ROB-GY 6323
reinforcement learning and optimal control for robotics

Lecture 12
Imitation learning

Course material

All necessary material will be posted on Brightspace

Code will be posted on the Github site of the class

https://github.com/righetti/optlearningcontrol

Discussions/Forum with Slack

Contact

ludovic.righetti@nyu.edu

Office hours in person

Wednesday 3pm to 4pm

370 Jay street - room 801

Course Assistant

Armand Jordana

aj2988@nyu.edu

Office hours Monday 1pm to 2pm

Rogers Hall 515

any other time by appointment only

Tentative schedule (subject to change)

Week	Lecture		Homework	Project
1	Intro	Lecture 1: introduction		
2	Trajectory optimization	Lecture 2: Basics of optimization	HW 1	
3		Lecture 3: QPs		
4		Lecture 4: Nonlinear optimal control		
5		Lecture 5: Model-predictive control		
6		Lecture 6: Sampling-based optimal control	HW 2	
7	Policy optimization	Lecture 7: Bellman's principle		
8		Lecture 8: Value iteration / policy iteration		Project 1
9		Lecture 9: Q-learning	HW 3	
10		Lecture 10: Deep Q learning		
11		Lecture 11: Actor-critic algorithms		
12		Lecture 12: Learning by demonstration	HW 4	Project 2
13		Lecture 13: Monte-Carlo Tree Search		
14		Lecture 14: Beyond the class		
15	Finals week			

Project 1 is due Nov 22nd

Question Can we directly compute the policy without knowing the Q- or value functions?

Answer Yes! for example using policy gradients

Policy gradient methods

Assume that we have a parametrized policy $u = \pi(x, \theta)$

Can we find a relation between the policy parameters θ
and the associated performance? e.g. find $J(\theta) = V_\pi(x_0)$?

Can we find the gradient $\frac{\partial}{\partial \theta} J(\theta) = \nabla J(\theta)$?

With the gradient, we can improve the policy with gradient descent

$$\theta \leftarrow \theta - \gamma \nabla J(\theta)$$

Stochastic policies

We will derive the policy gradient for stochastic policies

Let's assume a stochastic policy $\pi(u|x,\theta) = \Pr\{u_t = u | x_t = x, \theta\}$

Policy gradient theorem

Let's define $J(\theta) = \mathbb{E}_{u_n \sim \pi_\theta}\left[\sum_{n=0}^{N} \alpha^n g(x_n, u_n)\right]$

$V_\pi^n(x_n) = \mathbb{E}_{u_n \sim \pi}\left[\sum_{k=n}^{N} \alpha^k g(x_k, u_k)\right]$ is the cost-to-go of policy π at stage n

$Q_\pi^n(x, u) = g(x, u) + \alpha V_\pi^{n+1}(x')$ is the state-action value function of policy π

The policy gradient theorem states that

$$\nabla_\theta J(\theta) = \mathbb{E}_{x,u \sim \pi}\left(\sum_{n=0}^{N} \alpha^n Q_\pi^n(x_n, u_n) \nabla_\theta \log \pi_\theta(u_n | x_n)\right)$$

REINFORCE (Monte-Carlo PG) [Williams, 1992]

Initialize the policy parameters θ for an input policy $\pi(u|x, \theta)$

Choose a step size γ (using discount factor α)

Loop forever (for each episode):

Generate an episode $x_0, u_0, x_1, u_1, \cdots, x_N, u_N$ following π

For each step t of the episode

$$G_t = \sum_{k=t}^{T} \alpha^k g(x_k, u_k)$$

$$\theta \leftarrow \theta - \gamma G_t \nabla_\theta \left[\ln \pi(u_t | x_t, \theta)\right]$$

REINFORCE with baseline

[Williams, 1992]

Replace $$\theta \leftarrow \theta - \gamma G_t \nabla_\theta \left[\ln \pi(u_t|x_t,\theta)\right]$$

with $$\theta \leftarrow \theta - \gamma \Big(G_t - b(x)\Big) \cdot \nabla_\theta \left[\ln \pi(u_t|x_t,\theta)\right]$$

where for example b(x) is an approximation of the value function
(this can help normalize the gradient step)

REINFORCE with baseline

[Williams, 1992]

Initialize parameters θ_V for value function $V(x, \theta_V)$

Initialize parameters θ_π for policy function $\pi(u|x, \theta_\pi)$

Choose step sizes $\gamma_\pi > 0$ and $\gamma_V > 0$

Loop forever (for each episode):

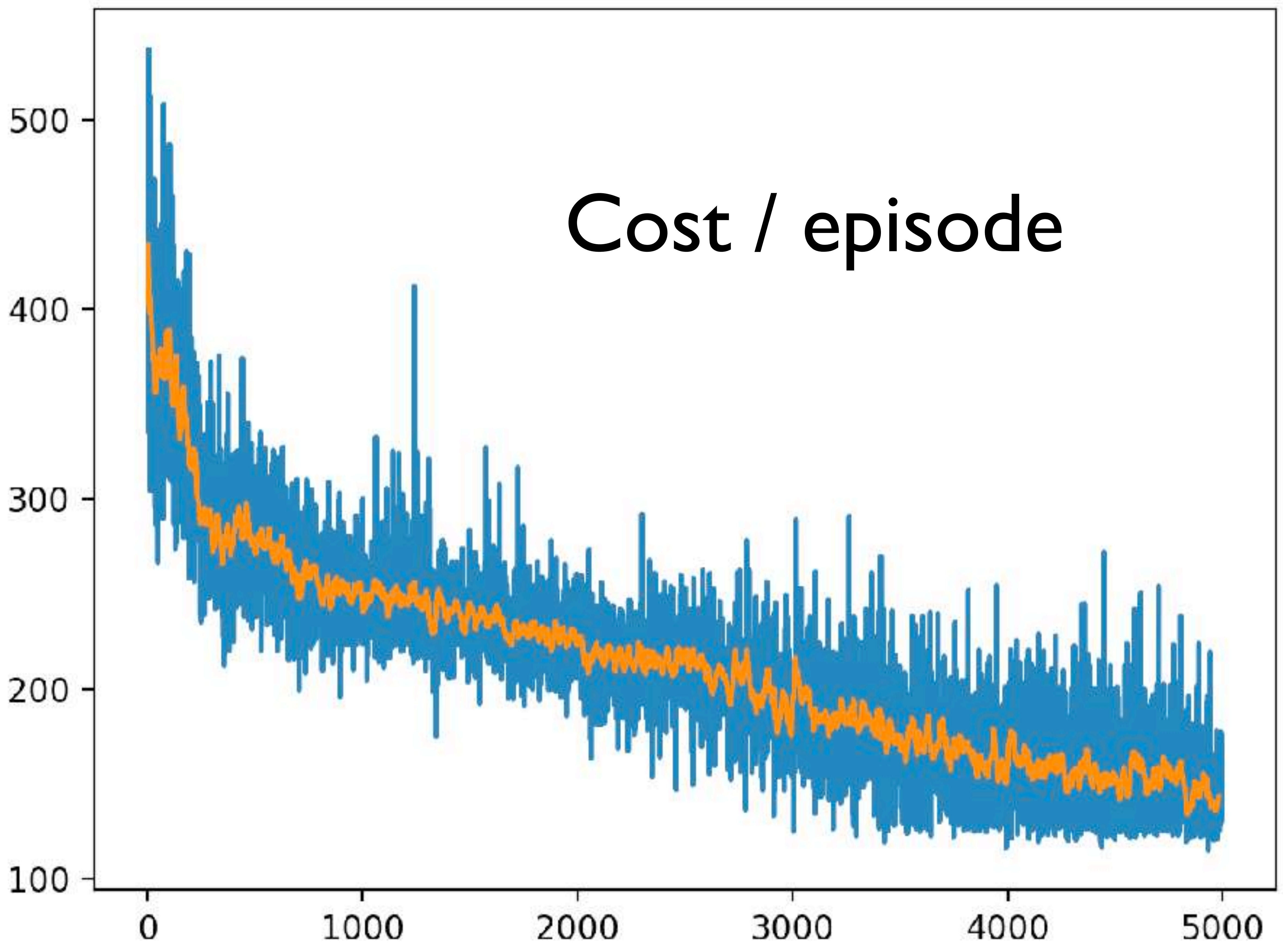
Generate an episode $x_0, u_0, x_1, u_1, \cdots, x_N, u_N$ following π

For each step t of the episode

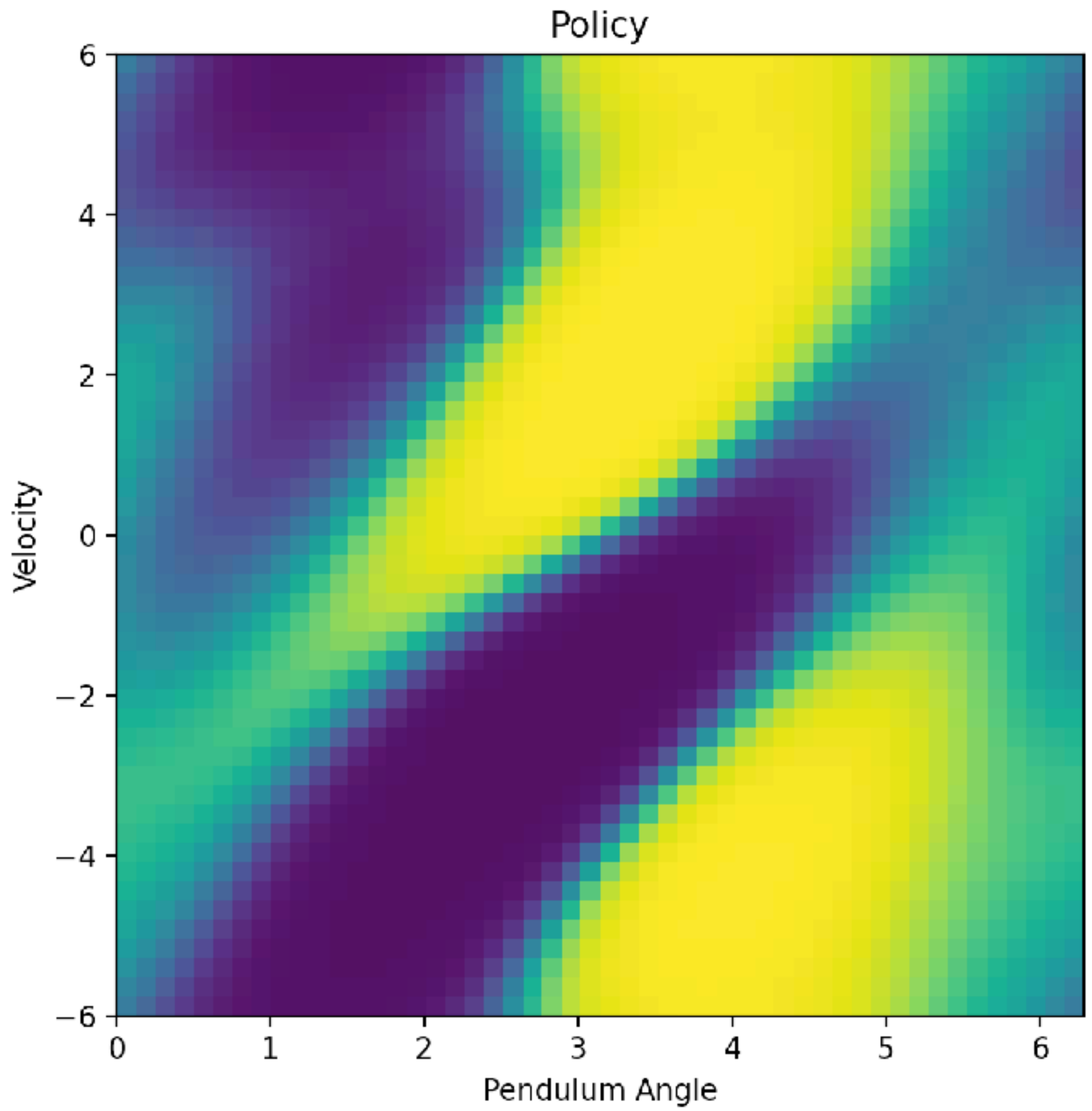
$$G_t = \sum_{k=t}^{T} \alpha^k g(x_k, u_k)$$

$$\theta_V \leftarrow \theta_V - \gamma_V \Big(V(x_t) - G_t \Big) \cdot \nabla_{\theta_V} V(x_t, \theta_V)$$

