Data Efficiency for RL in Control Applications

Seminar for FDD3359 – Reinforcement Learning

Key to RL Successes

- AlphaGo
- Environment Model
  - Exact $p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$
  - Cheap to query
  - Extensive offline training
  - Online Planning
    > *MCTS Rollouts*
    > *Value function from offline training*

Images from Wired, Business

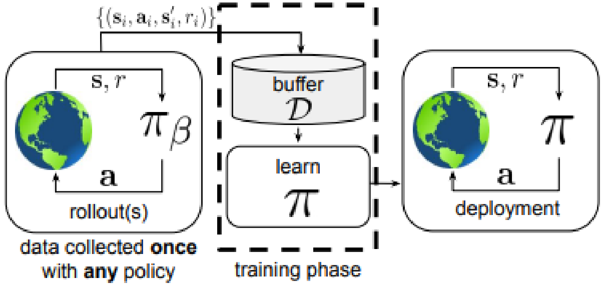
Real Systems

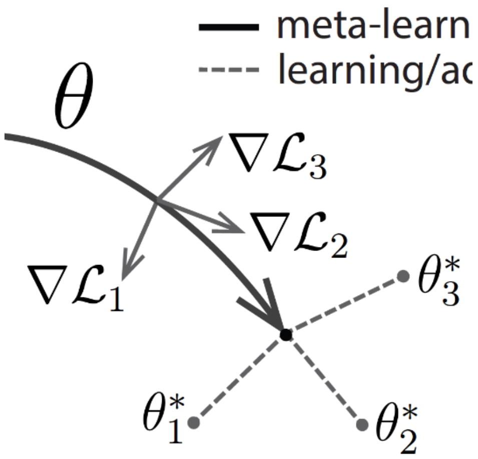
- Most problems
- Environment Model
  - No exact $p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$
  - Expensive/dangerous to query
  - Limited data for offline training
  - Online Planning?


Images from UniTree, ABB and NASDAQ

Possible Solutions

- Offline Reinforcement Learning
  - Fixed dataset
  - Avoid online environment query
- Meta Reinforcement Learning
  - Few-shot adaptation for target task
  - Limit environment query
- ...

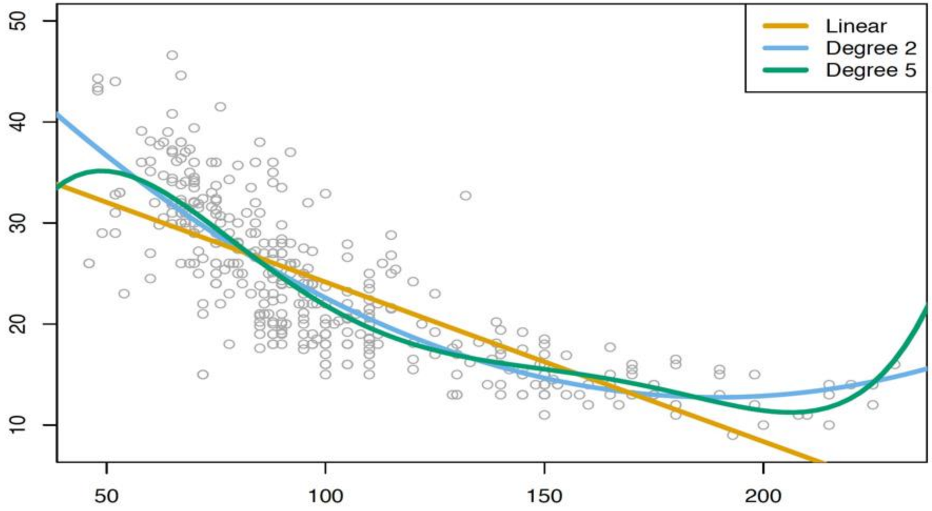




General Rule for Practising Machine Learning

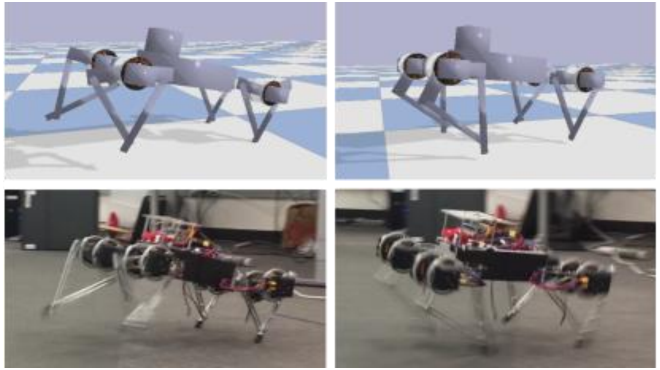
- More data
  - High capacity model
  - Overfitting

- More domain knowledge
  - Choose “right” model
  - Limit the parameter space

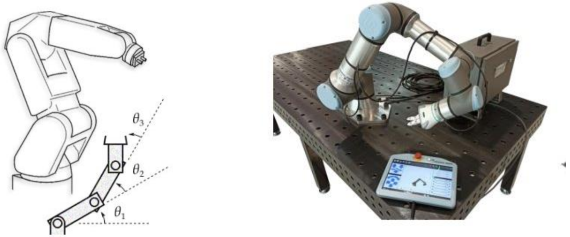


More Data...

- Synthetic data from simulation
  - Various robot dynamics simulators



- Sim-to-real gap
  - Dynamics
    - > *Contact/actuator/friction...*
  - Perception



Images from PyBullet, https://arxiv.org/abs/2009.13303 and NVIDIA

More Data...

Research Article | ETH Zurich and Intel | 1

Learning Quadrupedal Locomotion over Challenging Terrain

JOONHO LEE[1,*], JEMIN HWANGBO[1,2,†], LORENZ WELLHAUSEN[1], VLADLEN KOLTUN[3], AND MARCO HUTTER[1]

[1] *Robotic Systems Lab, ETH Zurich, Zurich, Switzerland*
[2] *Robotics and Artificial Intelligence Lab, KAIST, Daejeon, Korea*
[3] *Intelligent Systems Lab, Intel, Santa Clara, CA, USA*
[†] *Substantial part of the work was carried out during his stay at 1*
[*] *Corresponding author: jolee@ethz.ch*

This is the accepted version of Science Robotics Vol. 5, Issue 47, eabc5986 (2020)
DOI: 10.1126/scirobotics.abc5986

Some of the most challenging environments on our planet are accessible to quadrupedal animals but remain out of reach for autonomous machines. Legged locomotion can dramatically expand the operational domains of robotics. However, conventional controllers for legged locomotion are based on elaborate state machines that explicitly trigger the execution of motion primitives and reflexes. These designs have escalated in complexity while falling short of the generality and robustness of animal locomotion. Here we present a radically robust controller for legged locomotion in challenging natural environments. We present a novel solution to incorporating proprioceptive feedback in locomotion control and demonstrate remarkable zero-shot generalization from simulation to natural environments. The controller is trained by reinforcement learning in simulation. It is based on a neural network that acts on a stream of proprioceptive signals. The trained controller has taken two generations of quadrupedal ANYmal robots to a variety of natural environments that are beyond the reach of prior published work in legged locomotion. The controller retains its robustness under conditions that have never been encountered during training: deformable terrain such as mud and snow, dynamic footholds such as rubble, and overground impediments such as thick vegetation and gushing water. The presented work opens new frontiers for robotics and indicates that radical robustness in natural environments can be achieved by training in much simpler domains.

More Data... from More Knowledge

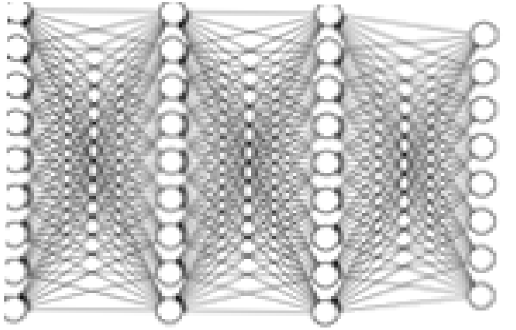
- Dedicated simulation RAI
  - Contact model
  - Actuator dynamics
  - Disturbances
  - Parameter randomization

- Terrain curriculum

- Teacher/student networks
  - Privileged simulation params

“Right” Policy

$$\pi(\mathbf{a}|\mathbf{s})$$



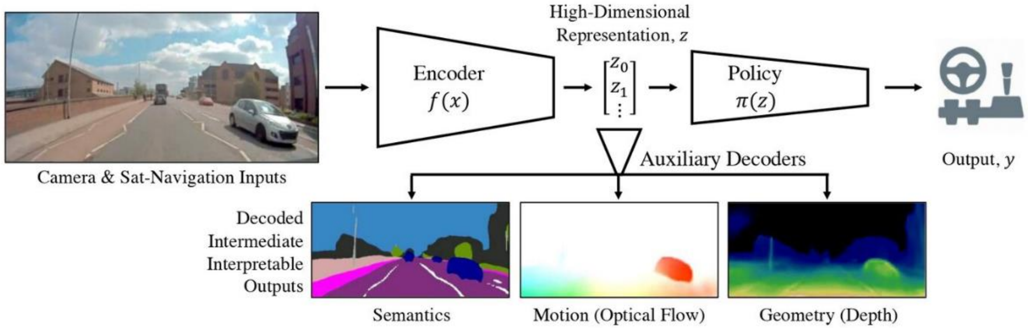
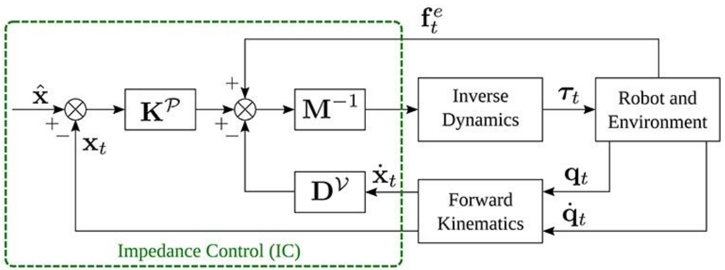
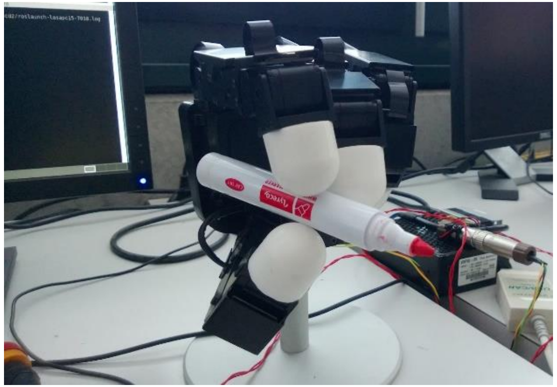


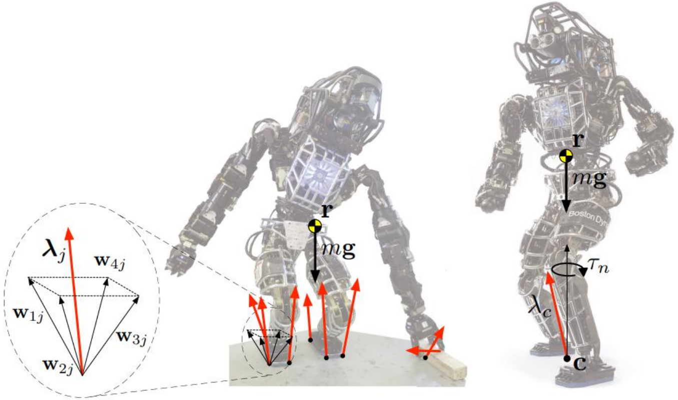
Image from https://wayve.ai/blog/learned-urban-driving

