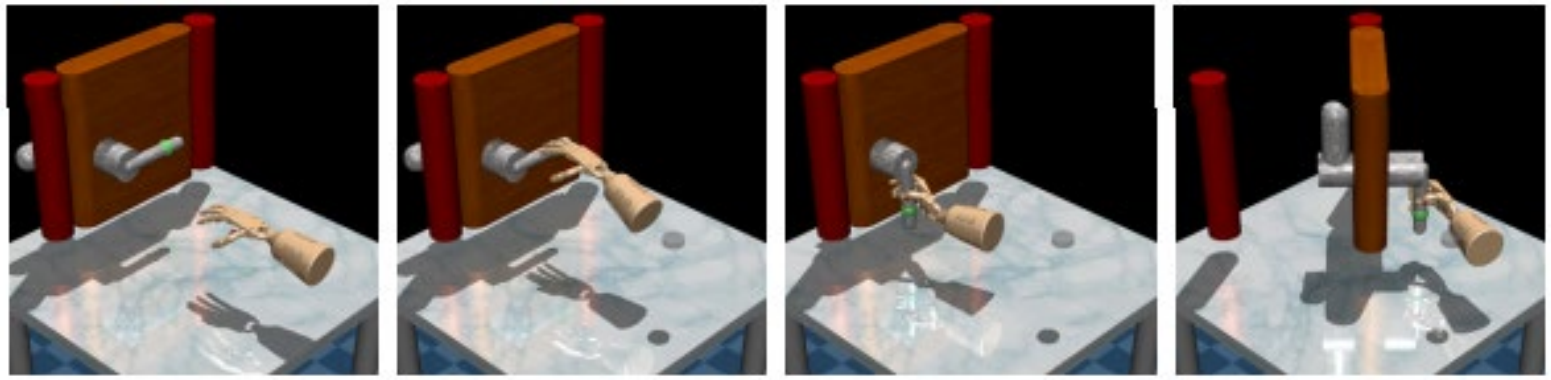
In-hand Manipulation



- Reposition blue pen to match orientation of green target
- Task complete when orientation is achieved
- Base of hand is fixed
- Large number of contacts with complex solutions
- Used a well shaped reward for training an expert

Manipulation Task 3

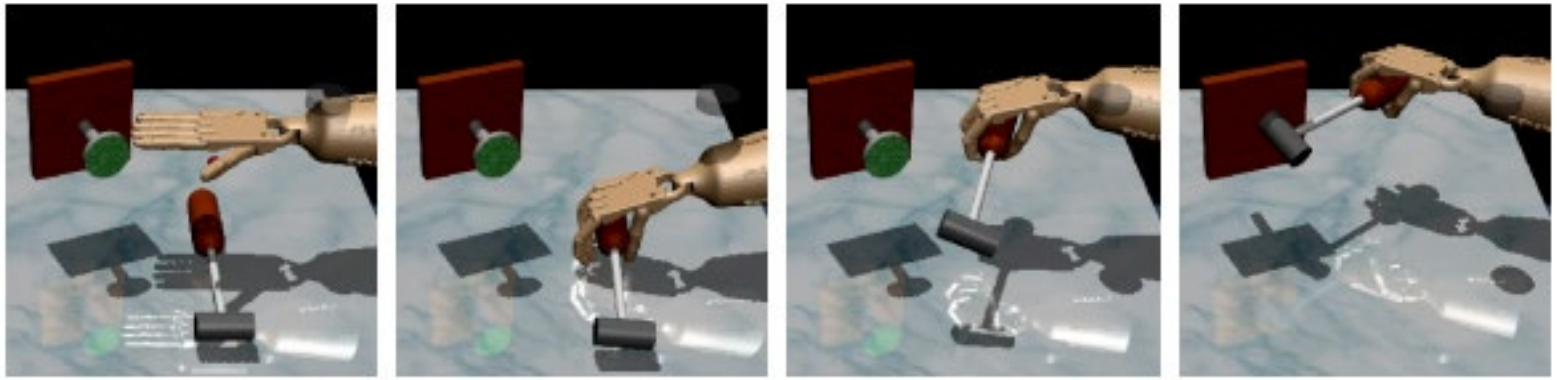
Door Opening



- Undo latch and swing door open
- Task complete when door touches door stopper
- No information of latch explicitly provided
- A lot of hidden sub-tasks
- Position of door is randomized