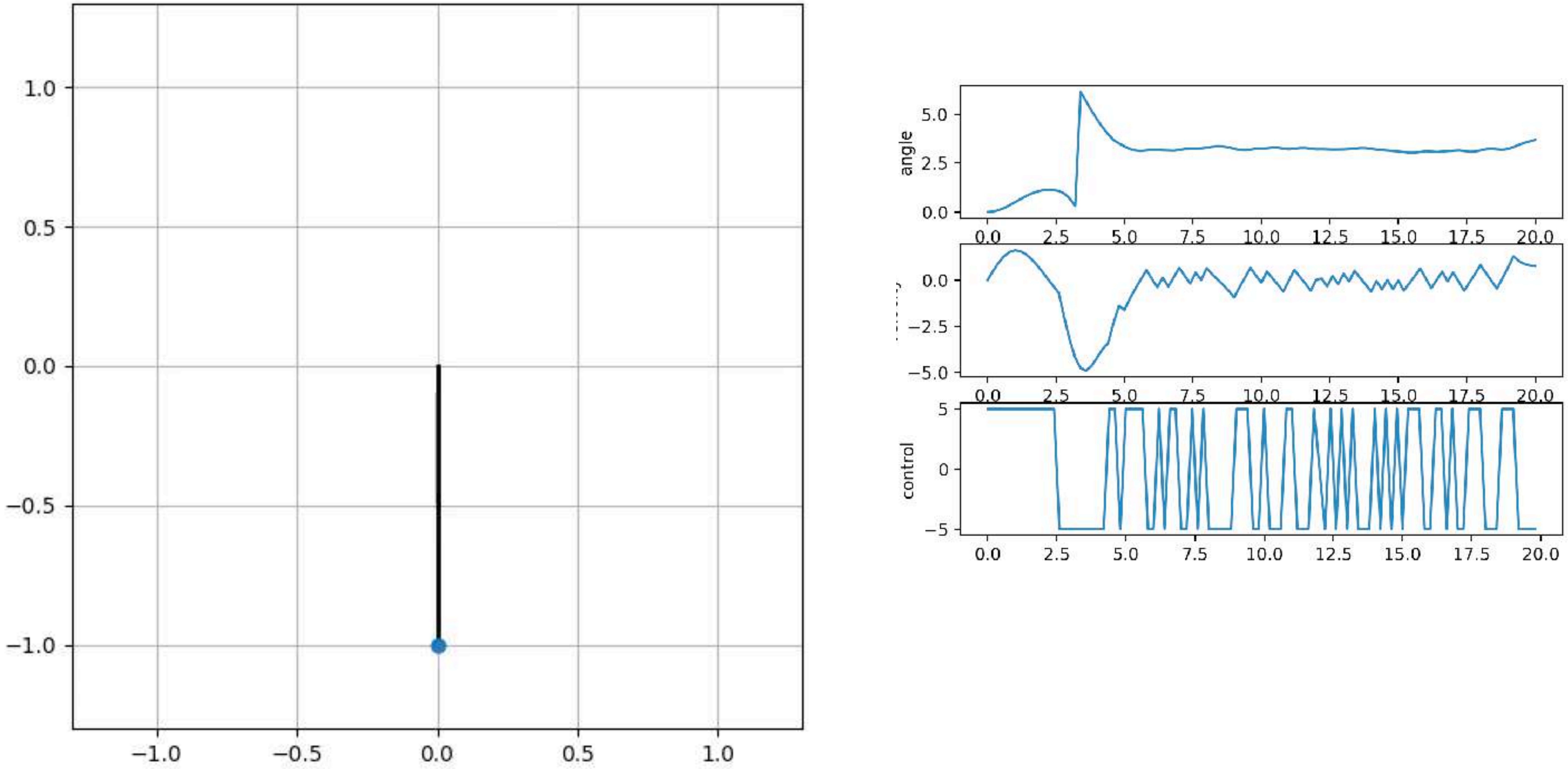
$$\theta_\pi \leftarrow \theta_\pi - \gamma_\pi \Big(G_t - V(x_t) \Big) \cdot \nabla_{\theta_\pi} \left[\ln \pi(u_t | x_t, \theta_\pi) \right]$$



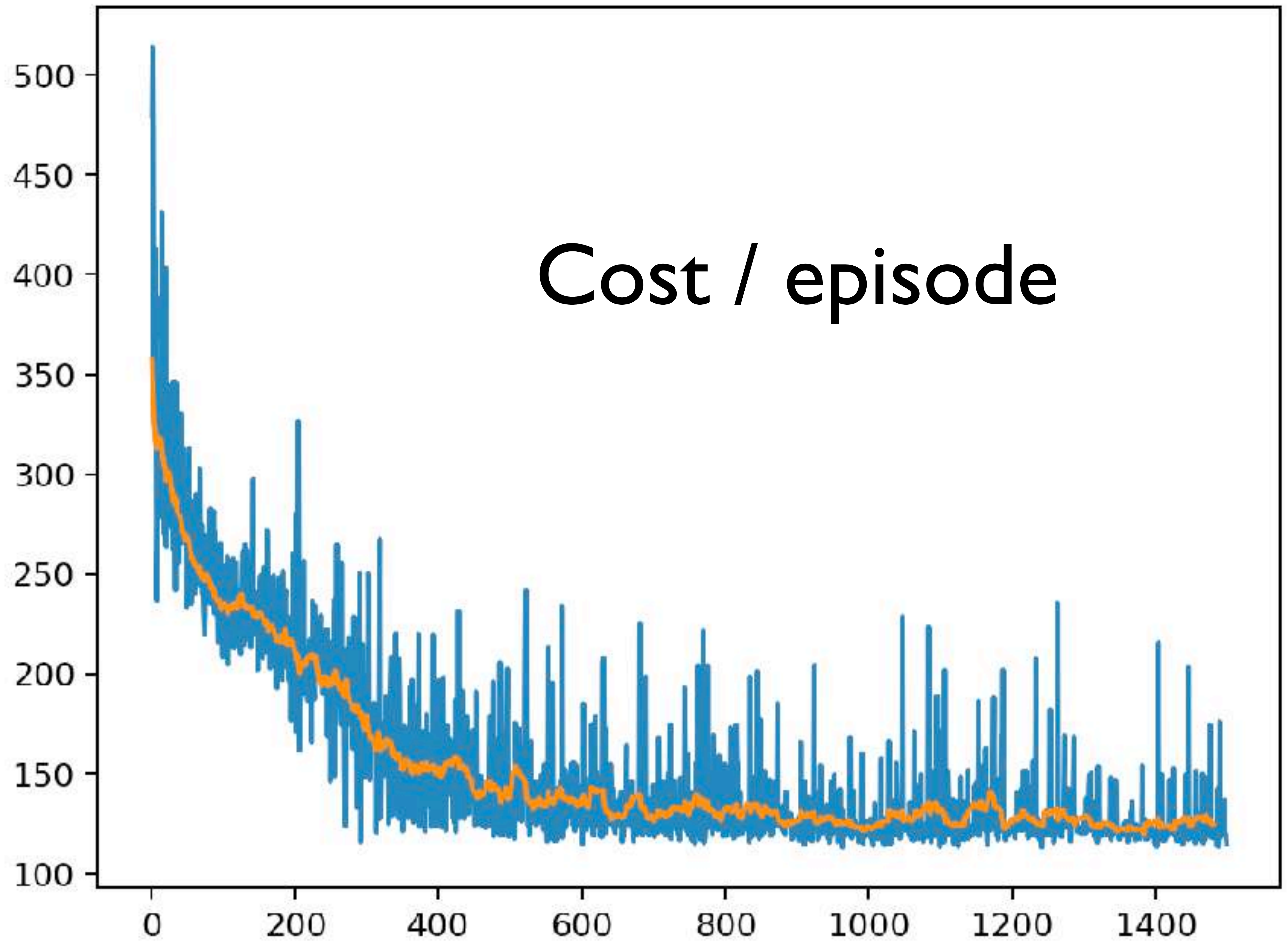
Cost / episode
500
400
300
200
100
0
1000
2000
3000
4000
5000

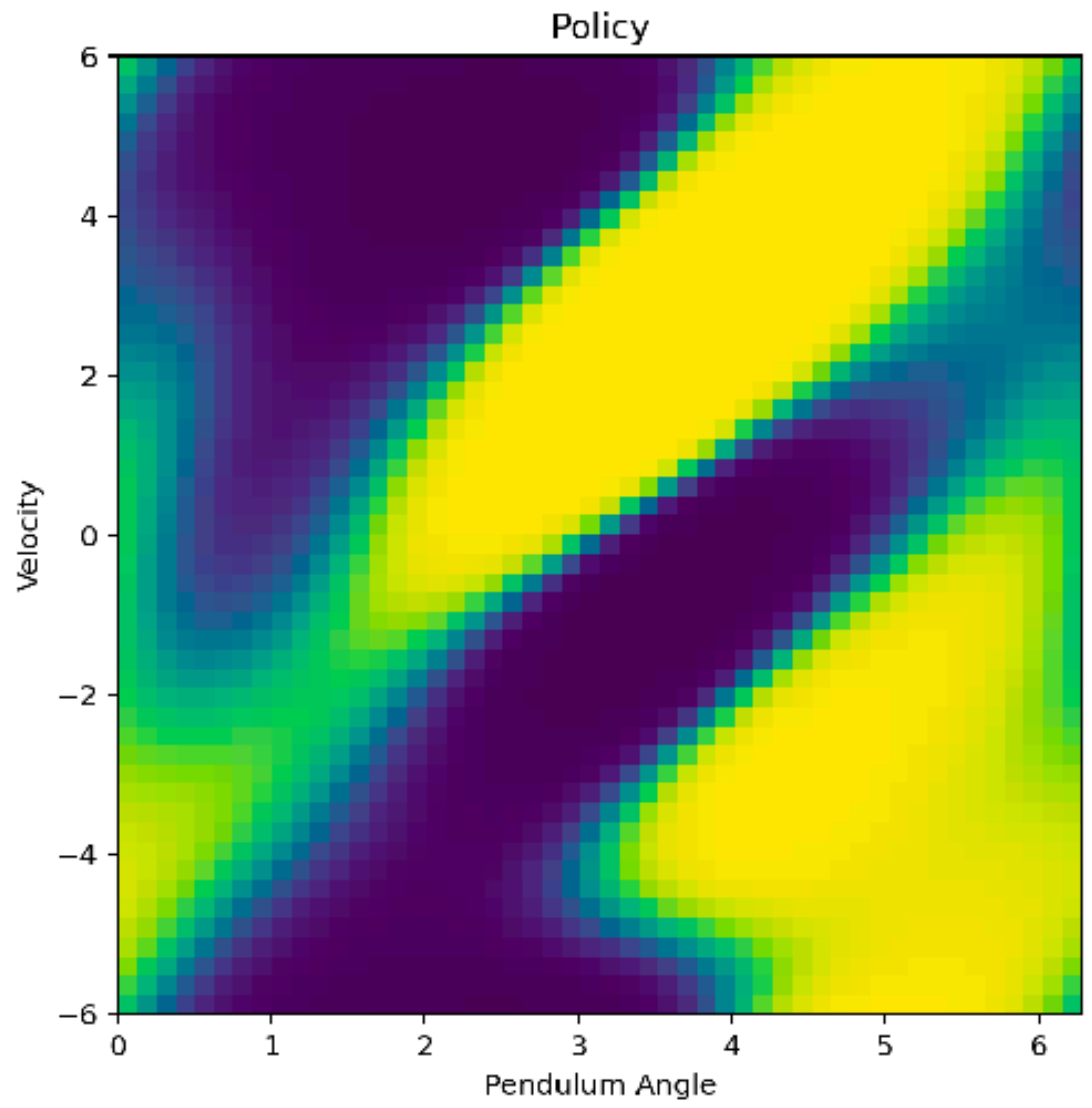


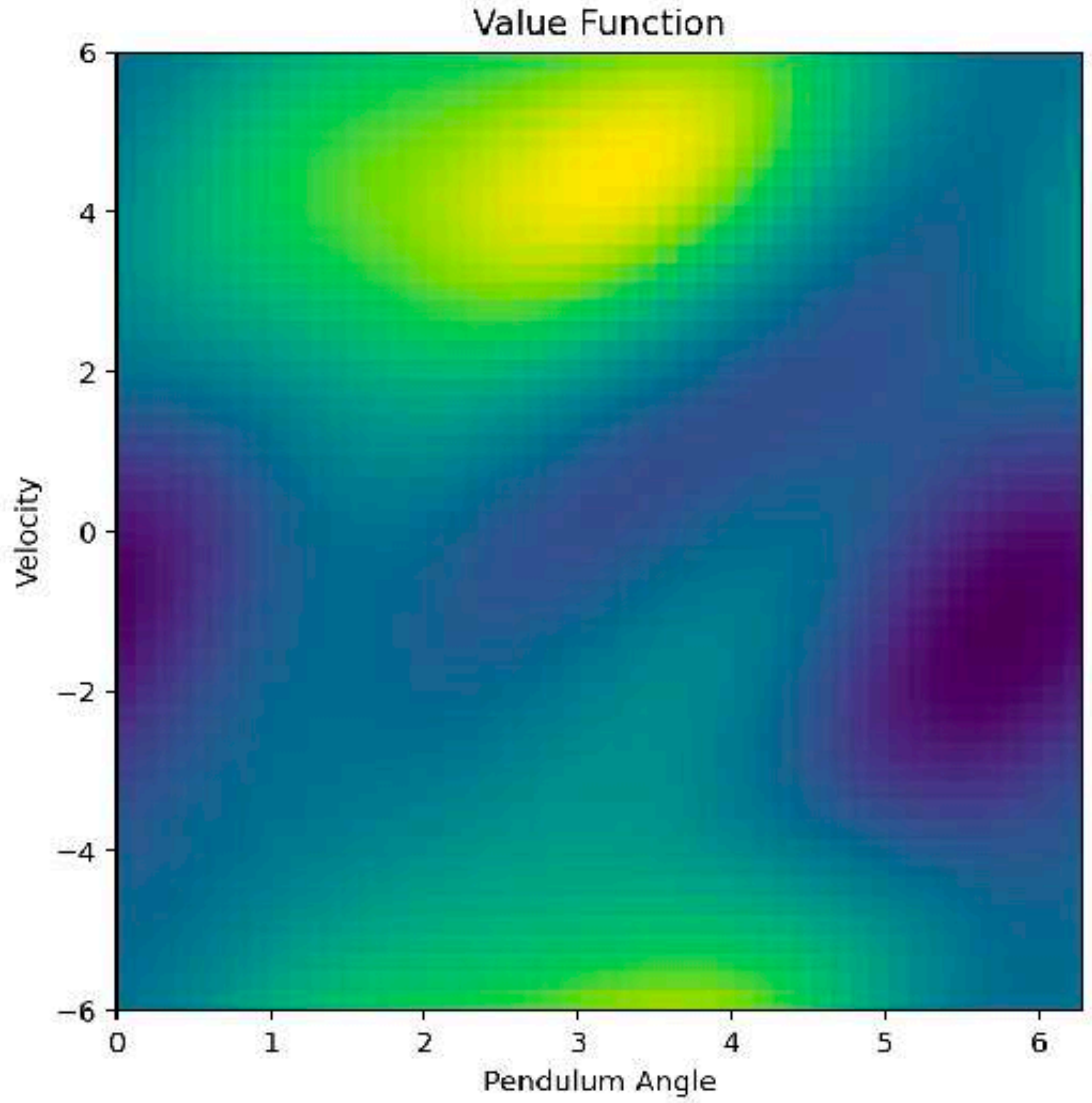
Policy
Velocity
Pendulum Angle
6
4
2
0
−2
−4
−6
0
1
2
3
4
5
6