“Right” Policy for Controlling Robots



$$
\begin{aligned}
\underset{\Gamma}{\text{minimize}} \quad & \sum_{k=1}^{M} L(\mathbf{q}[k], \mathbf{v}[k], \ddot{\mathbf{r}}[k], \boldsymbol{\lambda}[k], h[k]) \\
\text{subject to} \quad & m\ddot{\mathbf{r}}[k] = m\mathbf{g} + \sum_j \boldsymbol{\lambda}_j[k] && \text{(linear momentum)} \\
& \mathbf{k}[k] = \mathbf{A}_G^{\mathbf{k}}(\mathbf{q}[k])\mathbf{v}[k] && \text{(angular momentum)} \\
& \dot{\mathbf{k}}[k] = \sum_j (\mathbf{c}_j[k] - \mathbf{r}[k]) \times \boldsymbol{\lambda}_j && \text{(angular momentum rate)} \\
& \forall_j \quad \boldsymbol{\lambda}_j[k] = \sum_{i=1}^{N_d} \beta_{ij}[k]\mathbf{w}_{ij} && \text{(friction)} \\
& \forall_{i,j} \quad \beta_{ij}[k] \geq 0 \\
& \mathbf{r}[k] = COM(\mathbf{q}[k]) && \text{(COM location)} \\
& \text{Kinematic constraints} \\
& \text{Time integration constraints}
\end{aligned}
$$





Images from Abu Dakka and Saveriano 2020 and Kuindersma et al 2014

Perspective from Optimal Control

Reinforcement Learning		Optimal Control	
State/Action	$\mathbf{s}$ $\mathbf{a}$	State/Control	$\mathbf{x}$ $\mathbf{u}$
Reward	$r(\mathbf{s}, \mathbf{a})$	Cost	$l(\mathbf{x}, \mathbf{u})$
State transition	$p(\mathbf{s}_{t+1} \| \mathbf{s}_t, \mathbf{a}_t)$	Dynamics	$\dot{\mathbf{x}} = f(\mathbf{x}, \mathbf{u})$

$$\arg\max_{\pi} \mathbb{E}_{\tau \sim p_\pi(\tau)} [r(\mathbf{s}_t, \mathbf{a}_t)]$$

$$\tau \sim p_\pi = p(\mathbf{s}_0) \prod_t \boxed{p(\mathbf{s}_{t+1} | \mathbf{s}_t, \mathbf{a}_t)} \pi(\mathbf{a}_t | \mathbf{s}_t)$$

$$\arg\min_{\mathbf{u}} \int l(\mathbf{x}(t), \mathbf{u}(t)) dt$$

$$s.t. \quad \dot{\mathbf{x}}(t) = f(\mathbf{x}(t), \mathbf{u}(t))$$

Revisit to Value Iteration

value iteration algorithm:

1. set $Q(\mathbf{s}, \mathbf{a}) \leftarrow r(\mathbf{s}, \mathbf{a}) + \gamma E[V(\mathbf{s}')]$
2. set $V(\mathbf{s}) \leftarrow \max_{\mathbf{a}} Q(\mathbf{s}, \mathbf{a})$

$$V^*(\mathbf{x}) = \max_a [r(\mathbf{s}, \mathbf{a}) + \gamma \mathbb{E}_{\mathbf{s}' \sim p(\mathbf{s}'|\mathbf{s},\mathbf{a})}(V^*(\mathbf{s}'))]$$

for optimal policy and any state

Hamilton-Jacobi-Bellman Equation

- Similar equation in continuous case

$$V^*(\mathbf{x}, t) = \min_{\mathbf{u}} \int_t l(\mathbf{x}(t), \mathbf{u}(t))dt$$

- We have

$$-\frac{\partial V^*(\mathbf{x},t)}{\partial t} = \min_{\mathbf{u}}[l(\mathbf{x}, \mathbf{u}) + \frac{\partial V^*(\mathbf{x},t)}{\partial \mathbf{x}} \cdot f(\mathbf{x}, \mathbf{u})]$$

- vs Value Iteration

$$V^*(\mathbf{x}) = \max_a[r(\mathbf{s}, \mathbf{a}) + \gamma \mathbb{E}_{\mathbf{s}' \sim p(\mathbf{s}'|\mathbf{s},\mathbf{a})}(V^*(\mathbf{s}'))]$$

HJB Equation - Derivation

- For a small time interval to future

$$V^*(\mathbf{x}, t) = \min_{\mathbf{u}} [V^*(\mathbf{x}(t+dt), t+dt) + \int_t^{t+dt} l(\mathbf{x}(\tau), \mathbf{u}(\tau)) d\tau]$$

- Taylor expansion of $V^*(\mathbf{x}(t+dt), t+dt)$

$$V^*(\mathbf{x}(t+dt), t+dt) = V^*(\mathbf{x}, t) + \frac{\partial V^*(\mathbf{x},t)}{\partial t} dt + \frac{\partial V^*(\mathbf{x},t)}{\partial \mathbf{x}} \cdot \boxed{\dot{\mathbf{x}}} dt + o(dt)$$

$$\dot{\mathbf{x}} = f(\mathbf{x}, \mathbf{u})$$

HJB Equation - Derivation

- Ignore high-order infinitesimal and subtract $V^*(\mathbf{x}, t)$

$$V^*(\mathbf{x},t) = \min_{\mathbf{u}}[V^*(\mathbf{x}(t+dt), t+dt) + \int_t^{t+dt} l(\mathbf{x}(\tau), \mathbf{u}(\tau))d\tau]$$

$$\downarrow$$

$$0 = \frac{\partial V^*(\mathbf{x},t)}{\partial t}dt + \min_{\mathbf{u}}[\frac{\partial V^*(\mathbf{x},t)}{\partial \mathbf{x}} \cdot f(\mathbf{x},\mathbf{u})dt + \int_t^{t+dt} l(\mathbf{x}(\tau), \mathbf{u}(\tau))d\tau]$$

HJB Equation - Derivation

- Divide by $dt \to 0$

$$0 = \frac{\partial V^*(\mathbf{x},t)}{\partial t}dt + \min_{\mathbf{u}}[\frac{\partial V^*(\mathbf{x},t)}{\partial \mathbf{x}} \cdot f(\mathbf{x},\mathbf{u})dt + \int_t^{t+dt} l(\mathbf{x}(\tau),\mathbf{u}(\tau))d\tau]$$

$$-\frac{\partial V^*(\mathbf{x},t)}{\partial t} = \min_{\mathbf{u}}[l(\mathbf{x},\mathbf{u}) + \frac{\partial V^*(\mathbf{x},t)}{\partial \mathbf{x}} \cdot f(\mathbf{x},\mathbf{u})]$$

- Note $\frac{\partial V^*(\mathbf{x},t)}{\partial t} = 0$ if cost-to-go is static $V^*(\mathbf{x})$

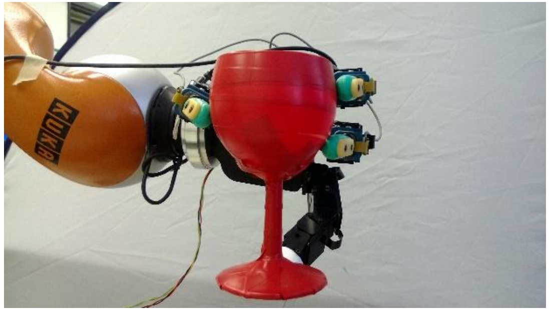
$$0 = \min_{\mathbf{u}}[l(\mathbf{x},\mathbf{u}) + \frac{\partial V^*(\mathbf{x})}{\partial \mathbf{x}} \cdot f(\mathbf{x},\mathbf{u})]$$

Control Affine Dynamics

- Structured $\dot{\mathbf{x}} = f(\mathbf{x}, \mathbf{u})$

$$\dot{\mathbf{x}} = f(\mathbf{x}) + g(\mathbf{x}) \cdot \mathbf{u}$$

- General motion equation of many physical systems, e.g. robots, rigid and deformable objects…



MuJoCo computes both forward and inverse dynamics in continuous time. Forward dynamics are then integrated over the specified `mjModel.opt.timestep` with the chosen numerical integrator. The general equations of motion in continuous time are

$$M\dot{v} + c = \tau + J^T f \tag{1}$$

The Jacobian establishes the relationship between quantities in joint and constraint coordinates. It maps motion vectors (velocities and accelerations) from joint to constraint coordinates: the joint velocity v maps to velocity Jv in constraint coordinates. The transpose of the Jacobian maps force vectors from constraint to joint coordinates: the constraint force f maps to force $J^T f$ in joint coordinates.

The joint-space inertia matrix M is always invertible. Therefore once the constraint force f is known, we can finalize the forward and inverse dynamics computations as

$$\begin{aligned} \text{forward:} \quad & \dot{v} = M^{-1}(\tau + J^T f - c) \\ \text{inverse:} \quad & \tau = M\dot{v} + c - J^T f \end{aligned}$$

Images from https://mujoco.readthedocs.io/en/latest/computation.html

Decomposition of Cost Function

- Structured $l(\mathbf{x}, \mathbf{u}) = c(\mathbf{x}) + \mathbf{u}^\top \mathbf{R}\mathbf{u}$ $\quad \mathbf{R} \succ 0$

- Use them in HJB equation and the optimality condition of $\mathbf{u}$ ...

$$-\frac{\partial V^*(\mathbf{x},t)}{\partial t} = \min_{\mathbf{u}}[l(\mathbf{x}, \mathbf{u}) + \frac{\partial V^*(\mathbf{x},t)}{\partial \mathbf{x}} \cdot f(\mathbf{x}, \mathbf{u})]$$

$$\downarrow$$

$$\mathbf{u}^* = \arg\min_{\mathbf{u}}[c(\mathbf{x}) + \mathbf{u}^\top \mathbf{R}\mathbf{u} + \frac{\partial V^*}{\partial \mathbf{x}} \cdot (f(\mathbf{x}) + g(\mathbf{x}) \cdot \mathbf{u})]$$

Decomposition of Cost Function

$$\mathbf{u}^* = \arg\min_{\mathbf{u}} \left[c(\mathbf{x}) + \mathbf{u}^\top \mathbf{R}\mathbf{u} + \frac{\partial V^*}{\partial \mathbf{x}} \cdot \left(f(\mathbf{x}) + g(\mathbf{x}) \cdot \mathbf{u} \right) \right]$$

$$\downarrow$$

$$\mathbf{u}^* = -\frac{1}{2}\mathbf{R}^{-1} g(\mathbf{x})^\top \left(\frac{\partial V^*(\mathbf{x}, t)}{\partial \mathbf{x}} \right)^\top$$

A closed-form solution...