Manipulation Task 4

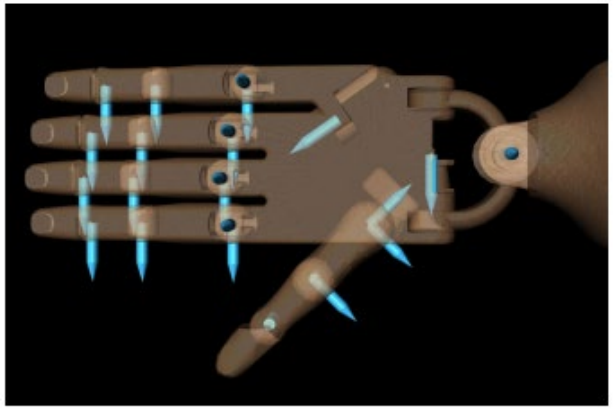
Tool Use



- Pickup and hammer nail
- Task complete when entire nail is inside the board
- Use tool instead of just relocation
- Multiple steps in task

Experimental Setup

ADROIT hand



HTC headset

HAPTIX Simulator

- 24-DOF hand
- First, middle, ring – 4 DOF each
- Little finger, thumb – 5 DOF each
- Wrist – 2 DOF
- Actuated with position control and has joint angle sensor
- MuJoCo physics simulation with friction
- 25 demonstrations for each task

CyberGlove 3

MDP definition:

$$\mathcal{M} = \{\mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{T}, \rho_0, \gamma\}$$

States — Rewards — Initial Probability distribution — Actions — Transition dynamics — Discount Factor

Value function:

$$V^{\pi}(s) = \mathbb{E}_{\pi,\mathcal{M}}\left[\sum_{t=0}^{\infty} \gamma^t r_t \mid s_0 = s\right]$$

Q function:

$$Q^{\pi}(s,a) = \underbrace{\mathbb{E}_{\mathcal{M}}\left[\mathcal{R}(s,a)\right]}_{\text{Reward for taking action a in state s}} + \underbrace{\mathbb{E}_{s' \sim \mathcal{T}(s,a)}\left[V^{\pi}(s')\right]}_{\text{Expected reward in state s'}}$$

Advantage function:

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

- Directly optimize parameters of policy to maximize objective

Sub-optimal

Vanilla Policy Gradient:

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

Fisher Information Matrix:

$$F_\theta = \frac{1}{NT} \sum_{i=1}^{N} \nabla_\theta \log \pi_\theta(a_t^i | s_t^i) \nabla_\theta \log \pi_\theta(a_t^i | s_t^i)^T$$

- Fisher information matrix measures the curvature (sensitivity) of policy relative to model parameters
- Fisher information matrix is related to the Hessian matrix

Methodology (NPG)

- Limit policy change based on parameter change
- Fisher information matrix maps between parameter space and policy space
- Generally use learning rate in optimization
- Poor step size leads to poor initialization
- Use Fisher information matrix to perform update

Gradient ascent update:

$$\theta_{k+1} = \theta_k + \sqrt{\frac{\delta}{g^T F_{\theta_k}^{-1} g}} F_{\theta_k}^{-1} g$$

Steepest Ascent direction

Normalized step-size

Methodology (Problems with RL)

- Challenges with using NPG
  - RL requires careful reward shaping
  - Impractical number of samples to learn (approx. 100 hours)
  - Unnatural movement
  - Not as robust to environmental variations
- Solution
  - Combine RL with demonstrations
  - Guide exploration and decrease sample complexity
  - Robust and natural looking behaviour
  - Demonstration Augmented Policy Gradient (DAPG)

Methodology (Pretraining with BC)

- Exploration in PG achieved with stochastic action distribution
- Poor initialization leads to slow exploration
- Behavioral Cloning (BC) guides exploration
- Reduces sample complexity

$$\underset{\theta}{\text{maximize}} \sum_{(s,a)\in\rho_D} \ln \pi_\theta(a|s)$$

- Mimic actions taken in demonstrations
- Does not guarantee effectiveness of policy due to distributional shift

Methodology (Fine-tuning with augmented loss)

- BC does not make optimal use of demonstrations
- Cannot learn subtasks (reaching, grasping, hammering)
- BC policy (only grasping)
- Capturing all data

$$g_{aug} = \sum_{(s,a)\in\rho_\pi} \nabla_\theta \ln \pi_\theta(a|s) A^\pi(s,a) + \sum_{(s,a)\in\rho_D} \nabla_\theta \ln \pi_\theta(a|s) w(s,a)$$