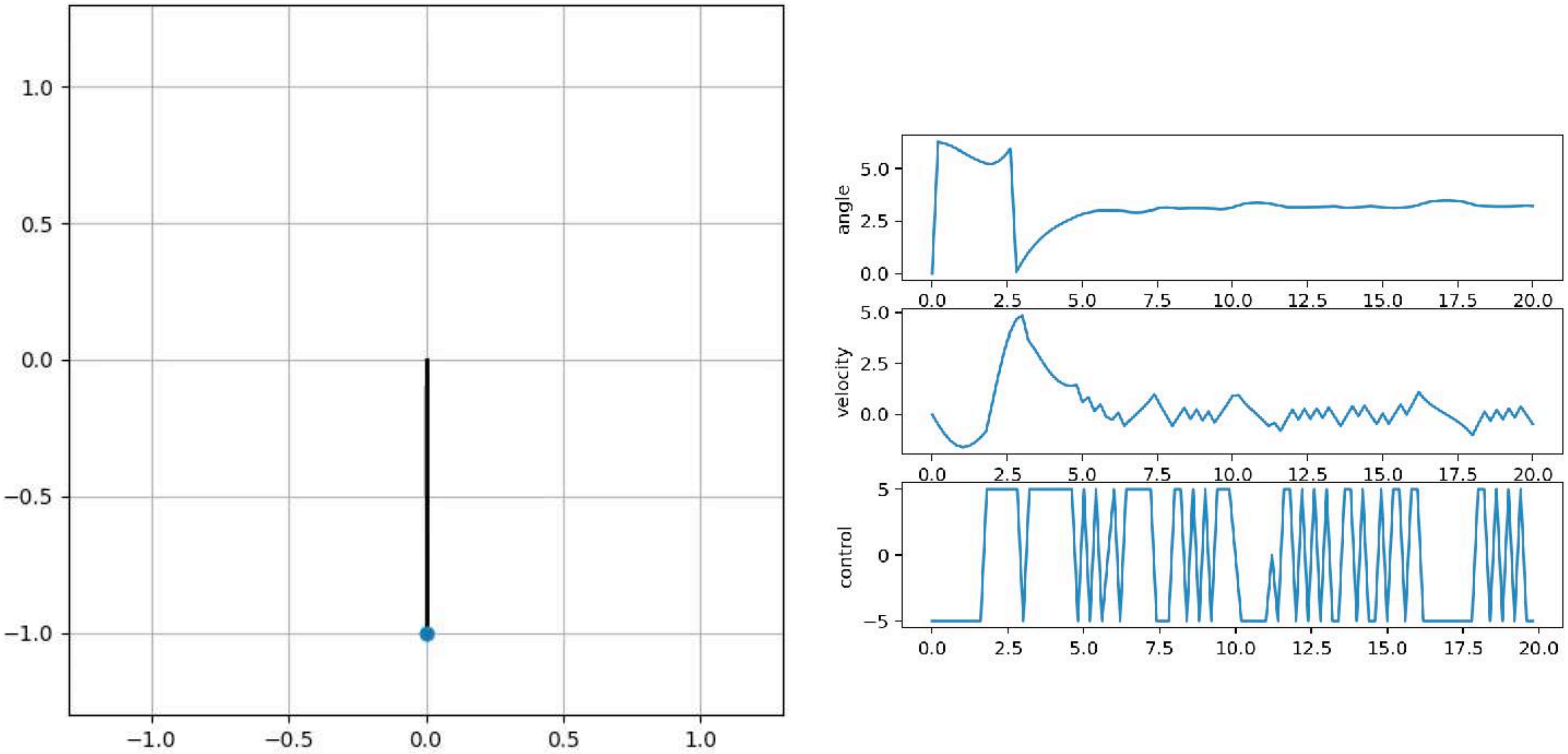

angle
control

REINFORCE with baseline








1.0
0.5
0.0
−0.5
−1.0
−1.0
−0.5
0.0
0.5
1.0
angle
5.0
2.5
0.0
velocity
5.0
2.5
0.0
control
5
0
−5
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0

Actor-critic methods

We can use the TD error directly instead of computing the return on the full episode

Policy gradient methods

REINFORCE $\nabla_\theta J(\theta) = \mathbb{E}\left[\sum_{n=0}^{N} G_n \nabla_\theta \log \pi(u_n|x_n,\theta)\right]$ $G_n = \sum_{k=n}^{N} \alpha^k g(x_k, u_k)$

REINFORCE with baseline $\nabla_\theta J(\theta) = \mathbb{E}\left[\sum_{n=0}^{N} (G_n - V(x_n)) \nabla_\theta \log \pi(u_n|x_n,\theta)\right]$

Actor-critic

$$\nabla_\theta J(\theta) = \mathbb{E}\left[\sum_{n=0}^{N} (g(x_n, u_n) + \alpha V(x_{n+1}) - V(x_n)) \nabla_\theta \log \pi(u_n|x_n,\theta)\right]$$

Policy gradient methods

$$\nabla_\theta J(\theta) = \mathbb{E}\left[\sum_{n=0}^{N} \Psi_n \nabla_\theta \log \pi(u_n|x_n,\theta)\right]$$

$$\Psi_n = \sum_{k=0}^{N} \alpha^k g(x_k,u_k)$$

$$\Psi_n = \sum_{k=n}^{N} \alpha^k g(x_k,u_k)$$

$$\Psi_n = \sum_{k=n}^{N} \alpha^k g(x_k,u_k) - b(x_n)$$

$$\Psi_n = g(x_n,u_n) + \alpha V(x_{n+1}) - V(x_n)$$

$$\Psi_n = Q_\pi(x_n,u_n)$$

$$\Psi_n = A_n = Q(x_n,u_n) - V(x_n)$$

Proximal policy optimization (PPO)

Explicit gradient descent $$\nabla_\theta J(\theta) = \mathbb{E}\left[\sum_{n=0}^{N} \Psi_n \nabla_\theta \log \pi(u_n|x_n,\theta)\right]$$

Equivalent to $$\min_\theta \mathbb{E}\left[\Psi_n \log \pi(u_n|x_n,\theta)\right]$$

Use the gradient of log to re-arrange the formula $$\min_\theta \mathbb{E}\left[A_n \frac{\pi(u_n|x_n,\theta)}{\pi(u_n|x_n,\theta_{old})}\right]$$

Proximal policy optimization (PPO)

"Clip" the total scaling

$$\min_{\theta} \mathbb{E}\left[\min\left(A_n \frac{\pi(u_n|x_n,\theta)}{\pi(u_n|x_n,\theta_{old})}, clip\left(\frac{\pi(u_n|x_n,\theta)}{\pi(u_n|x_n,\theta_{old})}, 1-\epsilon, 1+\epsilon\right) A_n\right)\right]$$

Run a lot of episodes in <u>parallel</u> (in simulation) to
improve the estimation of the gradient and expectation

Proximal policy optimization (PPO)

While not converged

For actors 1, … , P do

Run the policy in the simulator for N time steps

Collect state/action transition

Compute advantage estimates $A_n = \sum_{k=n}^{N} (\alpha\lambda)^{k-n} \delta_k$

End for

Do gradient descent on the cost

$$\min_{\theta} \mathbb{E}\left[\min\left(A_n \frac{\pi(u_n|x_n,\theta)}{\pi(u_n|x_n,\theta_{old})}, clip\left(\frac{\pi(u_n|x_n,\theta)}{\pi(u_n|x_n,\theta_{old})}, 1-\epsilon, 1+\epsilon\right) A_n\right)\right]$$

Update the value function estimates (e.g. TD-learning)

Proximal policy optimization (PPO)

Lots of heuristics but it works rather well in practice
Parallelization and clipping help a lot to get good gradient steps

PPO is considered "state of the art" for deep RL in robotics

BUT it is rarely used as is - a lot of engineering around is necessary

Getting started with RL… CleanRL

CleanRL Search vwxyzjn/cleanrl v1.0.0 5.7k 642

CleanRL - Overview

license MIT | tests passing | docs success | discord 44 online | Views 13k | code style black | imports isort | 🤗 Models Huggingface | Open in Colab

CleanRL is a Deep Reinforcement Learning library that provides high-quality single-file implementation with research-friendly features. The implementation is clean and simple, yet we can scale it to run thousands of experiments using AWS Batch. The highlight features of CleanRL are:

- 📜 Single-file implementation
- *Every detail about an algorithm variant is put into a single standalone file.*
- For example, our `ppo_atari.py` only has 340 lines of code but contains all implementation details on how PPO works with Atari games, **so it is a great reference implementation to read for folks who do not wish to read an entire modular library**.
- 📊 Benchmarked Implementation (7+ algorithms and 34+ games at https://benchmark.cleanrl.dev)
- 📈 Tensorboard Logging
- 🪛 Local Reproducibility via Seeding
- 🎮 Videos of Gameplay Capturing
- 🧫 Experiment Management with Weights and Biases
- 💸 Cloud Integration with docker and AWS

You can read more about CleanRL in our technical paper and documentation.

CleanRL only contains implementations of **online** deep reinforcement learning algorithms. If you are looking for **offline** algorithms, please check out corl-team/CORL, which shares a similar design philosophy as CleanRL.

[Chane-Sane et al IROS 2024]