Further Approximation – Linear Quadratic Regulator

- Lets further the control affine and quadratic approximations

$$\dot{\mathbf{x}} = f(\mathbf{x}) + g(\mathbf{x}) \cdot \mathbf{u} \quad \longrightarrow \quad \dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u}$$

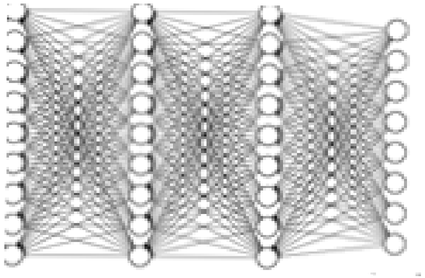
$$l(\mathbf{x}, \mathbf{u}) = c(\mathbf{x}) + \mathbf{u}^\top \mathbf{R}\mathbf{u} \quad \longrightarrow \quad l(\mathbf{x}, \mathbf{u}) = \mathbf{x}^\top \mathbf{Q}\mathbf{x} + \mathbf{u}^\top \mathbf{R}\mathbf{u}$$

- We will have quadratic $V^*(\mathbf{x}) = \mathbf{x}^\top \mathbf{P}\mathbf{x}$
- And a linear solution

$$\mathbf{u}^* = -\frac{1}{2}\mathbf{R}^{-1} g(\mathbf{x})^\top \left(\frac{\partial V^*(\mathbf{x},t)}{\partial \mathbf{x}}\right)^\top \quad \longrightarrow \quad \mathbf{u}^* = -\frac{1}{2}\mathbf{R}^{-1}\mathbf{B}^\top \mathbf{P}\mathbf{x}$$

Power of Linear Policy

- So what if

$\mathbf{x}$  $\mathbf{u}$

$$\mathbf{u} = \mathbf{K}\mathbf{x}$$

- We can always assume differentiability and have local linear approximation...

$$\dot{\mathbf{x}} = f(\mathbf{x}, \mathbf{u})$$

$$\delta\dot{\mathbf{x}} = \frac{\partial f}{\partial x}\Big|_{\mathbf{x},\mathbf{u}} \delta x + \frac{\partial f}{\partial u}\Big|_{\mathbf{x},\mathbf{u}} \delta u$$

Power of Linear Policy

Simple random search of static linear policies is competitive for reinforcement learning

Horia Mania
hmania@berkeley.edu

Aurelia Guy
lia@berkeley.edu

Benjamin Recht
brecht@berkeley.edu

Department of Electrical Engineering and Computer Science
University of California, Berkeley

Abstract

Model-free reinforcement learning aims to offer off-the-shelf solutions for controlling dynamical systems without requiring models of the system dynamics. We introduce a model-free random search algorithm for training static, linear policies for continuous control problems. Common evaluation methodology shows that our method matches state-of-the-art sample efficiency on the benchmark MuJoCo locomotion tasks. Nonetheless, more rigorous evaluation reveals that the assessment of performance on these benchmarks is optimistic. We evaluate the performance of our method over hundreds of random seeds and many different hyperparameter configurations for each benchmark task. This extensive evaluation is possible because of the small computational footprint of our method. Our simulations highlight a high variability in performance in these benchmark tasks, indicating that commonly used estimations of sample efficiency do not adequately evaluate the performance of RL algorithms. Our results stress the need for new baselines, benchmarks and evaluation methodology for RL algorithms.

Algorithm 1 Augmented Random Search (ARS): four versions **V1**, **V1-t**, **V2** and **V2-t**

1: **Hyperparameters:** step-size α, number of directions sampled per iteration N, standard deviation of the exploration noise ν, number of top-performing directions to use b ($b < N$ is allowed only for **V1-t** and **V2-t**)
2: **Initialize:** $M_0 = \mathbf{0} \in \mathbb{R}^{p \times n}$, $\mu_0 = \mathbf{0} \in \mathbb{R}^n$, and $\Sigma_0 = \mathbf{I}_n \in \mathbb{R}^{n \times n}$, $j = 0$.
3: **while** ending condition not satisfied **do**
4: Sample $\delta_1, \delta_2, \ldots, \delta_N$ in $\mathbb{R}^{p \times n}$ with i.i.d. standard normal entries.
5: Collect $2N$ rollouts of horizon H and their corresponding rewards using the $2N$ policies

$$\textbf{V1:} \begin{cases} \pi_{j,k,+}(x) = (M_j + \nu\delta_k)x \\ \pi_{j,k,-}(x) = (M_j - \nu\delta_k)x \end{cases}$$

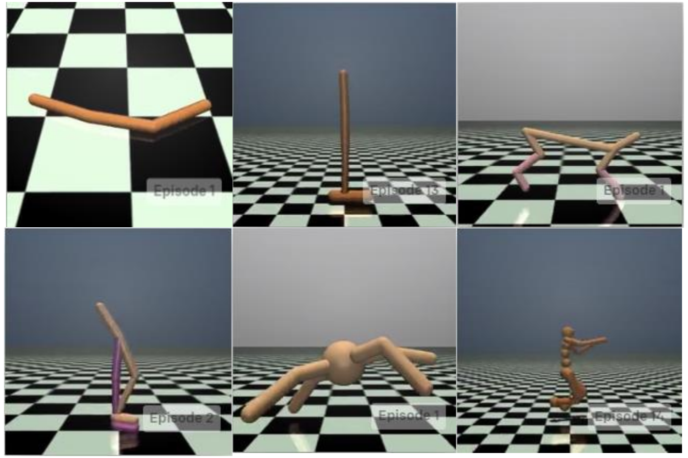
$$\textbf{V2:} \begin{cases} \pi_{j,k,+}(x) = (M_j + \nu\delta_k)\,\mathrm{diag}\,(\Sigma_j)^{-1/2}\,(x - \mu_j) \\ \pi_{j,k,-}(x) = (M_j - \nu\delta_k)\,\mathrm{diag}(\Sigma_j)^{-1/2}(x - \mu_j) \end{cases}$$

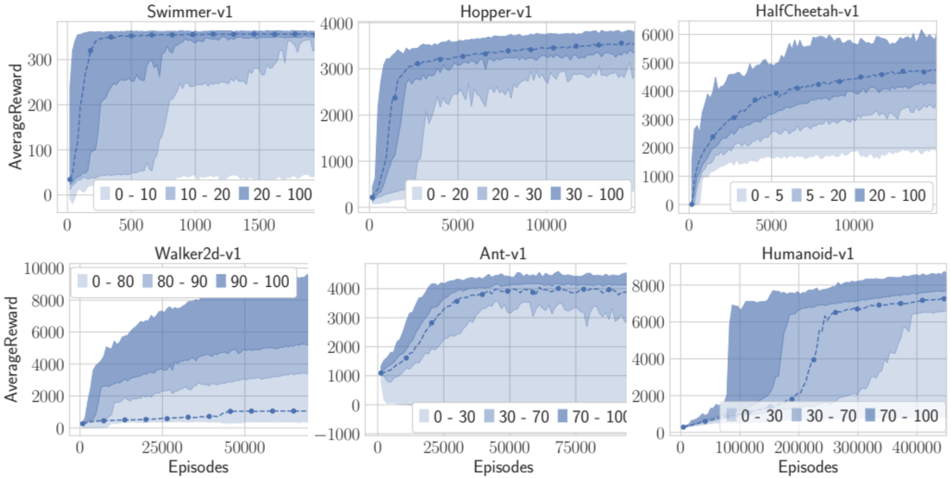
for $k \in \{1, 2, \ldots, N\}$.
6: **V1-t, V2-t:** Sort the directions δ_k by $\max\{r(\pi_{j,k,+}), r(\pi_{j,k,-})\}$, denote by $\delta_{(k)}$ the k-th largest direction, and by $\pi_{j,(k),+}$ and $\pi_{j,(k),-}$ the corresponding policies.
7: Make the update step:

$$M_{j+1} = M_j + \frac{\alpha}{b\sigma_R} \sum_{k=1}^{b} \left[r(\pi_{j,(k),+}) - r(\pi_{j,(k),-}) \right] \delta_{(k)},$$

where σ_R is the standard deviation of the $2b$ rewards used in the update step.
8: **V2:** Set μ_{j+1}, Σ_{j+1} to be the mean and covariance of the $2NH(j+1)$ states encountered from the start of training.[1]
9: $j \leftarrow j + 1$
10: **end while**

Power of Linear Policy





Images from OpenAI Gym and Mania et al, 2018

Embedded Optimization as Policy

- Still have a policy with inspired structure without a closed-form solution

$$\mathbf{u}^* = \arg\min_{\mathbf{u}} \int l(\mathbf{x}(t), \mathbf{u}(t))dt$$
$$s.t. \quad \dot{\mathbf{x}}(t) = f(\mathbf{x}(t), \mathbf{u}(t))$$

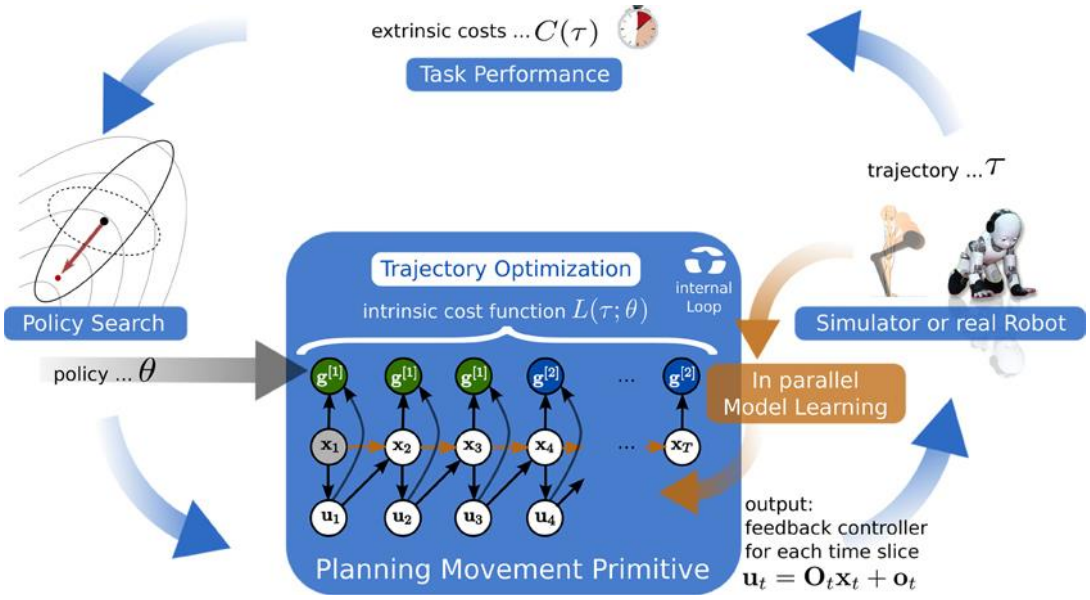
$$\mathbf{u}^* = \arg\min_{\mathbf{u}} \sum_t^T l(\mathbf{x}_t, \mathbf{u}_t)$$
$$s.t. \quad \mathbf{x}_{t+1} = f(\mathbf{x}_t, \mathbf{u}_t)$$

$$\nabla \mathbf{u}^* \text{ ?}$$

- How about differentiating the entire optimization process for policy gradient algorithms – differentiable optimization

Embedded Optimization as Policy

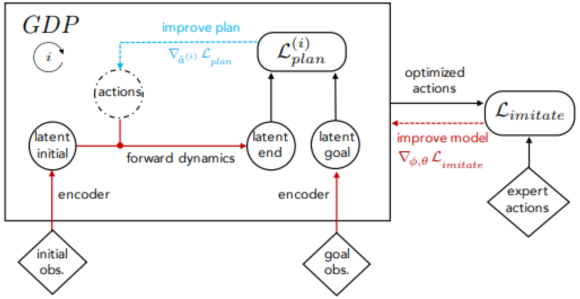
- Using internal planning as primitives
  - Quadratic intrinsic cost
  - Inference for internal TO
  - Evolution strategy for PS



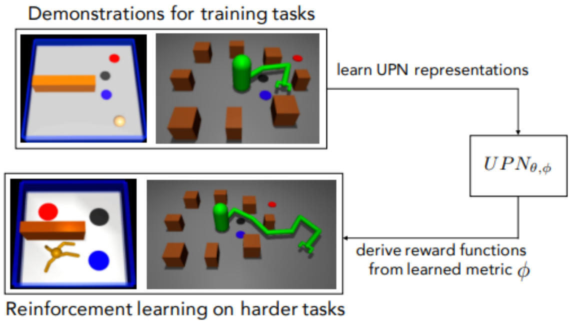
Images from Ruckert et al, 2013

Embedded Optimization as Policy

- Unrolling optimization iteration steps
  - Leveraging automatic differentiation
  - Policy learning via imitation
  - Huber loss in the latent space
  - Using learned encoders for RL