Policy gradient

Behavioral cloning

Dataset from policy

Dataset from demonstrations

Weighting function

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

iteration

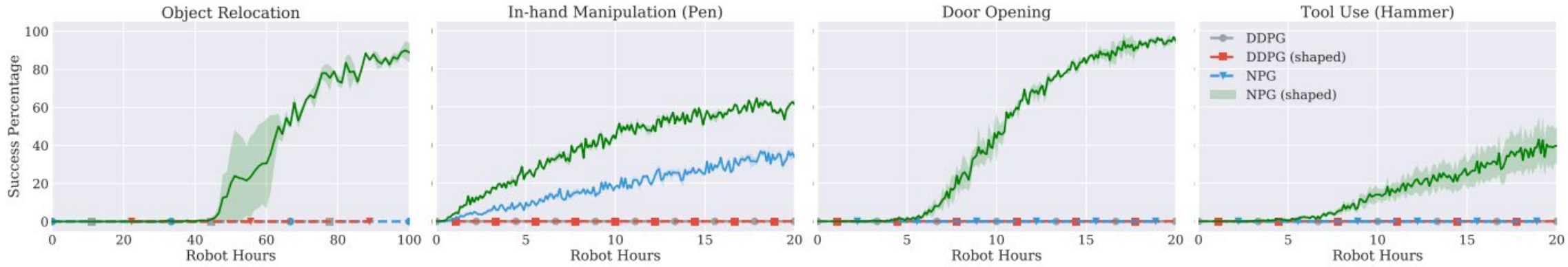
hyperparameters

Reinforcement learning from scratch

- Can RL cope with high dimensional manipulation tasks ?
- Is it robust to variations in environment ?
- Are movements safe and can they be used on real hardware ?

- Compare NPG vs DDPG (Deep Deterministic Policy Gradient)
- DDPG is a policy gradient actor-critic algorithm that is off-policy
- Stochastic policy for exploration, estimates deterministic policy
- Score based on percentage of successful trajectories (100 samples)
- Sparse Reward vs Reward shaping

Reinforcement learning from scratch



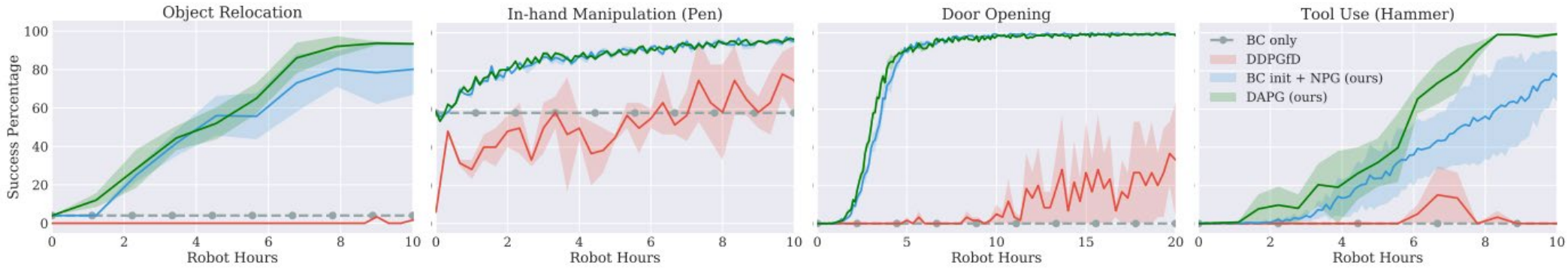
- NPG learns with reward shaping, DDPG fails to learn
- DDPG is sample efficient but sensitive to hyper-parameters
- Resulting policies have unnatural behaviors
- Poor sample efficiency, cant use on hardware
- Cannot generalize to unseen environment (weight and ball size change)

Reinforcement learning with demonstrations

- Does incorporating demonstrations reduce learning time?
- Comparison of DAPG vs DDPGfD (.. from Demonstrations)?
- Does it result in human like behaviour ?

- DDPGfD better version of DDPG (demonstrations in replay buffer, prioritized experience replay, n-step returns, regularization)
- Only use sparse rewards

Reinforcement learning with demonstrations



- DAPG outperforms DDPGfD
- DAPG requires few robot hours
- Can be used on real hardware
- Robust and human behavior
- Generalizes to unseen environment

RL iterations to achieve 90% success

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

NPG

Future Work

- Tests on real hardware
- Reduce sample complexity using novelty based exploration methods
- Learn policies from raw visual inputs and tactile sensing

Results