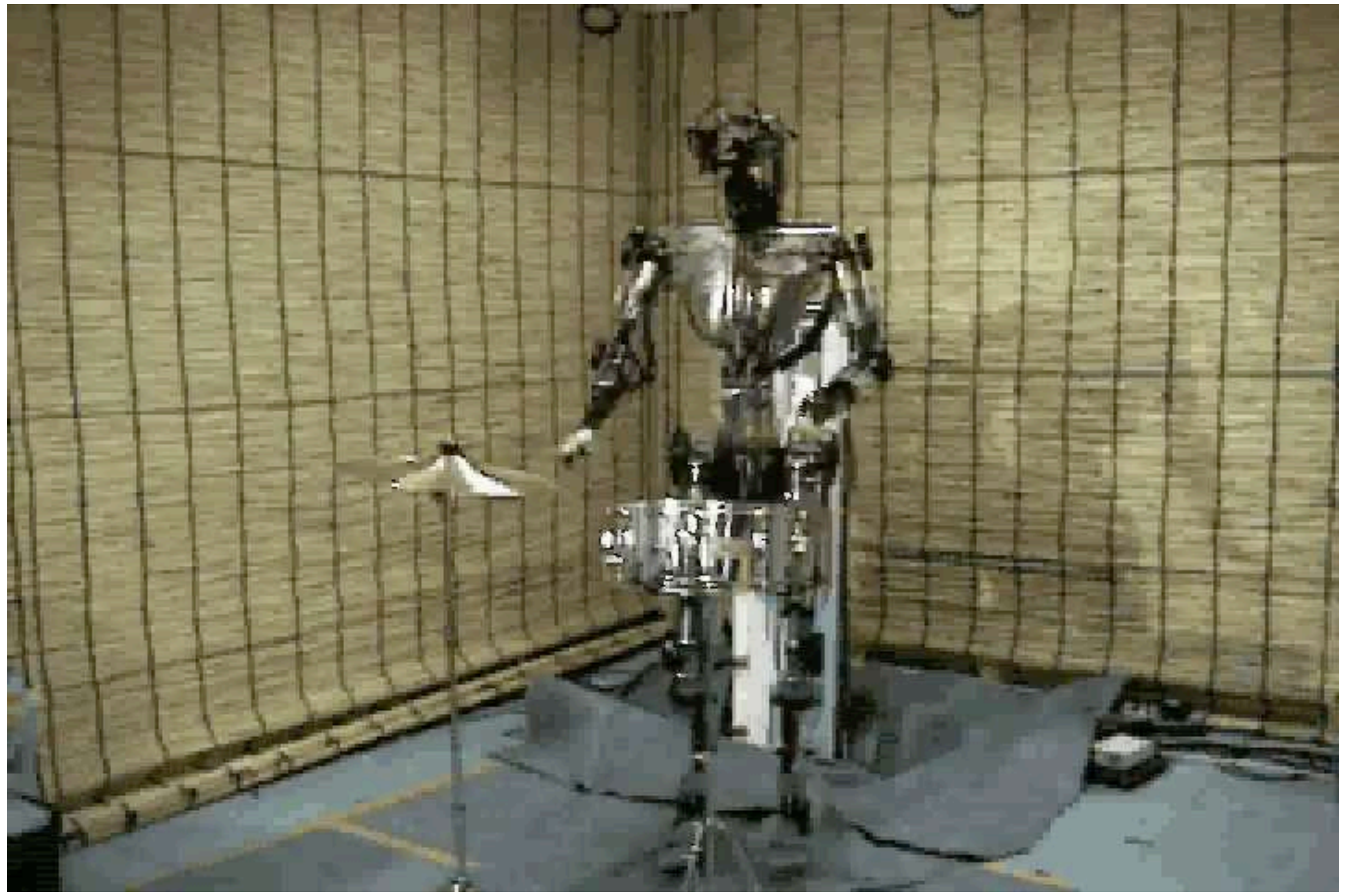
Imitation learning

Learning optimal policies from demonstration

Learning trajectories from demonstrations

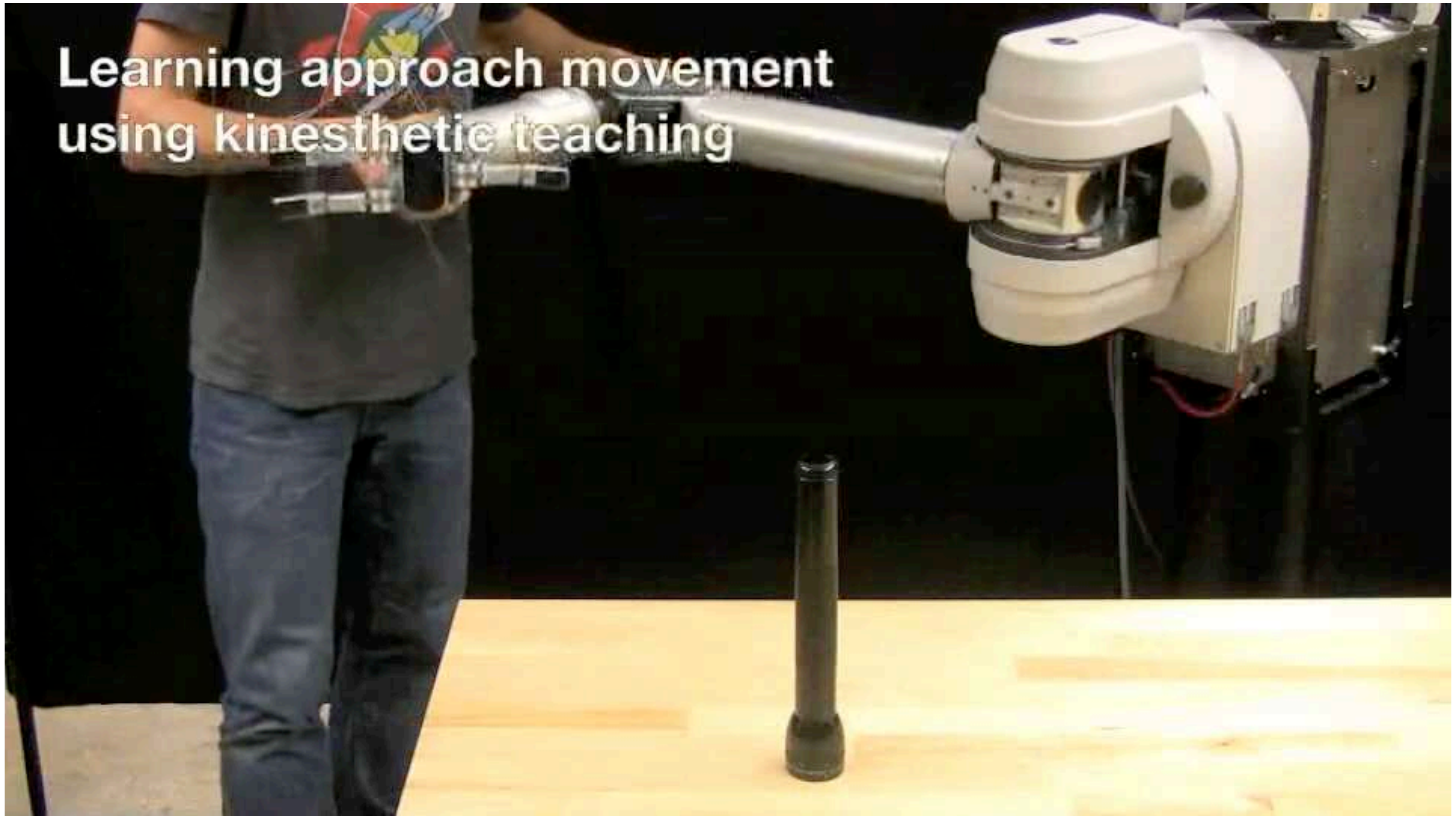
Idea: show the robot what to do, record the movement and replay it

Find a way to a controller around the trajectory to recover from perturbations



[Ijspeert et al. 2002]

Learning trajectories from demonstrations (kinesthetic teaching)



[Pastor et al. 2011]

Behavioral cloning: learning <u>policies</u> from demonstrations

Provide a lot of demonstrations and learn a policy from it

Input: a dataset of demonstrations $(x_0, u_0, x_1, u_1, \cdots, x_N, u_N)$

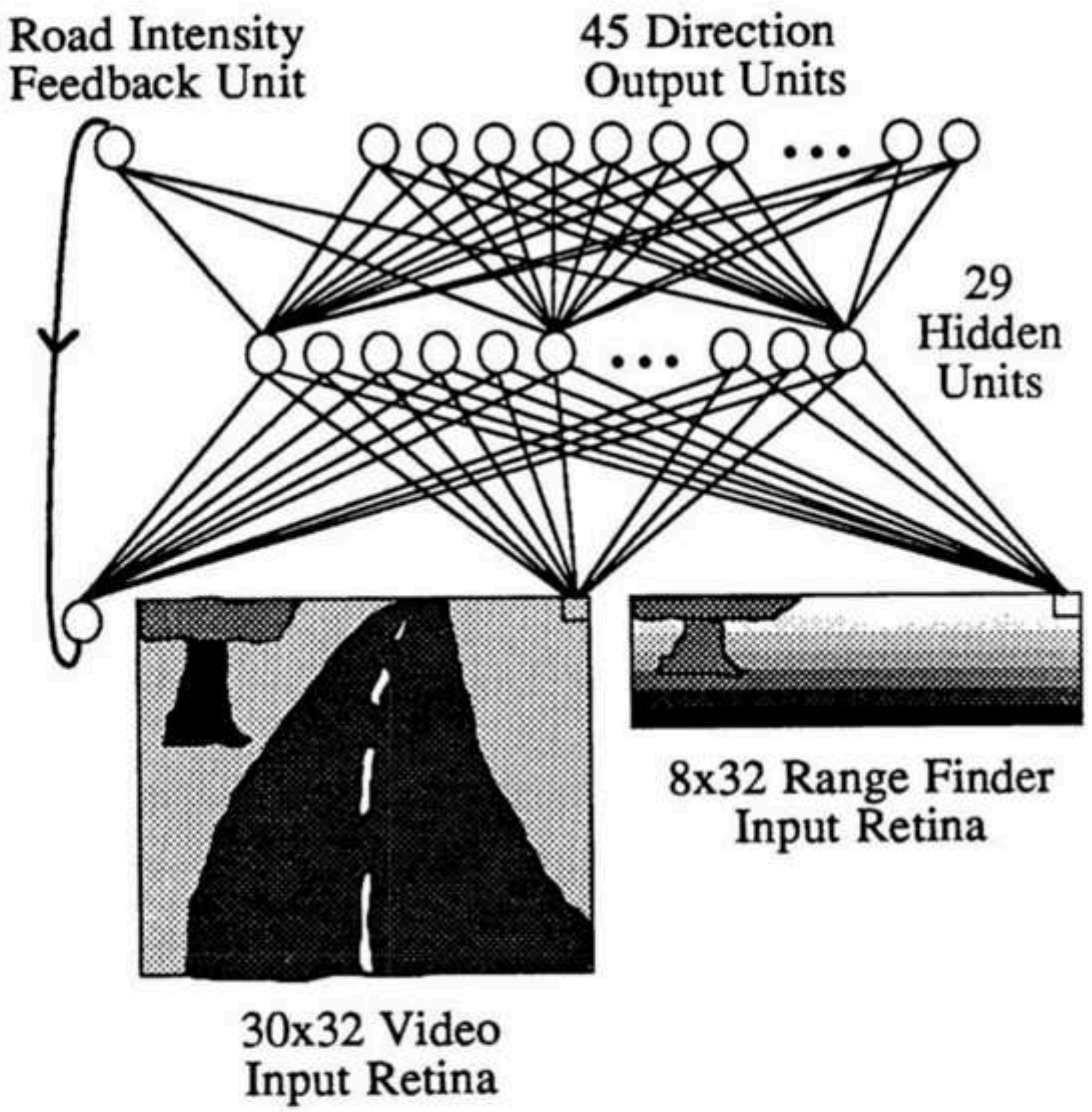
Output: a policy $u_n = \pi(x_n)$

A supervised learning problem!

$\min_\theta \sum_N (\pi_\theta(x_n) - u_n)^2$

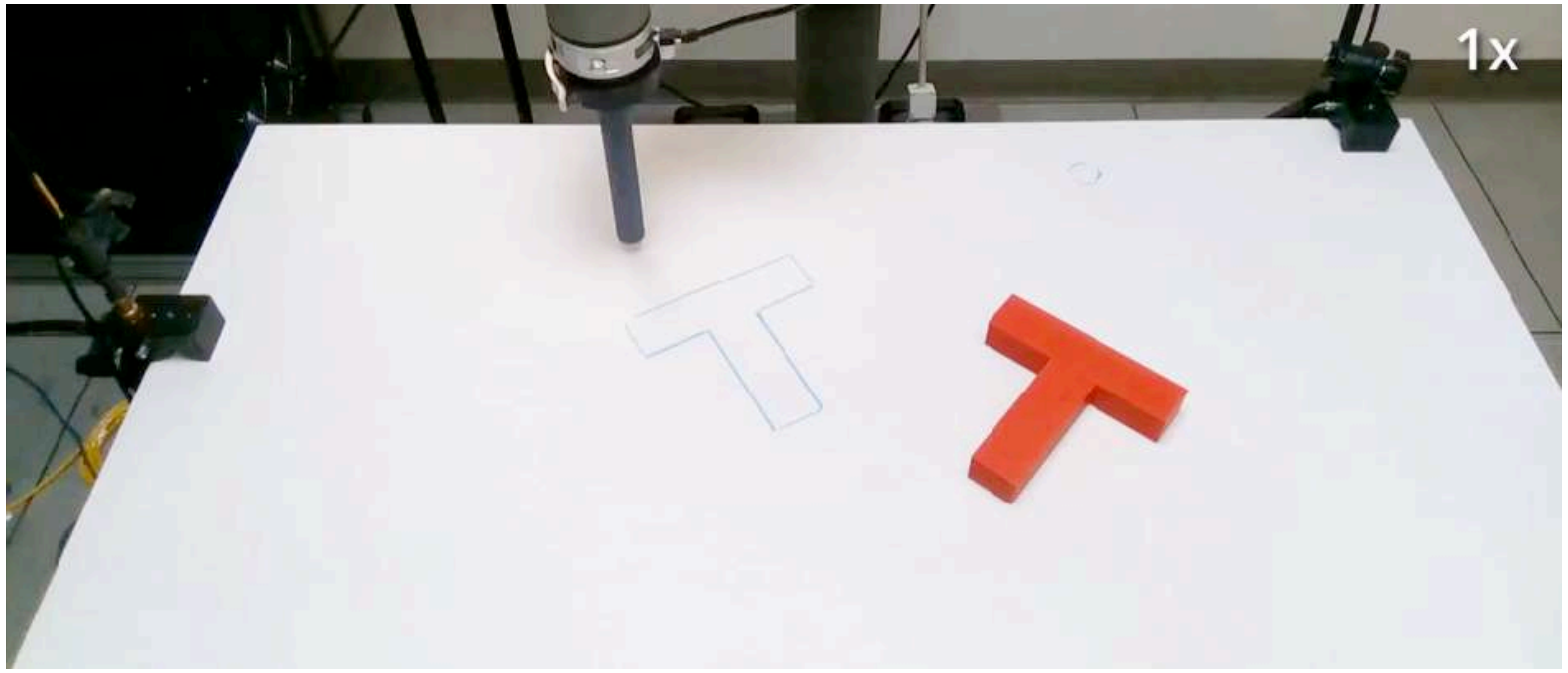
Behavioral cloning: learning <u>policies</u> from demonstrations