(a) Universal Planning Network (UPN)



(b) Leveraging learned latent representations

Embedded Optimization as Policy

- Avoiding backwarding iterative steps for gradients by differentiating KKT conditions

$$\mathcal{L}(\tau, \lambda) = \sum_{t=1}^{T} \left(\frac{1}{2} \tau_t^\top C_t \tau_t + c_t^\top \tau_t \right) + \sum_{t=0}^{T-1} \lambda_t^\top (F_t \tau_t + f_t - x_{t+1})$$

$$\nabla_{\tau_t} \mathcal{L}(\tau^\star, \lambda^\star) = C_t \tau_t^\star + c_t + F_t^\top \lambda_t^\star - \begin{bmatrix} \lambda_{t-1}^\star \\ 0 \end{bmatrix} = 0$$

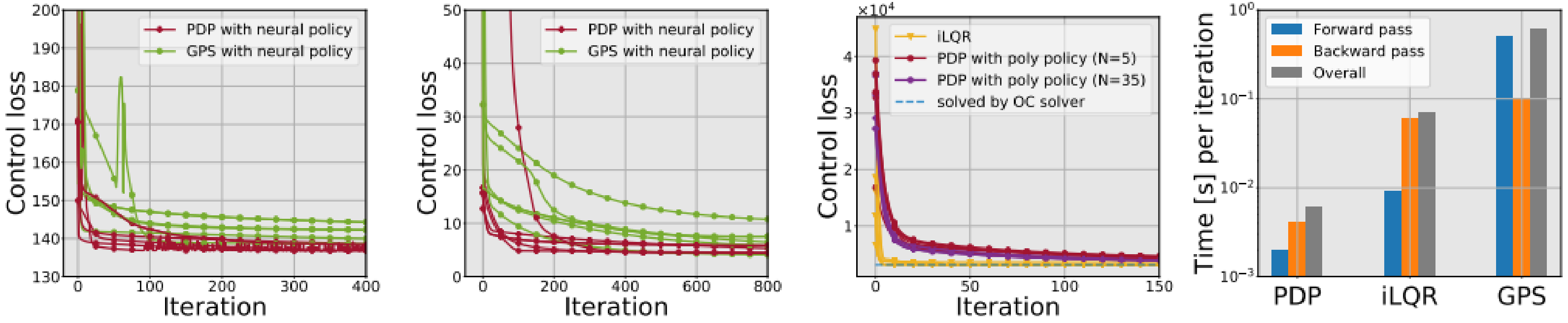
$$\nabla_{C_t} \ell = \frac{1}{2} \left(d_{\tau_t}^\star \otimes \tau_t^\star + \tau_t^\star \otimes d_{\tau_t}^\star \right) \qquad \nabla_{c_t} \ell = d_{\tau_t}^\star \qquad \nabla_{x_{\text{init}}} \ell = d_{\lambda_0}^\star$$

$$\nabla_{F_t} \ell = d_{\lambda_{t+1}}^\star \otimes \tau_t^\star + \lambda_{t+1}^\star \otimes d_{\tau_t}^\star \qquad \nabla_{f_t} \ell = d_{\lambda_t}^\star$$

- With $\tau_t = \{\mathbf{x}_t, \mathbf{u}_t\}$ $F_t = [\mathbf{A}_t, \mathbf{B}_t]$ and linear terms c_t f_t
- Construct another LQR to solve $d_{\lambda_t}^*$ $d_{\tau_t}^*$

Embedded Optimization as Policy

- Forwarding the policy still needs to iteratively
  - following gradient-based optimization (Jin et al, 2020)
  - or forming local LQ problems (Amos et al, 2018)
  - Solution must exist
  - Often requires parameterized dynamics model/trajectories



(a) Cart-pole control (b) Robot arm control (c) Quadrotor planning (d) Timing results

 Images from Jin et al, 2020

Embedded Optimization as Policy

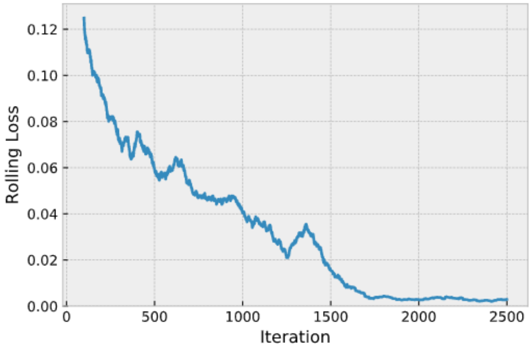
- Using convex optimization
  - Fast policy forwarding
  - No local optimality issue

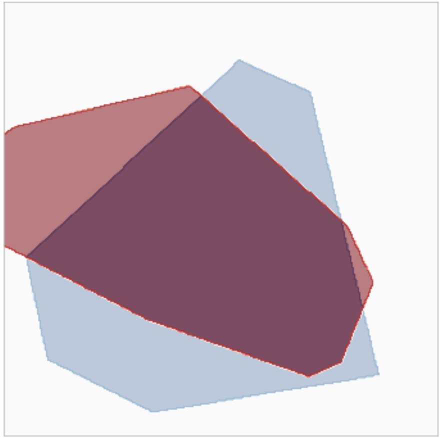
- Differentiable layers

  (https://github.com/cvxgrp/cvxpylayers)

  Generating data from

$$\hat{y} = \mathrm{argmin}_y \ \frac{1}{2}||x - y||_2^2$$
$$\mathrm{s.t.}\ \ Gy \leq h.$$

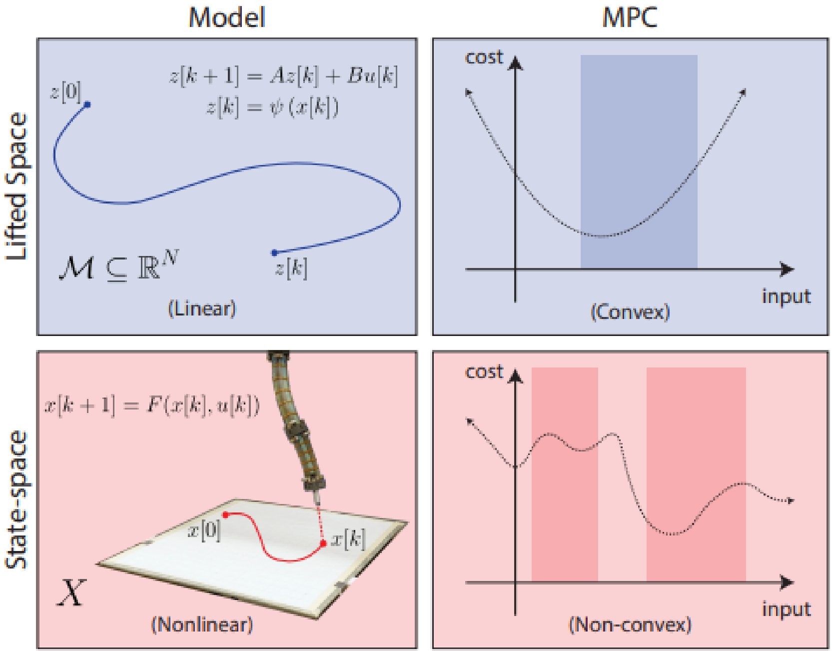




Images from Agrawal et al, 2019;

Convex Approximation – Koopman Representation

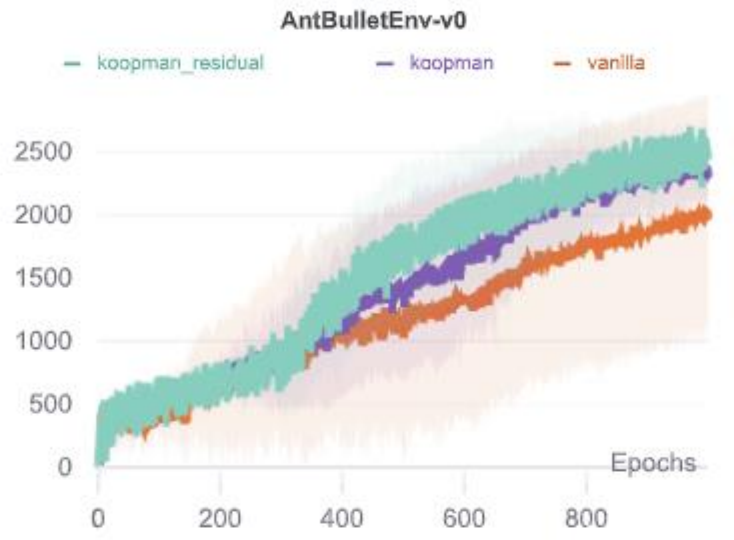
- How about non-convex dynamics constraints
- Using a high-dimensional representation such that states evolve with a linear model
- Can be LQR again using quadratic cost in the transformed representation space

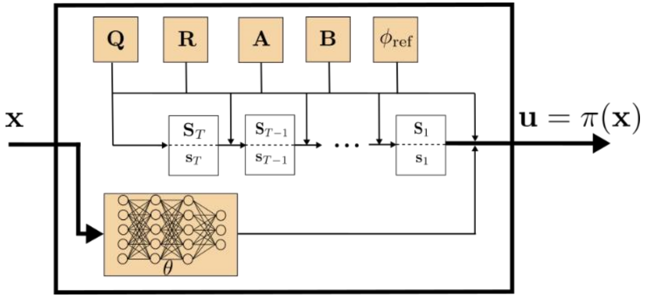


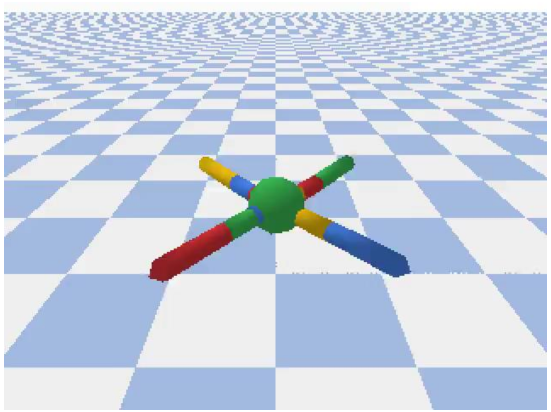
Images from Bruder et al, 2019

Convex Approximation – Koopman Representation

- Use as RL policy

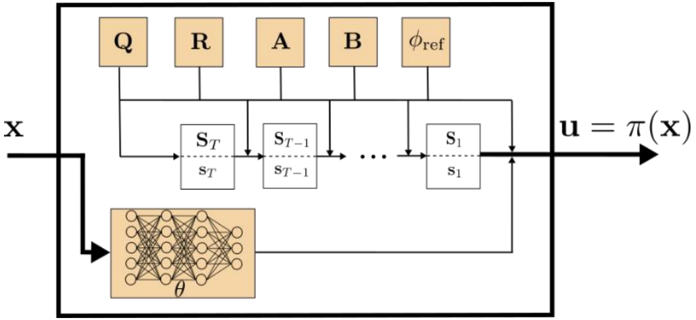


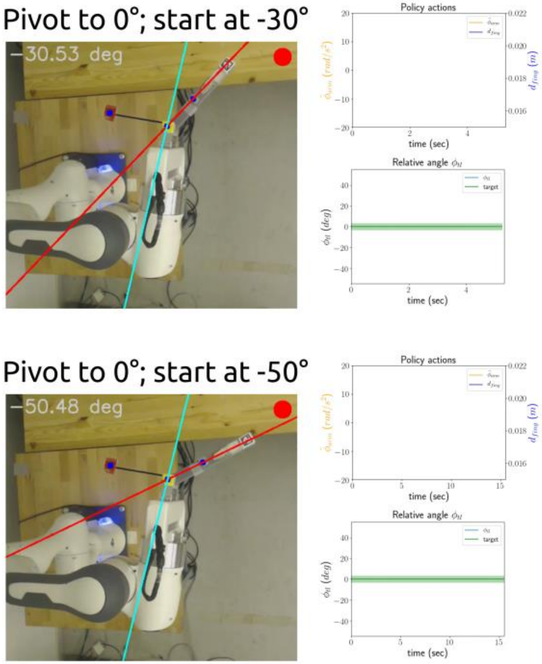




Convex Approximation – Koopman Representation

- And just neural network inference for policy evaluation





Policy with Robotic Control Structure

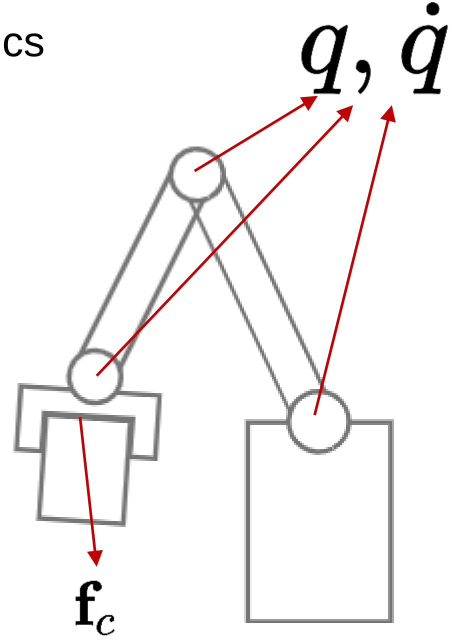
- We usually have a concrete form of manipulator dynamics