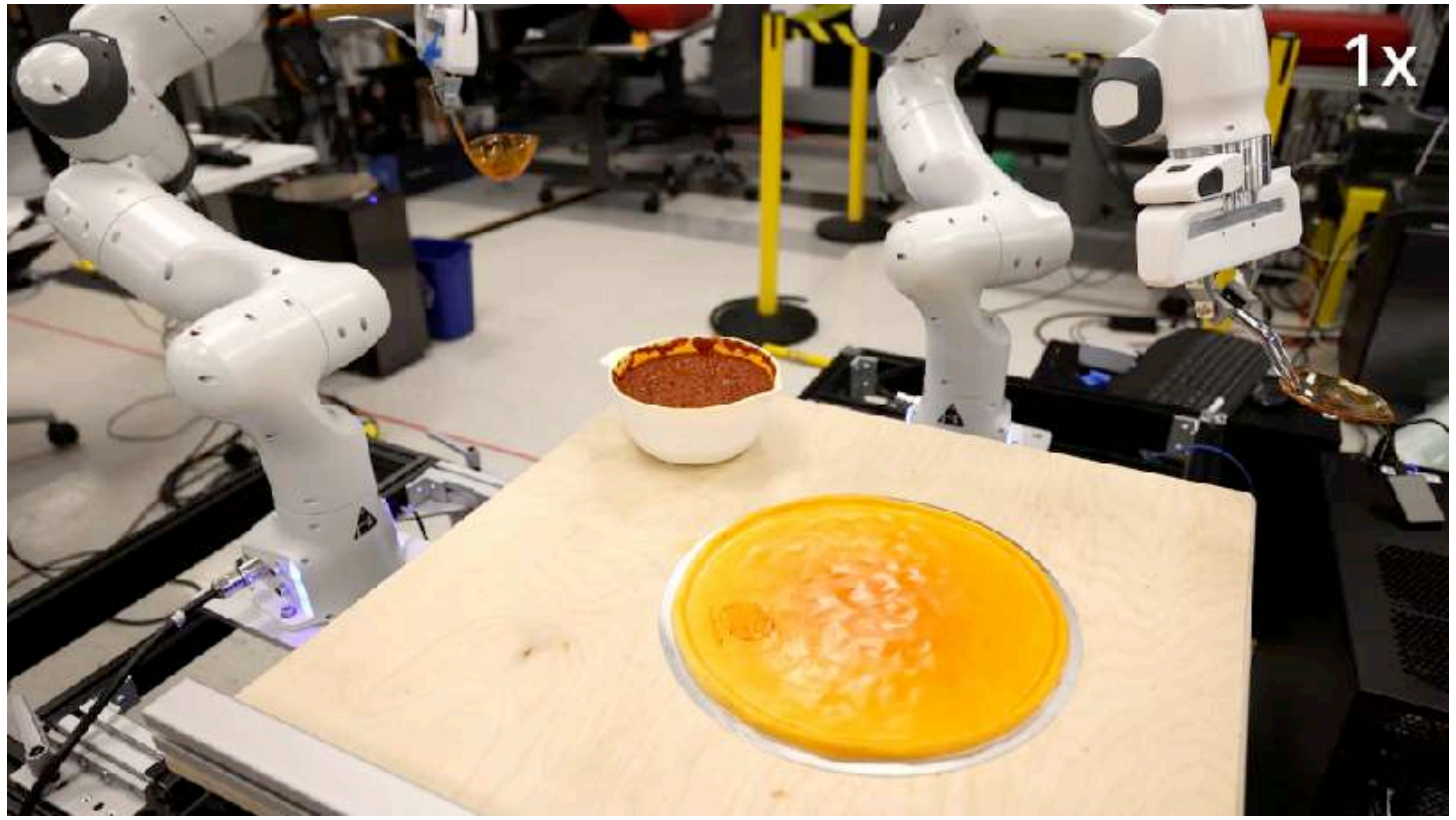
Provide a lot of demonstrations and learn a policy from it



[Pomerleau 1989]

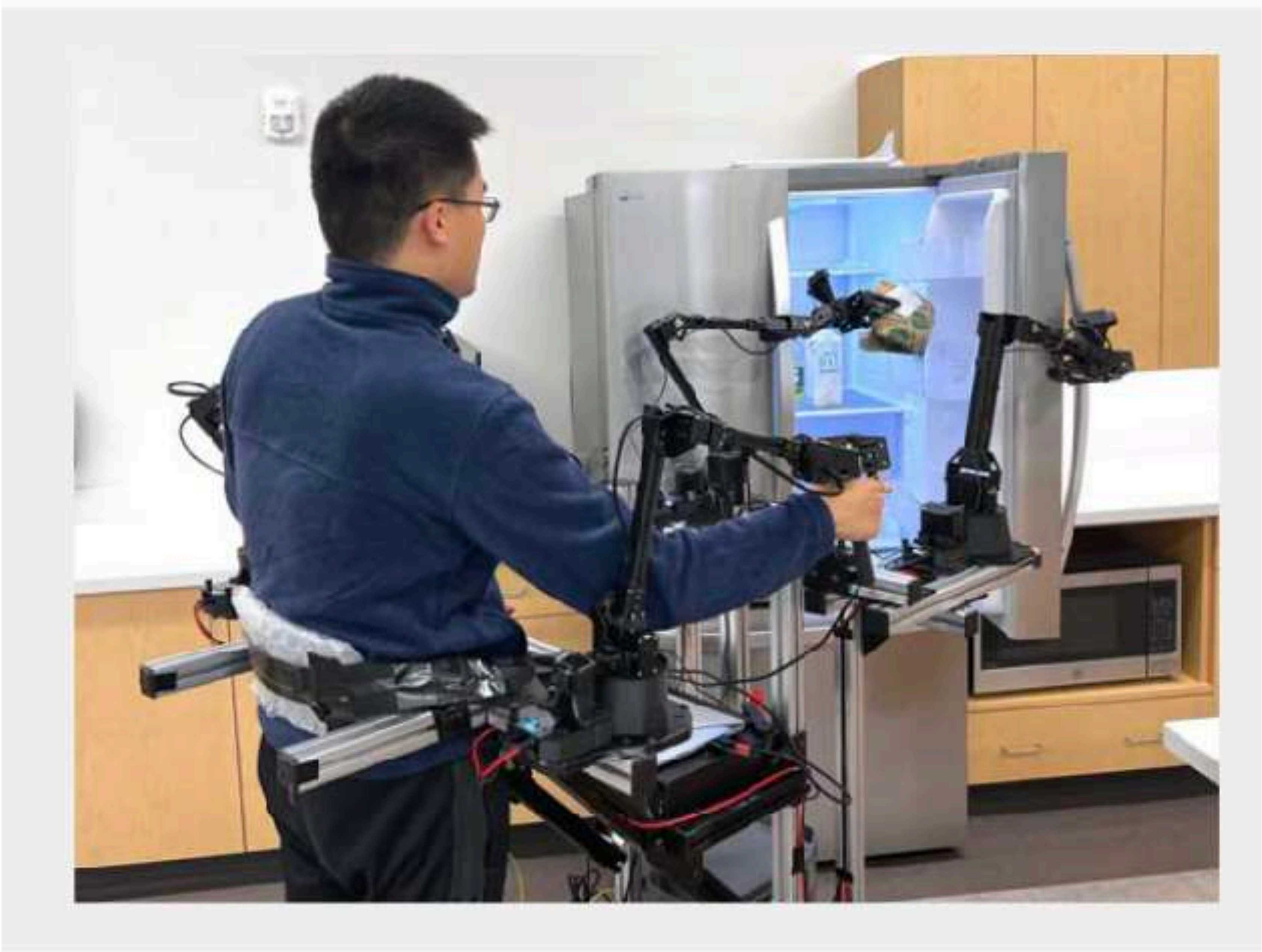
Can learn very complex behaviors





[Chi et al. 2024]

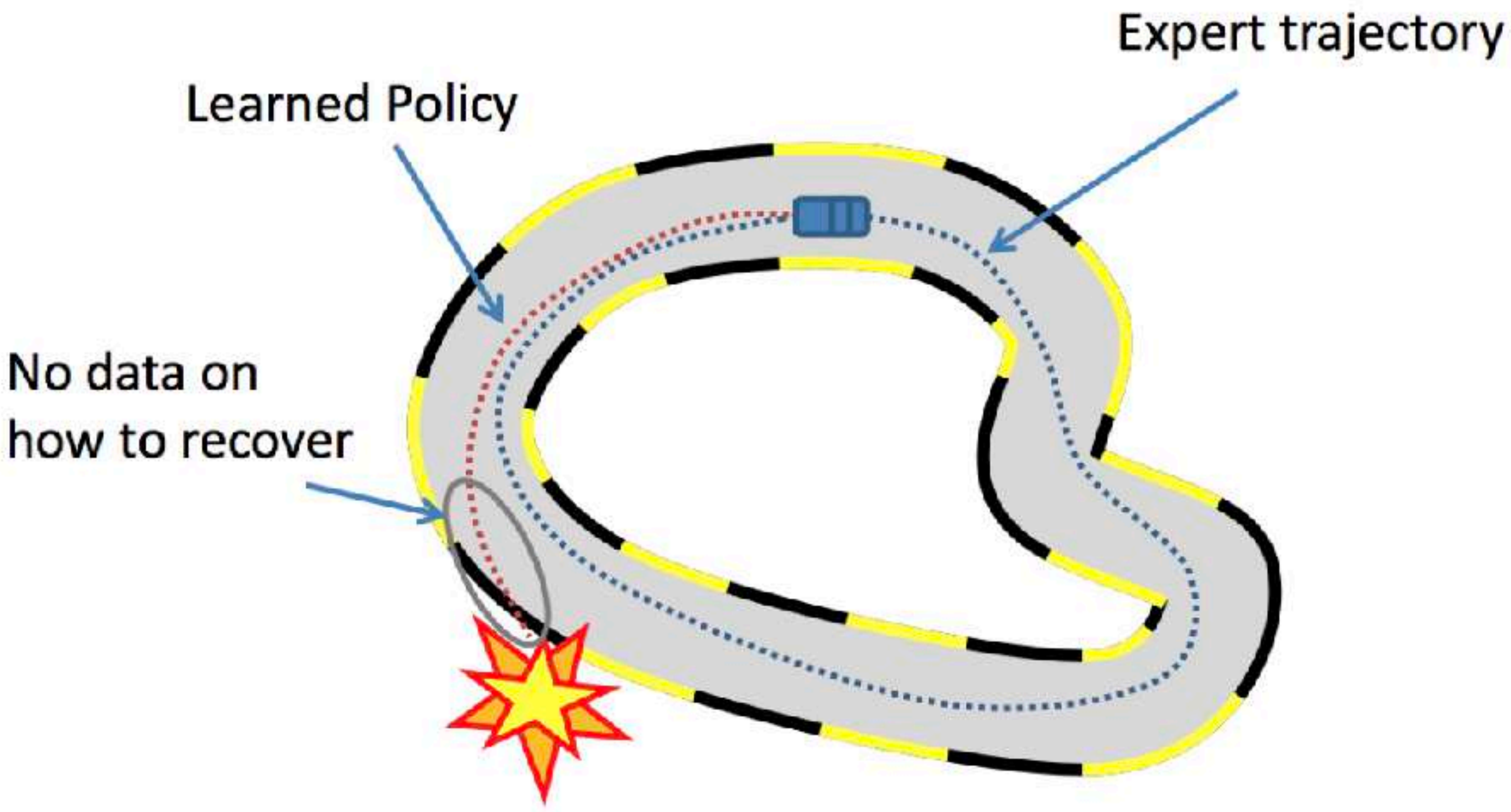
A big trend in robotics!



Challenge: produce a lot of data!
Use advanced NN models (diffusion models, transformers, etc)

Behavioral cloning: learning <u>policies</u> from demonstrations

Problem: compounding errors leads the robot out of demonstration distribution



Dataset aggregation DAGGER

Idea: as the robot does into “unseen territory” collect data and ask an “expert” to provide the correct control (In effect we relabel the data the robot is collecting)

[Ross et al. 2011]

Dataset aggregation DAGGER

Initialize $\mathcal{D} \leftarrow \emptyset$.
Initialize $\hat{\pi}_1$ to any policy in Π.
for $i = 1$ **to** N **do**
  Let $\pi_i = \beta_i \pi^* + (1 - \beta_i)\hat{\pi}_i$.
  Sample T-step trajectories using π_i.
  Get dataset $\mathcal{D}_i = \{(s, \pi^*(s))\}$ of visited states by π_i and actions given by expert.
  Aggregate datasets: $\mathcal{D} \leftarrow \mathcal{D} \bigcup \mathcal{D}_i$.
  Train classifier $\hat{\pi}_{i+1}$ on $\mathcal{D}$.
end for
Return best $\hat{\pi}_i$ on validation.

Algorithm 3.1: DAGGER Algorithm.

[Ross et al. 2011]

Back to reinforcement learning

Combining RL and BC to learn visuomotor policies

Can we add vision sensors into RL?