$$\dot{\mathbf{x}} = f(\mathbf{x}) + g(\mathbf{x}) \cdot \mathbf{u}$$

$$\downarrow$$

$$\mathbf{M}(q)\ddot{q} + \mathbf{b}(q, \dot{q}) + \mathbf{g}(q) = \mathbf{u} + \mathbf{J}_c(q)^\top \mathbf{f}_c$$

$$\mathbf{x} = \{q, \dot{q}\}$$

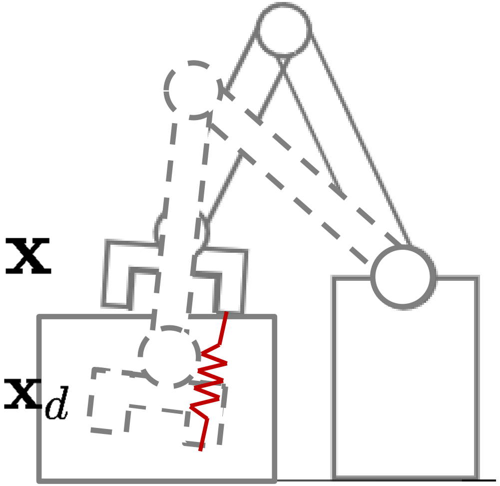


Motion Compliance

- Implicitly control of the interaction force

$$\mathbf{u} = -\mathbf{K}(\mathbf{x} - \mathbf{x}_d)$$

- Characterize robot behavior as a spring
- Policy learning is to find the right stiffness



Policy with Variable Compliance

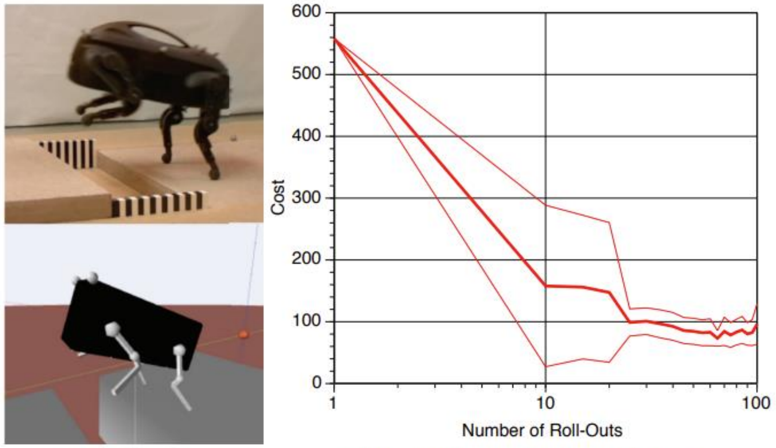
- Implicitly control of the interaction force

$$\mathbf{u} = -\mathbf{K}(\mathbf{x} - \mathbf{x}_d)$$

$$\mathbf{u}(t) = -\mathbf{K}(t)(\mathbf{x}(t) - \mathbf{x}_d(t))$$

- Parameterize compliance and desired trajectories for independent joint DOF

- Learning to jump with 100 trials



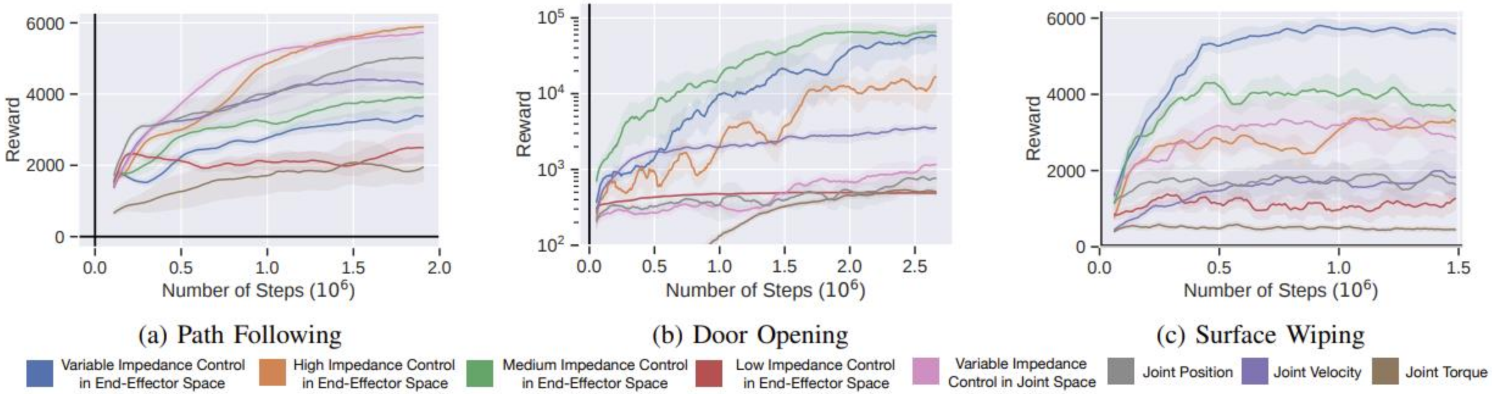
(a) Real & Simulated Robot Dog (b) Learning curve for Dog Jump with $\mathbf{PI}^2$ $\pm 1std$

Policy with Variable Compliance

- Implicitly control of the interaction force

$$\mathbf{u} = -\mathbf{K}(\mathbf{x} - \mathbf{x}_d) \longrightarrow \mathbf{u} = -\mathbf{K}(\mathbf{x})\mathbf{x}$$

- State-dependent stiffness in the joint/end-effector space
- Neural networks to output positive K



 Images from Martin-Martin et al, 2019

Control Guarantees for RL Policy

- Can we search policies that have some inherent guarantees?

$$\pi_\theta \longrightarrow \pi_{\theta'}$$

- Having some statements like $\forall \theta' \quad x(t) \in \ldots$

- Stability/Convergence/Consensus…

Lyapunov Stability

For $\mathbf{u}(\mathbf{x})$ that steers trajectory of $\mathbf{x}$ under $\dot{\mathbf{x}} = f(\mathbf{x}, \mathbf{u})$

An equilibrium $\mathbf{x}_e \quad s.t. \quad f(\mathbf{x}_e, \mathbf{u}(\mathbf{x}_e)) = 0$ is

- Stable if $\forall \epsilon > 0, \exists \delta > 0, s.t. \|\mathbf{x}(0) - \mathbf{x}_e\| < \delta \rightarrow \forall t \geq 0 \|\mathbf{x}(t) - \mathbf{x}_e\| < \epsilon$

- Asymptotic stable if … $\lim_{t \to 0} \|\mathbf{x}(t) - \mathbf{x}_e\| \rightarrow 0$

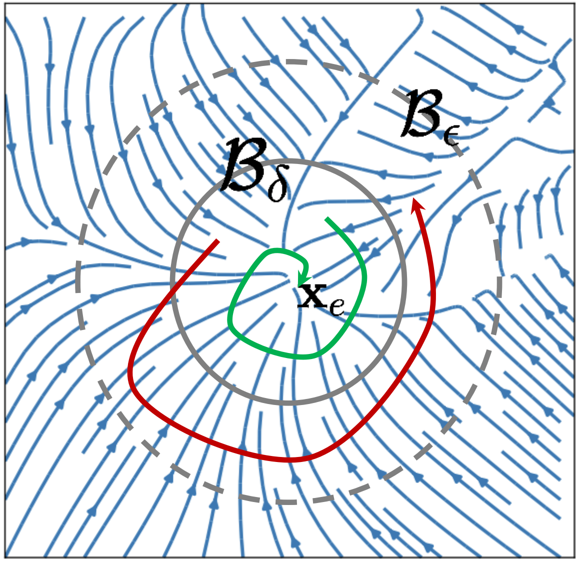
- Globally asymptotic stable (G. A. S.) if $\forall \mathbf{x}(t) \rightarrow \lim_{t \to 0} \|\mathbf{x}(t) - \mathbf{x}_e\| \rightarrow 0$

Lyapunov Stability

For $\mathbf{u}(\mathbf{x})$ that steers trajectory of $\mathbf{x}$ under $\dot{\mathbf{x}} = f(\mathbf{x}, \mathbf{u})$

An equilibrium $\mathbf{x}_e \quad s.t. \quad f(\mathbf{x}_e, \mathbf{u}(\mathbf{x}_e)) = 0$ is

- Stable
- Asymptotic stable
- Globally asymptotic stable (G. A. S.)



How to Certify That

If we can find a scalar function bounded on all sublevel sets and $V(\mathbf{x}) > 0 \quad \forall \mathbf{x} \neq \mathbf{x}_e$

An equilibrium $\mathbf{x}_e \quad s.t. \quad f(\mathbf{x}_e, \mathbf{u}(\mathbf{x}_e)) = 0$ is

- Stable if $\dot{V}(\mathbf{x}) = \nabla V(\mathbf{x}) \cdot f(\mathbf{x}, \mathbf{u}(\mathbf{x})) \leq 0 \quad \forall \mathbf{x} \in \mathcal{B}_e$

- Asymptotic stable $\dot{V}(\mathbf{x}) = \nabla V(\mathbf{x}) \cdot f(\mathbf{x}, \mathbf{u}(\mathbf{x})) < 0 \quad \forall \mathbf{x} \in \mathcal{B}_e, \mathbf{x} \neq \mathbf{x}_e, \dot{V}(\mathbf{x}_e) = 0$

- Globally asymptotic stable (G. A. S.)

$$\dot{V}(\mathbf{x}) = \nabla V(\mathbf{x}) \cdot f(\mathbf{x}, \mathbf{u}(\mathbf{x})) < 0 \quad \forall \mathbf{x} \neq \mathbf{x}_e, \dot{V}(\mathbf{x}_e) = 0$$

How to Certify That

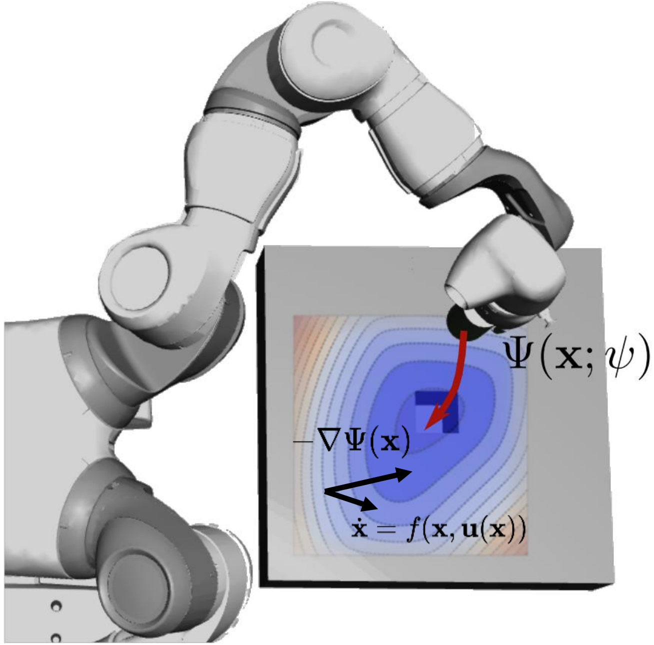
If we can find a scalar function bounded on all sublevel sets and $V(\mathbf{x}) > 0 \quad \forall \mathbf{x} \neq \mathbf{x}_e$

An equilibrium $\mathbf{x}_e \quad s.t. \quad f(\mathbf{x}_e, \mathbf{u}(\mathbf{x}_e)) = 0$ is

- Stable

- Asymptotic stable

- Globally asymptotic stable (G. A. S.)



Are General (Robotic) Controllers Stable

$$\mathbf{M}(q)\ddot{q} + \mathbf{b}(q, \dot{q}) + \mathbf{g}(q) = \mathbf{u} + \mathbf{J}_c(q)^\top \mathbf{f}_c$$

$$\mathbf{u} = -\mathbf{K}(\mathbf{x} - \mathbf{x}_d) \qquad \mathbf{K} \succ 0$$

$$\mathbf{u}(t) = -\mathbf{K}(t)(\mathbf{x}(t) - \mathbf{x}_d(t))$$

$$\mathbf{u} = -\mathbf{K}(\mathbf{x})\mathbf{x}$$

$$\mathbf{u}^* = \arg\min_{\mathbf{u}} [l(\mathbf{x}, \mathbf{u}) + \frac{\partial V^*(\mathbf{x})}{\partial \mathbf{x}} \cdot f(\mathbf{x}, \mathbf{u})]$$

$$0 = \min_{\mathbf{u}} [l(\mathbf{x}, \mathbf{u}) + \underbrace{\boxed{\frac{\partial V^*(\mathbf{x})}{\partial \mathbf{x}} \cdot f(\mathbf{x}, \mathbf{u})}}_{\dot{V}(\mathbf{x})}] \qquad l(\mathbf{x}, \mathbf{u}) > 0$$

Parameterize policy with Energy-Shaping Controllers

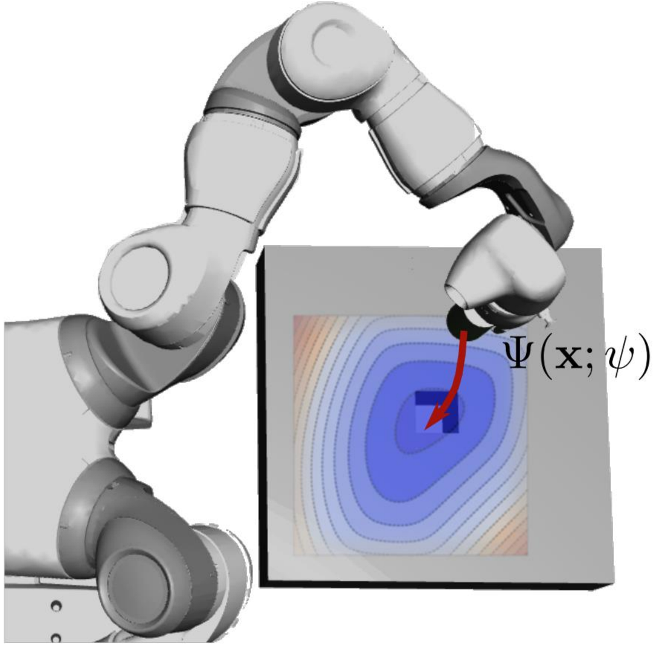
$$\mathbf{M}(q)\ddot{q} + \mathbf{b}(q, \dot{q}) + \mathbf{g}(q) = \mathbf{u} + \mathbf{J}_c(q)^\top \mathbf{f}_c$$

$$\mathbf{u} = -\frac{\partial}{\partial q} V(q) - \mathbf{D}(\dot{\mathbf{q}})\dot{q}$$

For convex $V(\mathbf{x})$ with a unique minimum

at q^* , q^* is G. A. S.

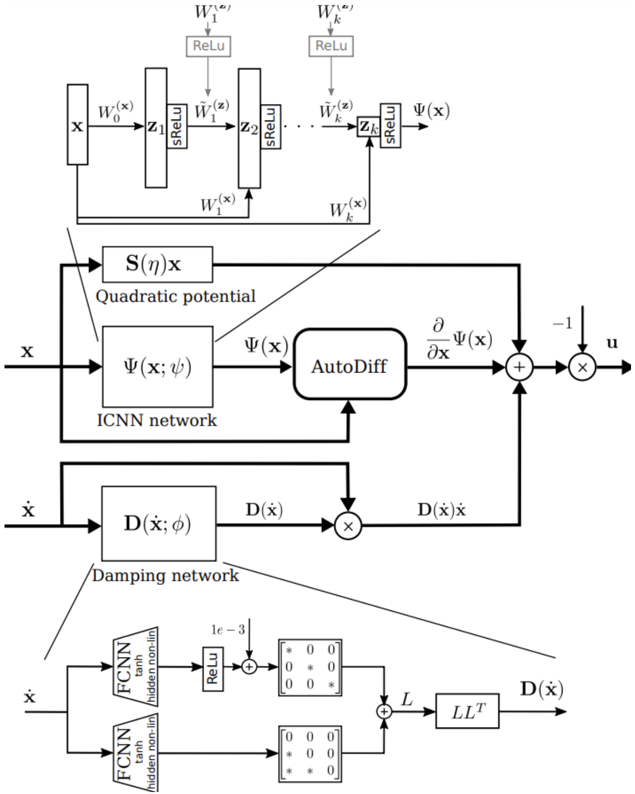
Policy biases a goal-directed behaviour



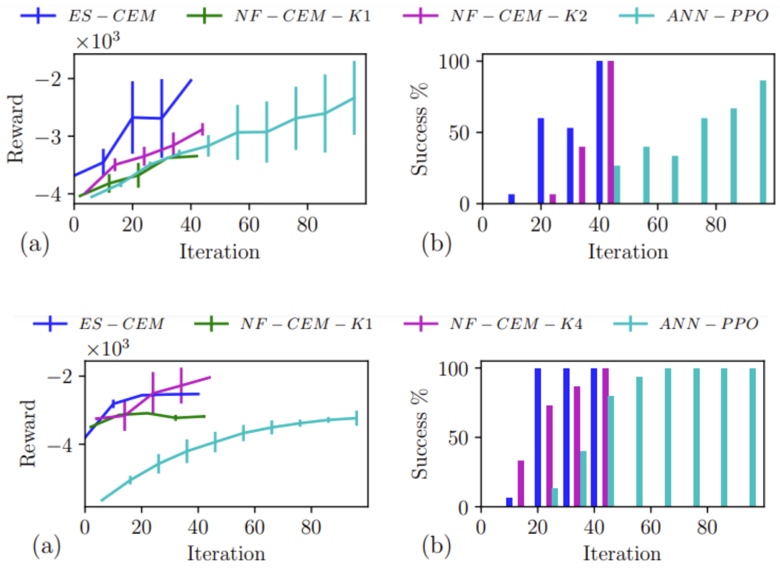
Parameterize policy with Energy-Shaping Controllers

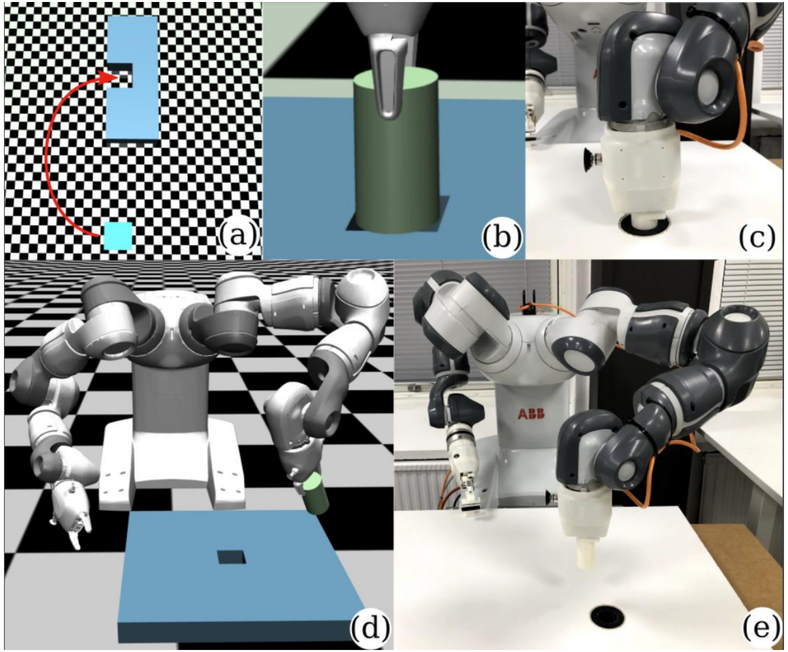
- How can we get a convex (neural network) energy function

- Input Convex Neural Network (ICNN, Amos et al, 2017)
  - Additive quadratic for strong convexity
  - Zero bias for unique minimum at origin



How Does It Work





How Does It Work

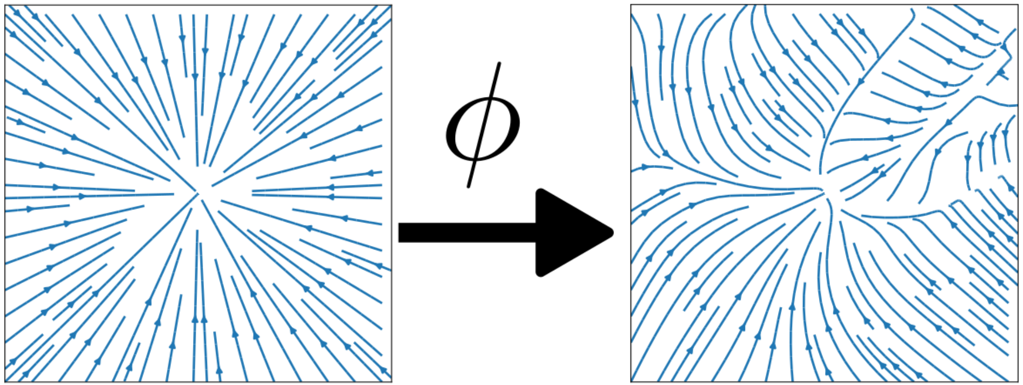
Reinforcement learning of the 2D block-insertion task

Policy: The proposed energy shaping policy
Algorithm: Cross Entropy Method

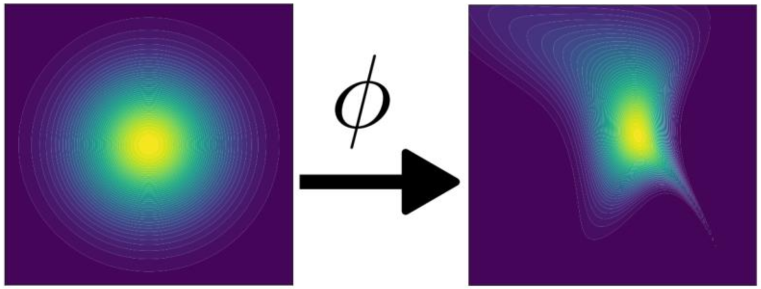
Another Way to Gain Stability Guarantees

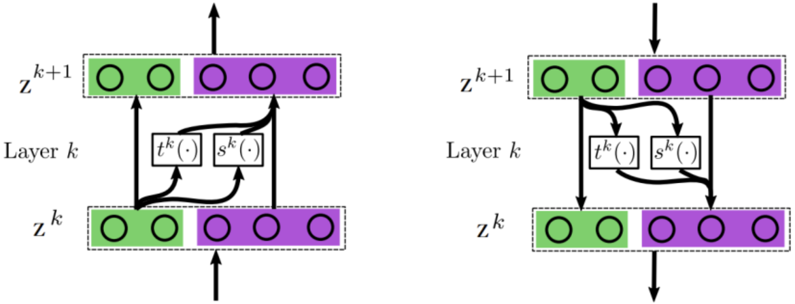
- Lyapunov analysis is usually easy for linear control & systems
  - LQR – Quadratic function $V^*(\mathbf{x}) = \mathbf{x}^\top \mathbf{P}\mathbf{x}$

- Can we construct policy parameterization by “non-linearizing” a stable linear control/system, with guarantees reserved?