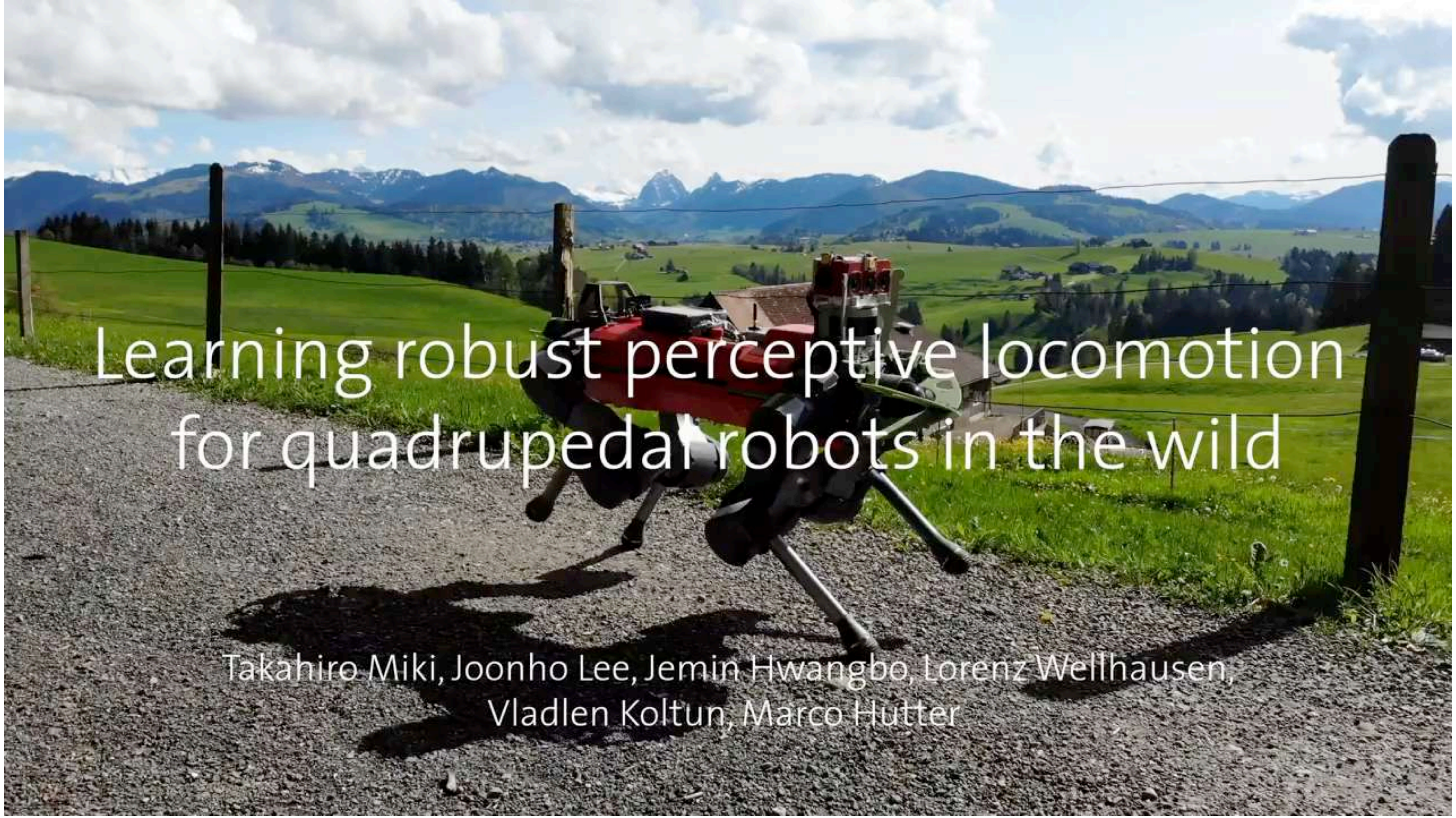
Combining RL and BC to learn visuomotor policies

RL usually does not work for complex robotic tasks if we provide only sensor information (e.g. vision + position sensors)
=> the problem is too difficult for algorithms to converge

Idea:
=> learn a policy with <u>all</u> the information using RL
=> "copy" the policy using only available sensors using behavior cloning

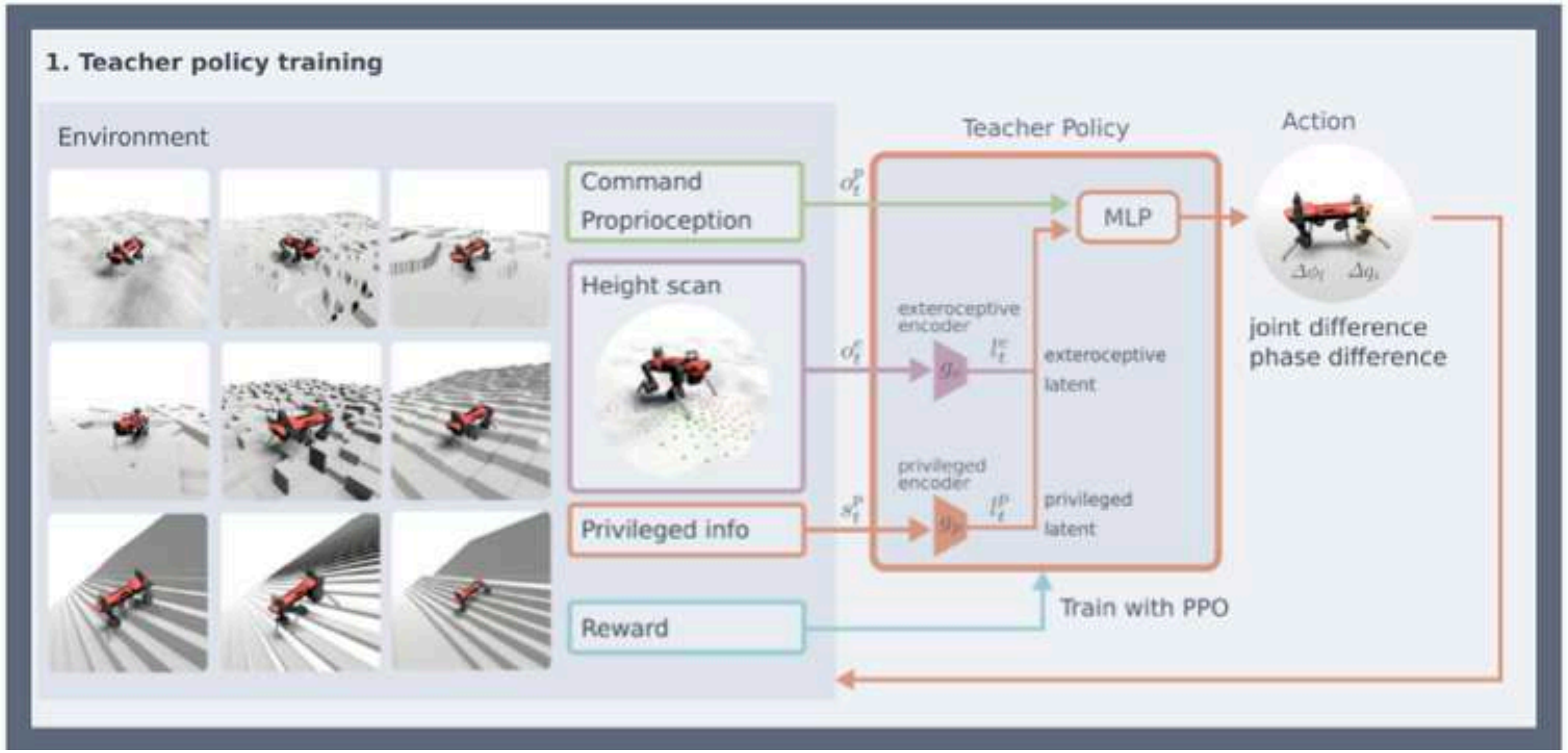
Combining RL and BC to learn visuomotor policies

Learning various behaviors

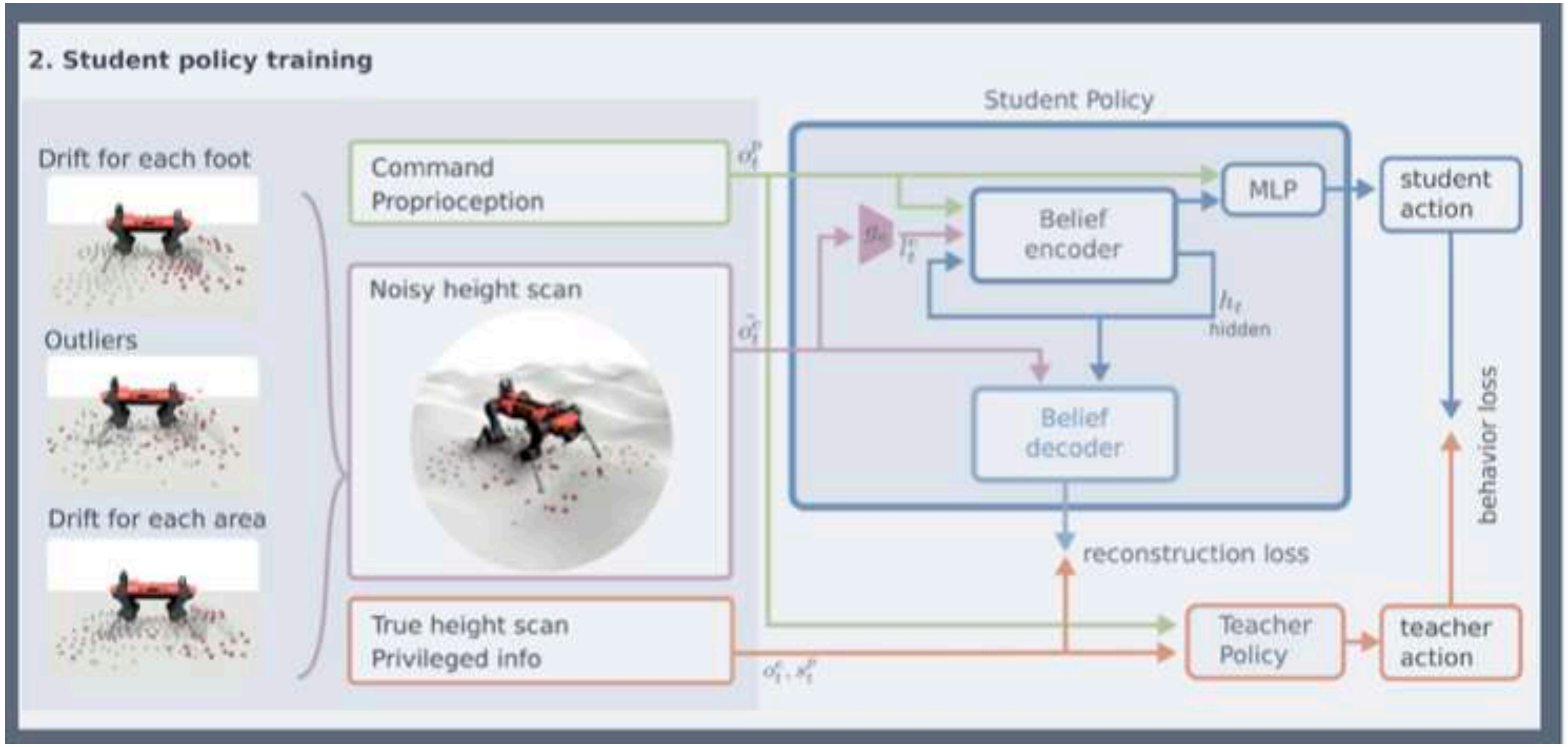


[Miki et al. Science 2022]

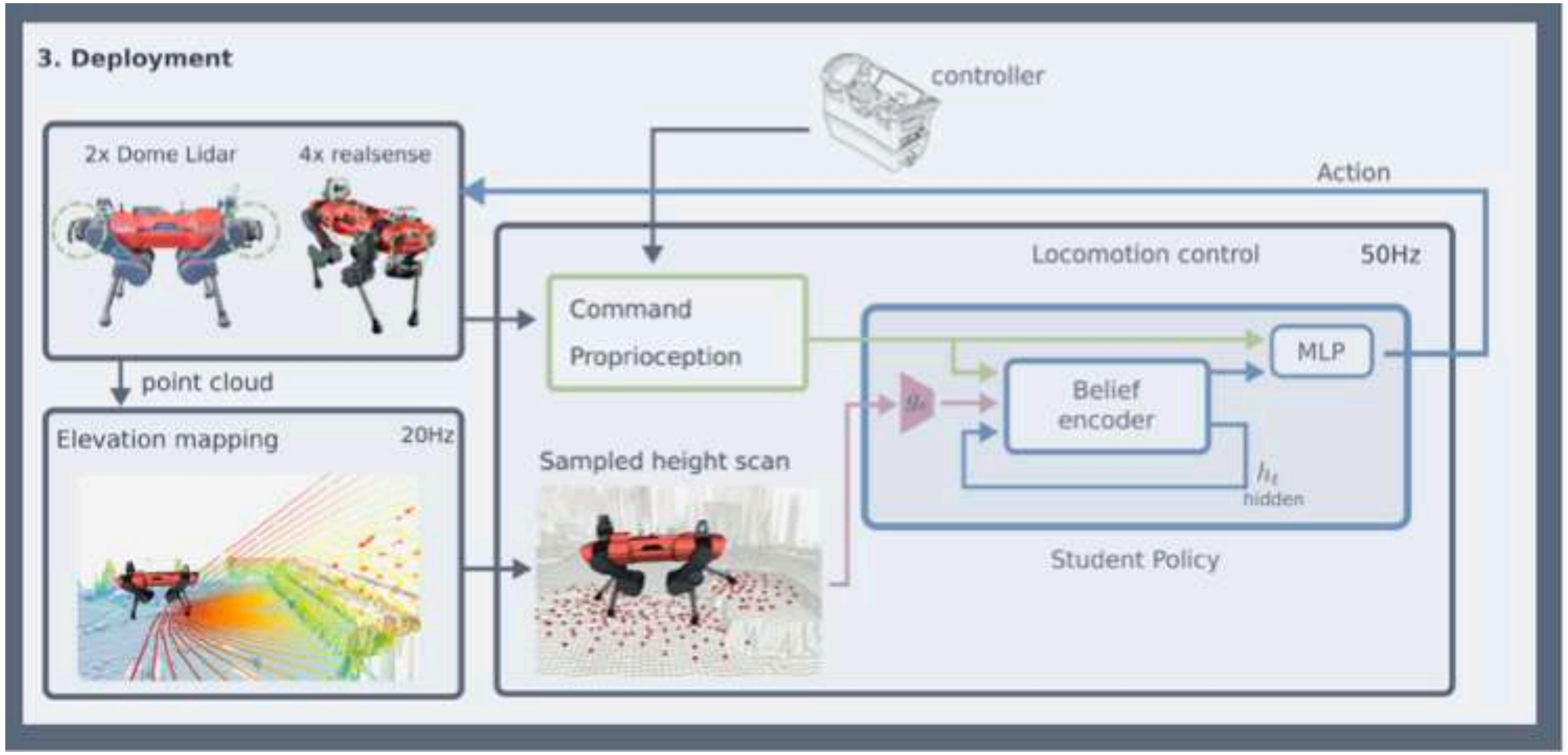
Combining RL and BC to learn visuomotor policies