Diffeomorphism via Normalizing-Flow Models





- Construct non-Gaussian models from Gaussians

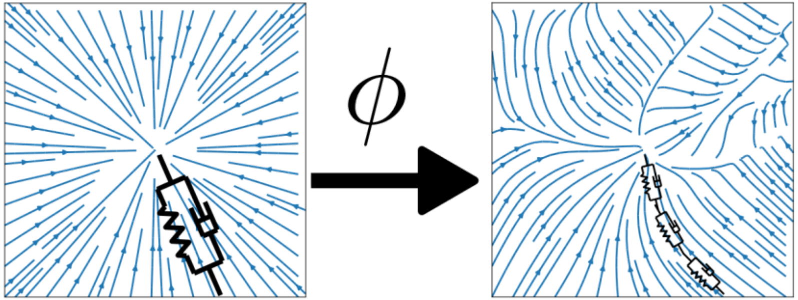
$$p(\mathbf{x}) = |\mathbf{J}_\phi(\mathbf{z})|^{-1} p(\mathbf{z}|\mu, \Sigma)$$

- Differentiable and invertible neural networks – Real NVP (Dinh et al , 2016)

- Affine composition

$$\mathbf{z}^{k+1} = \phi^k(\mathbf{z}^k)$$
$$\mathbf{z}^{k+1}_{1:d^k} = \mathbf{z}^k_{1:d^k}$$
$$\mathbf{z}^{k+1}_{d^k+1:d} = \mathbf{z}^k_{d^k+1:d} \odot \exp(\mathbf{s}^k(\mathbf{z}^k_{1:d^k})) + \mathbf{t}^k(\mathbf{z}^k_{1:d^k})$$

Constructing Stable Normalizing-Flow Controller



$$\mathbf{u} = -\mathbf{S}\mathbf{x} - \mathbf{D}\dot{\mathbf{x}}$$

$$\mathbf{u} = \mathbf{g}(\mathbf{x}) - \mathbf{D}\dot{\mathbf{x}} - \mathbf{J}_\phi(\mathbf{x})^\top \mathbf{S}[\phi(\mathbf{x}) - \phi(\mathbf{x}_{\text{ref}})]$$

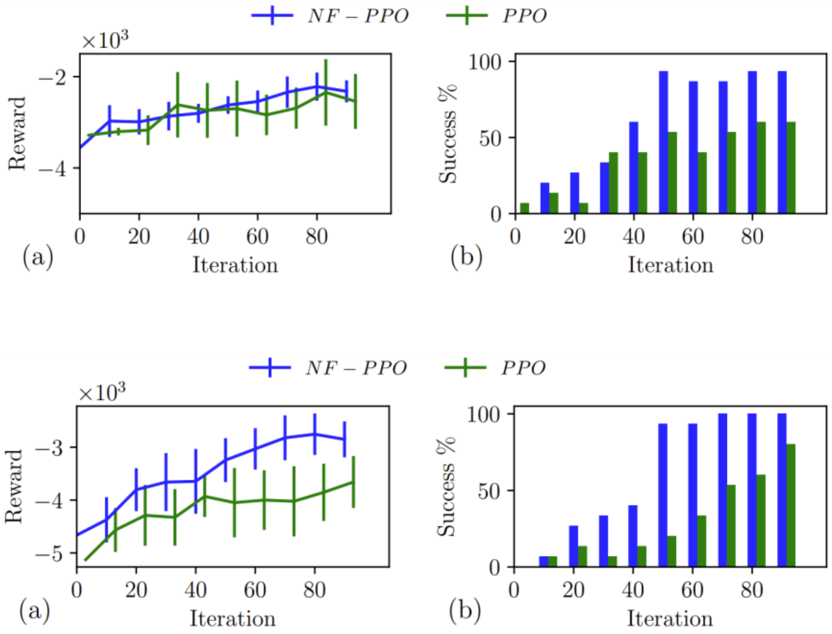
- Neural network parameterized
- G. A. S. of $\mathbf{x}_{\text{ref}}$ under

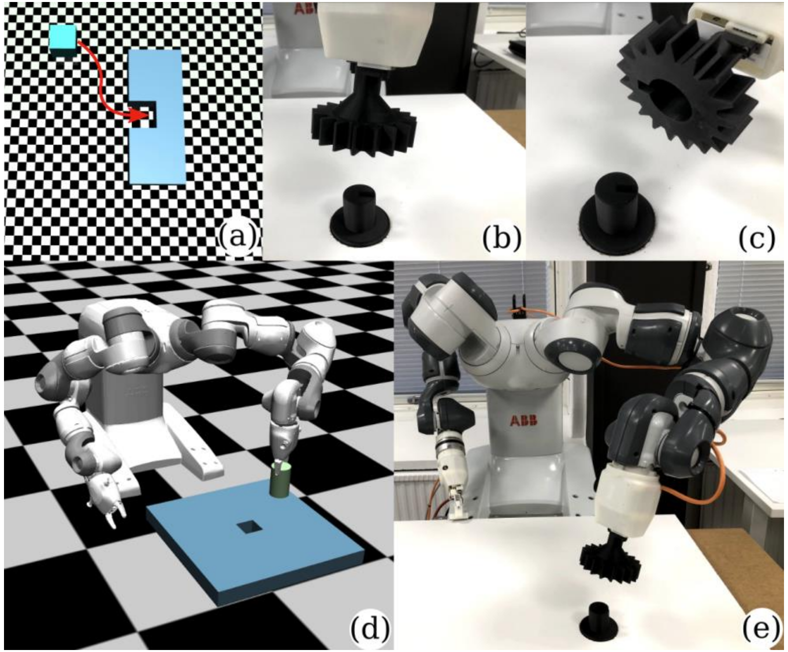
$$\mathbf{M}(q)\ddot{q} + \mathbf{b}(q,\dot{q}) + \mathbf{g}(q) = \mathbf{u} + \mathbf{J}_c(q)^\top \mathbf{f}_c$$

- Stable variable impedance controller comparing to Martin-Martin, 2019

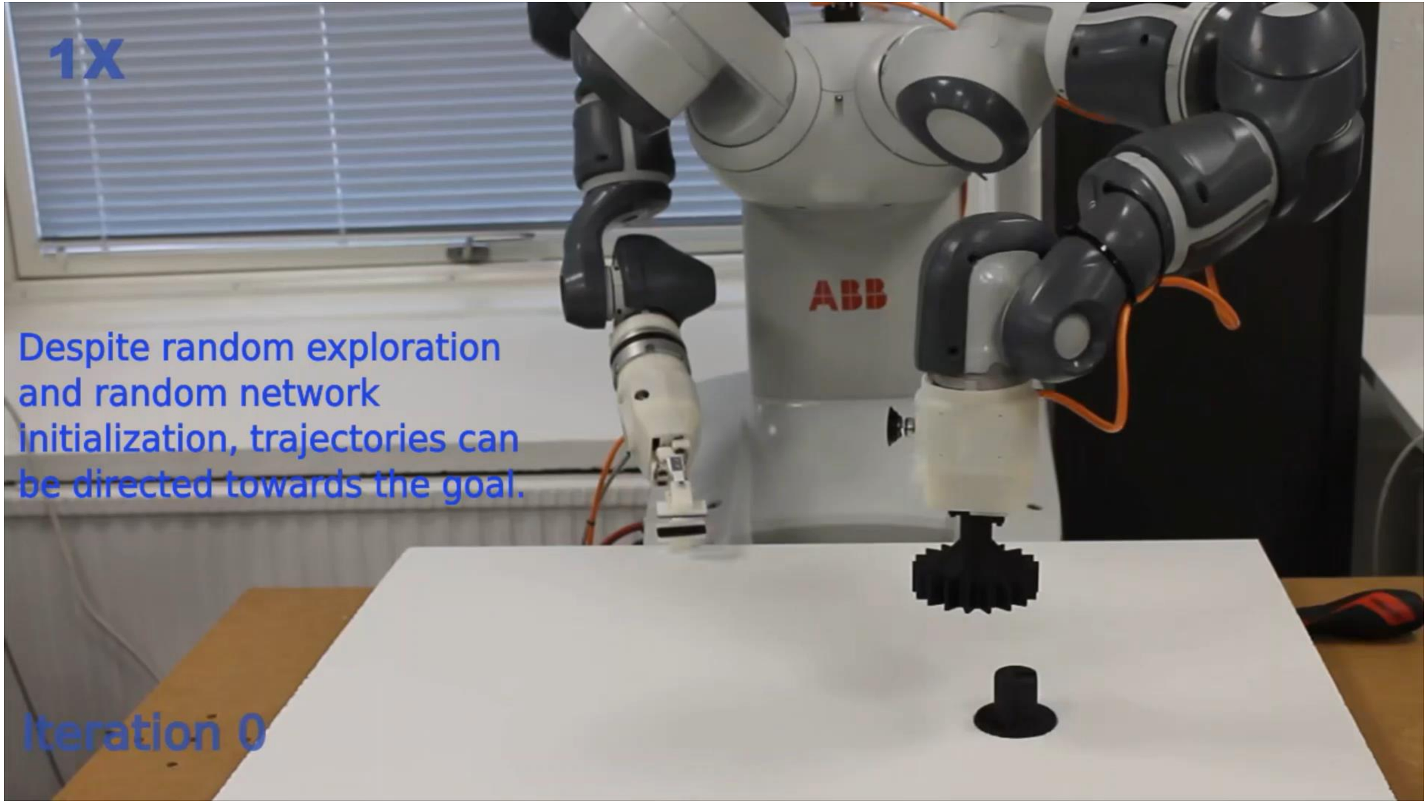
$$\mathbf{u} = -\mathbf{K}(\mathbf{x})\mathbf{x}$$