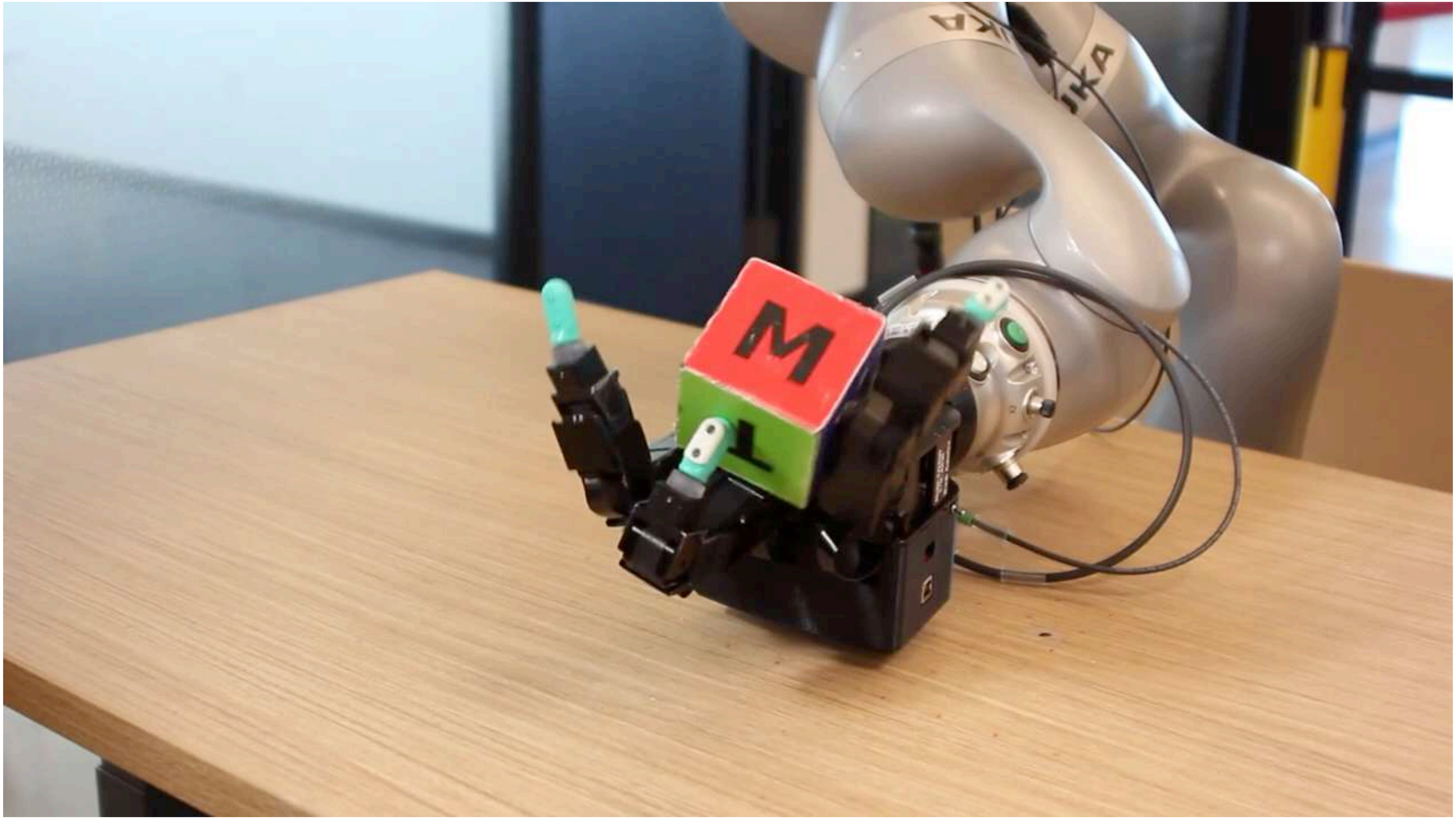


Combining RL and BC to learn visuomotor policies



Combining RL and BC to learn visuomotor policies

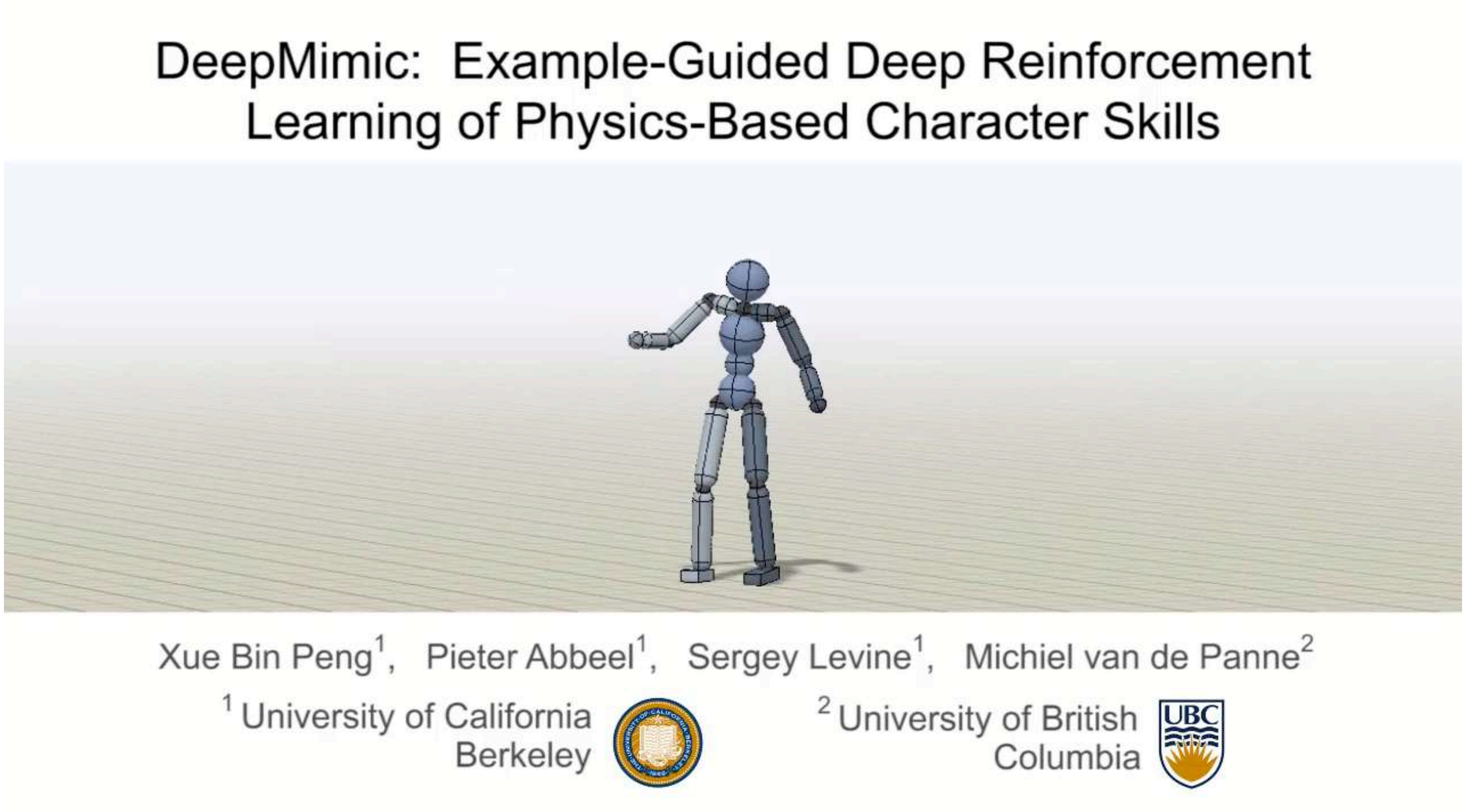




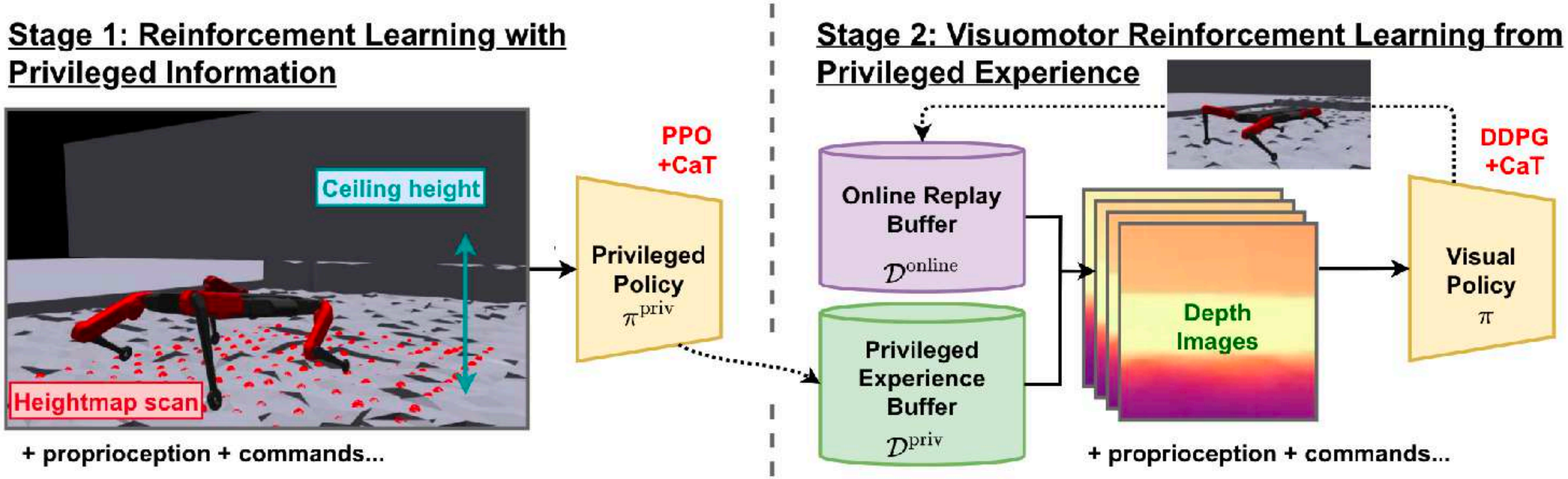
[Handa et al. 2022]

Using demonstrations to bootstrap RL

RL to imitate demonstrations

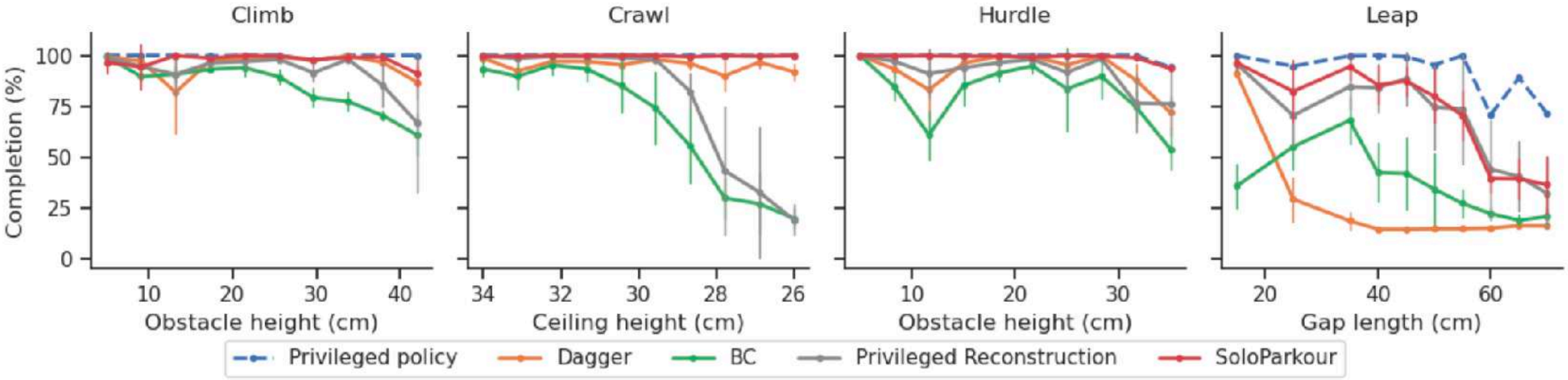


Adding demonstrations in the replay buffer

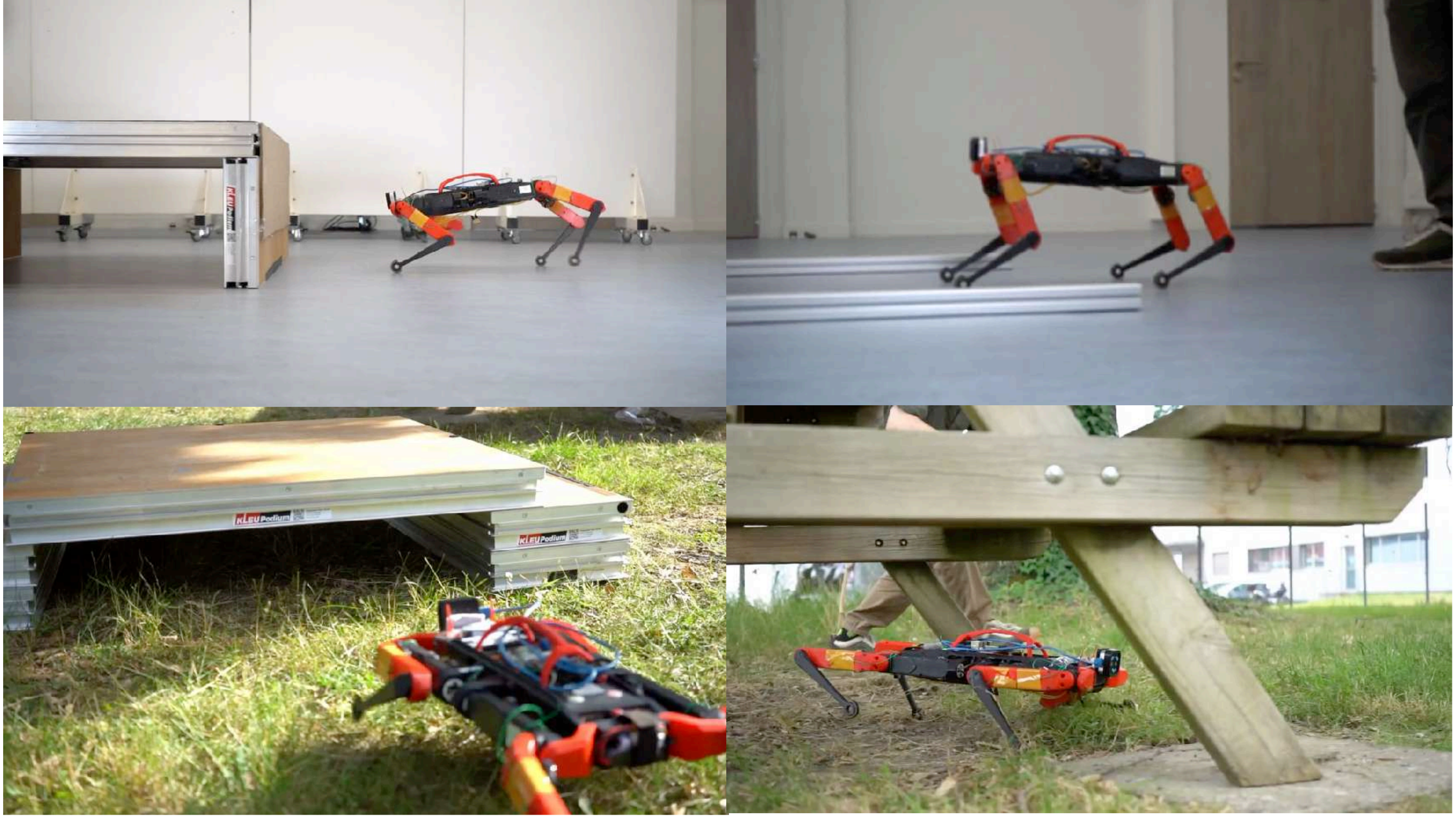


[Chane-Sane et al. CoRL 2024]

Adding demonstrations in the replay buffer



[Chane-Sane et al. CoRL 2024]



[Chane-Sane et al. CoRL 2024]

Learning cost functions from demonstrations

Inverse RL and apprenticeship learning

Inverse RL / inverse OC

Can we infer the cost function from a demonstration?

Useful for:

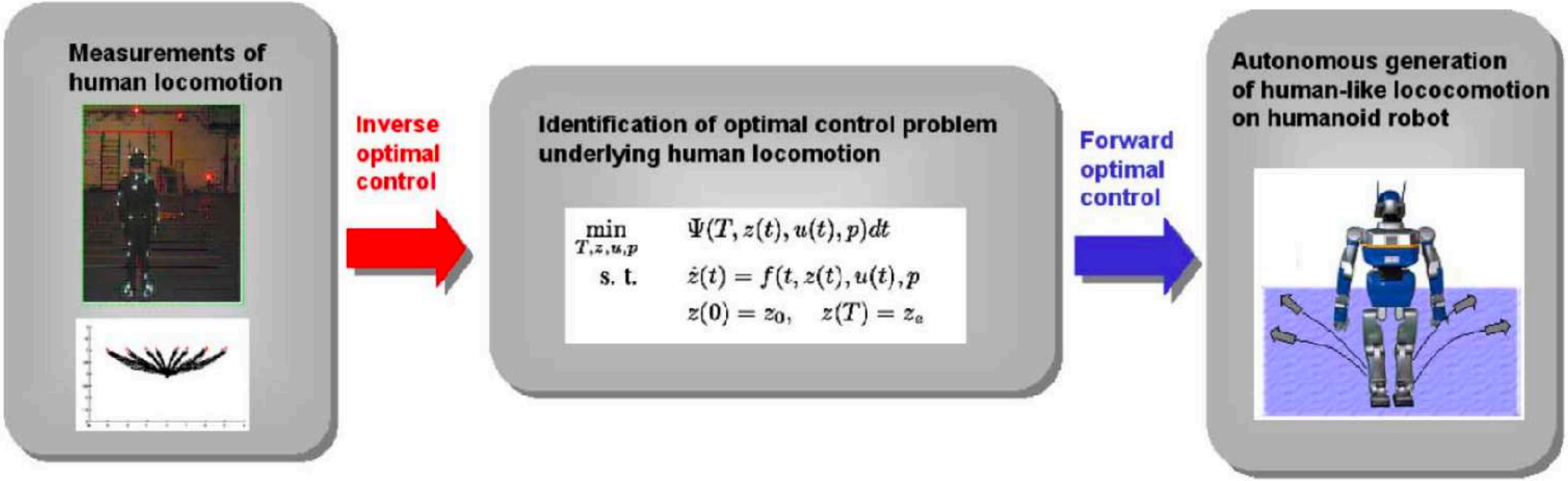
- Learning from demonstrations
- Apprenticeship learning
- Transferring skills across robots
- Also… analyzing human behavior

Inverse RL / inverse OC

Can we infer the cost function from a demonstration?

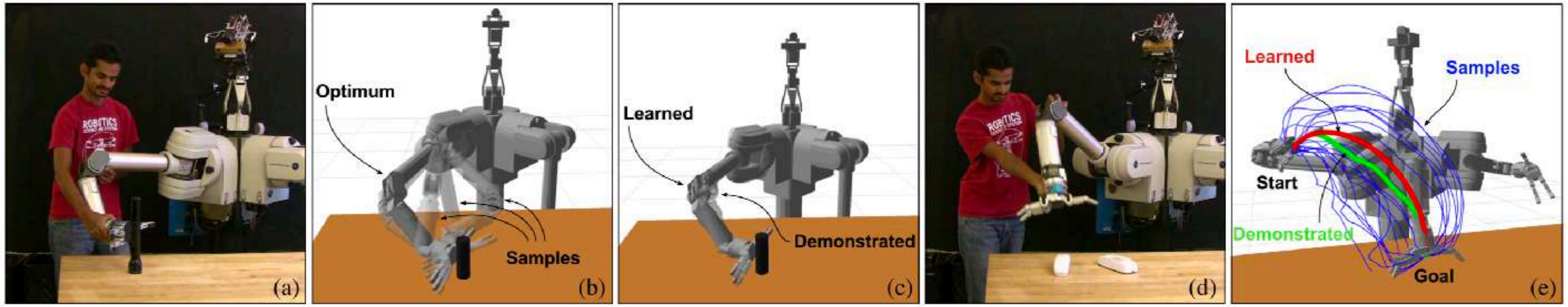
Inverse optimal control / inverse reinforcement learning

[Mombaur et al. 2010]



Inverse optimal control / inverse reinforcement learning

[Kalakrishnan et al. 2013]



Model-based reinforcement learning

Model-based reinforcement learning

We can learn:

- a value function
- a policy
- a model?

Model-based RL
=> learn a model + do optimal control with the model

Model-based reinforcement learning

How do we learn a model?

If the dynamics is linear

$$x_{n+1} = Ax_n + Bu_n$$

we can find A and B from data
=> regression problem

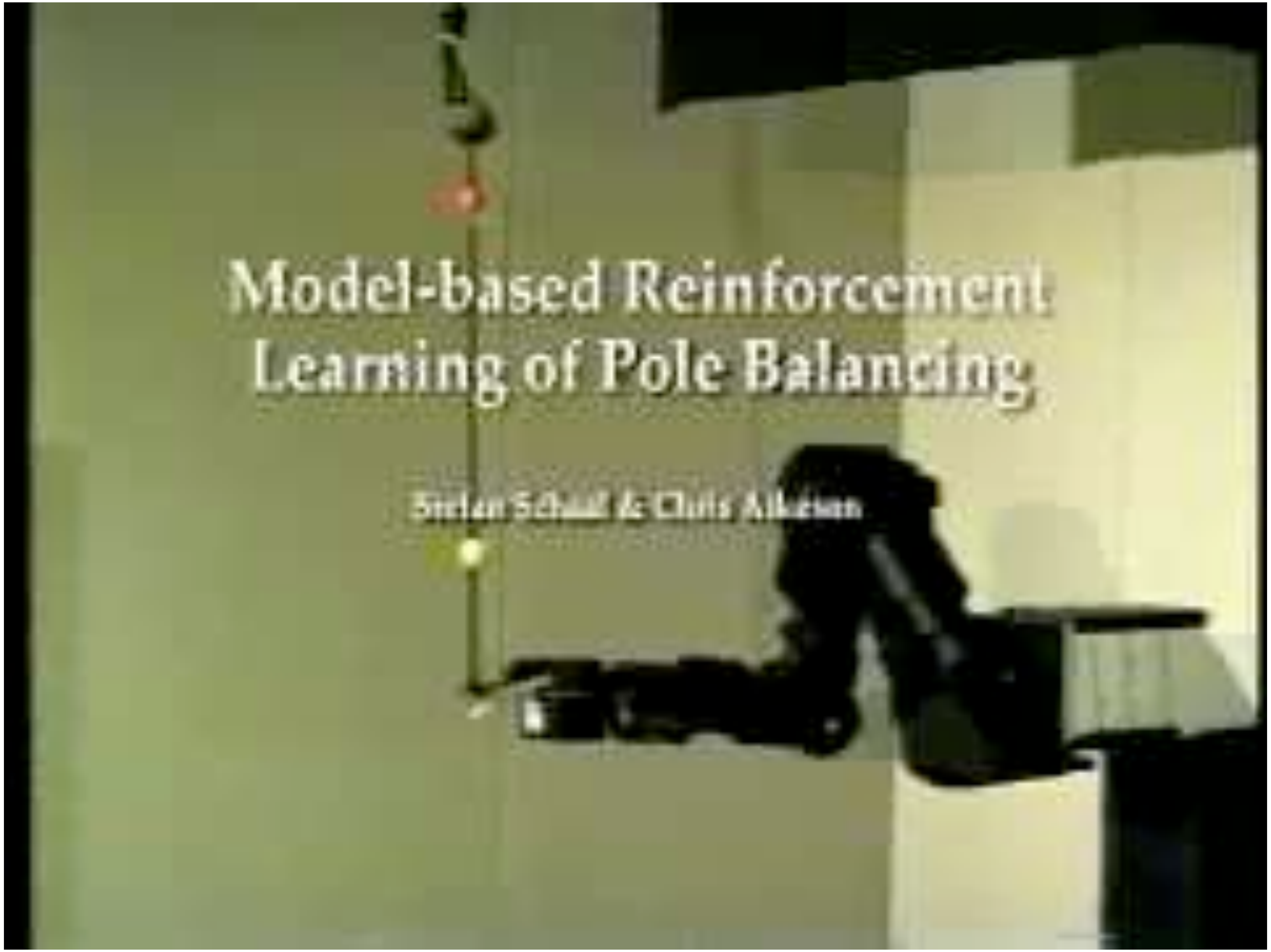
Nonlinear models

1. Use a function approximator for nonlinear functions that is "linearization" friendly
   (e.g. locally weighted regression, Schaal et al. 1997 or Gaussian Processes, Deisenroth et al. 2011)
   => good to do LQR and related, exploit linearity

2. Learn a nonlinear model
   => typically linearization is problematic - might need other techniques to solve OC problems (e.g. cross-entropy methods)

Model-based reinforcement learning

[Schaal and Atkeson ~1995]



Model-based reinforcement learning

[Schaal and Atkeson ~1995]

Advantages of model-based RL

- It tends to be sample-efficient
  => can be used on robots
- Can be used to solve other tasks (i.e. we can change the cost function and keep the model)
- It is easy to compute a locally optimal policy (trajectory optimization) while adding constraints

Issues / drawbacks

- generating enough data to learn the model
- what controller do we use to generate the first samples?
- difficulty to learn models capable to predict long in the future (nonlinear dynamics is tricky)
- mapping from model to policy might not work
- still need to solve an OC problem all the time

Combining everything

Apprenticeship learning

[Abbeel, 2010]

Apprenticeship learning

[Abbeel, 2010]

Apprenticeship learning

[Abbeel, 2010]