How Does It Work





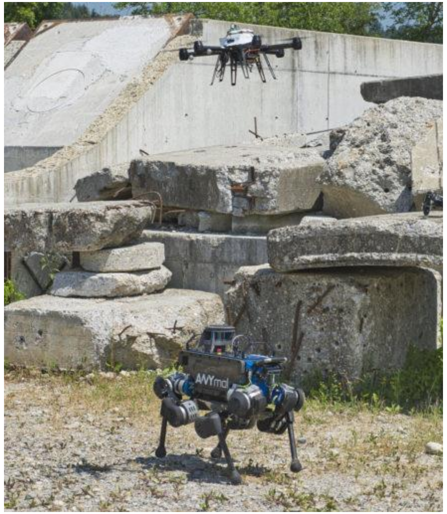
How Does It Work

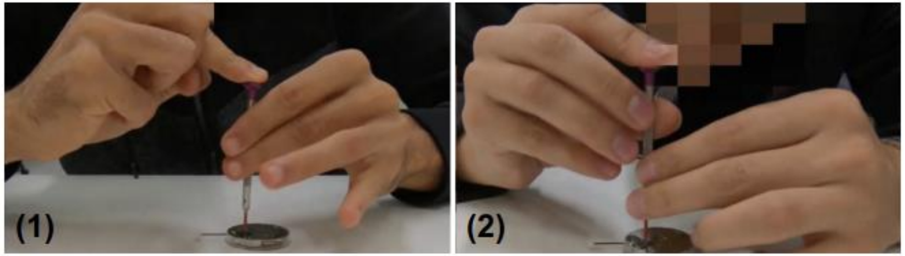


From Single Robot to Multiple Robots

- More dexterity and capacity
  - Heterogenous robot teams
  - Anthropomorphic robots

- Larger and diverse action/policy space

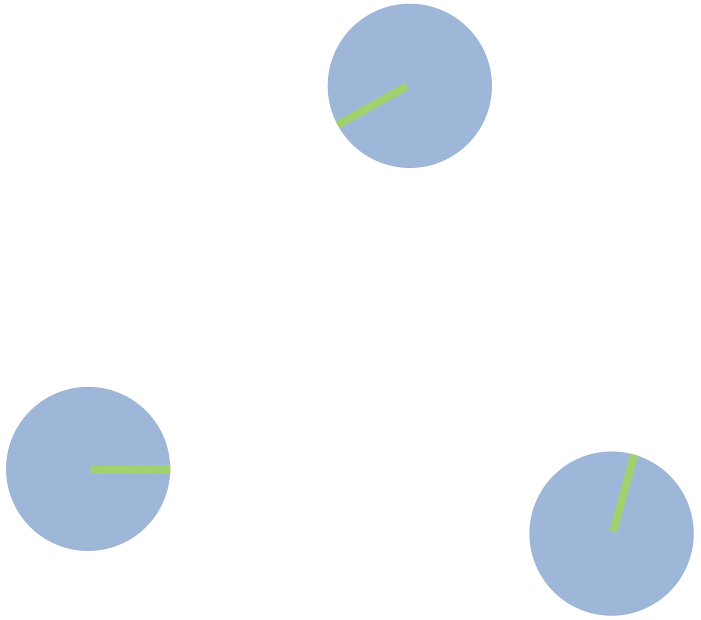




Images from NCCR Robotics and Yao et al, 2021

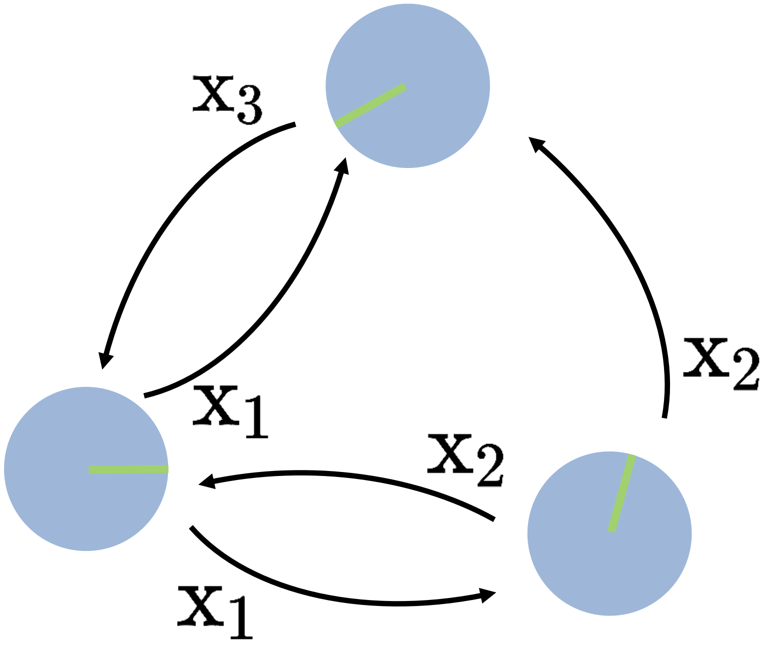
Consensus in Multi-Robot Systems

- N robots



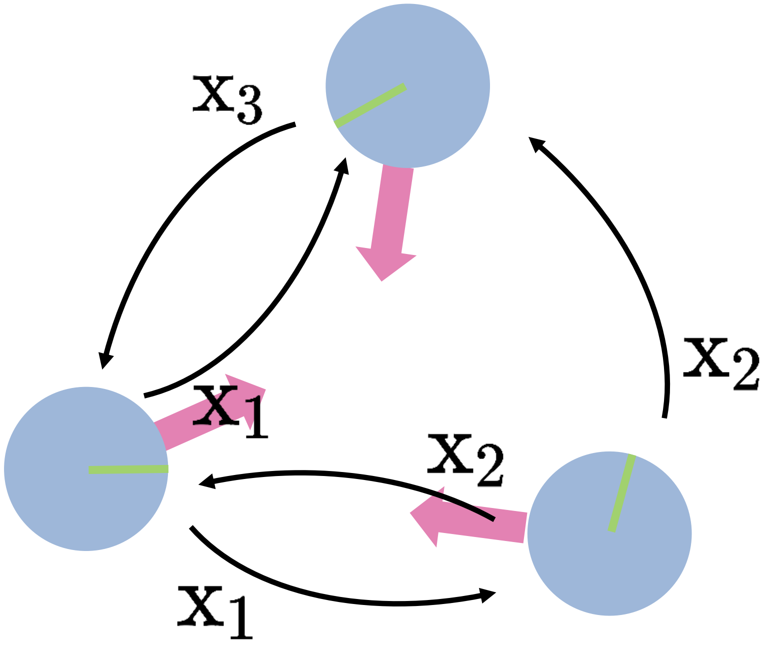
Consensus in Multi-Robot Systems

- N robots
  - Communicate with neighbouring robots



Consensus in Multi-Robot Systems

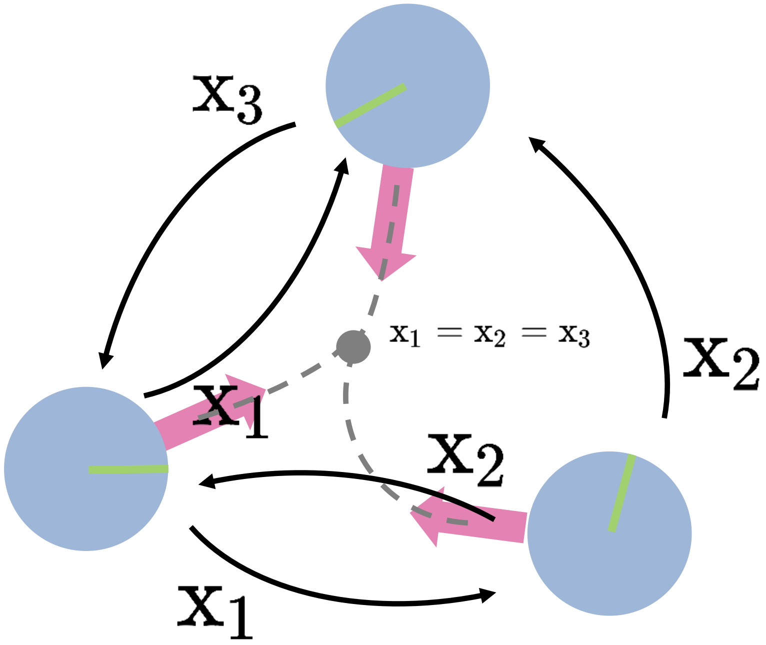
- N robots
  - Communicate with neighbouring robots
  - Apply distributed control



Consensus in Multi-Robot Systems

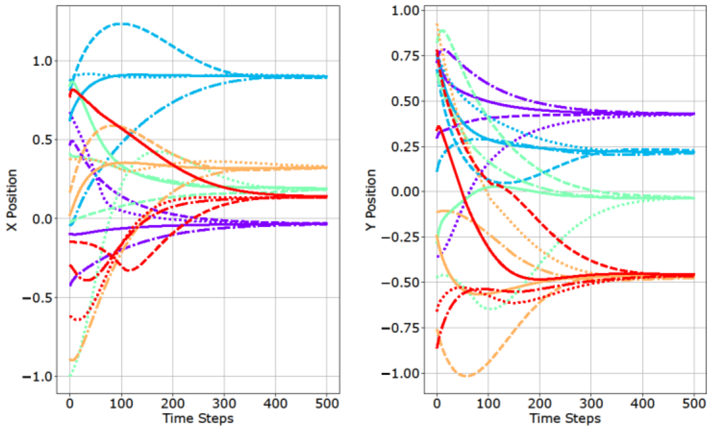
- N robots
  - Communicate with neighbouring robots
  - Apply distributed control
  - Reach agreement on communicated information

e.g. $\dot{\mathbf{x}} = \mathbf{L}\mathbf{x}$



Consensus-based Normalizing-Flow Control

- Nonlinear transient behavior
  - Allowing to learn complex rendez-vous trajectories



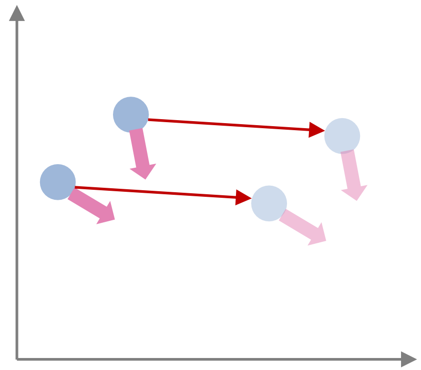
- Apply normalizing-flow to consensus problem: “non-linearize” basic protocol
  - N robots with 2nd-order dynamics
  - Global consensus stabilization

$$\bar{\mathbf{u}} = \mathbf{g}(\mathbf{x}) - \mathbf{D}\dot{\mathbf{x}} - \bar{\mathbf{J}}(\mathbf{x})^\top (\mathbf{L} \otimes I_n)\bar{\phi}(\mathbf{x})$$

$$\mathbf{x} = [\mathbf{x}_1^\top, \ldots, \mathbf{x}_n^\top]^\top \in \mathbb{R}^{nd}$$

Symmetry in Consensus

- Translation-invariant
  - Control remains when all robots are translated by a same vector
  - Keep a consistent behavior across entire workspace



$$\dot{\mathbf{x}} = \mathbf{L}\mathbf{x}$$

$$\bar{\mathbf{u}} = \mathbf{g}(\mathbf{x}) - \mathbf{D}\dot{\mathbf{x}} - \bar{\mathbf{J}}(\mathbf{x})^\top (\mathbf{L} \otimes I_n)\bar{\phi}(\mathbf{x})$$

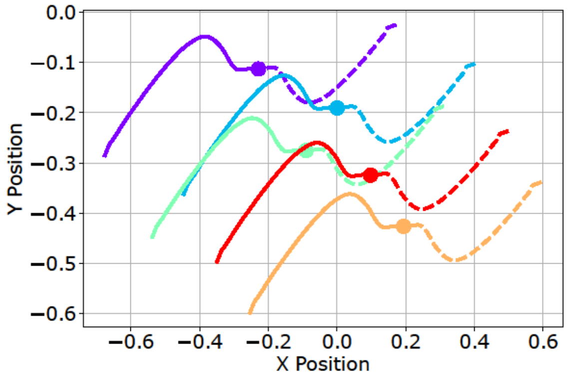
Translation-invariant Normalizing-Flow Control

- N=2 robots, e.g. dual-arm
- Global consensus stablization

$$\tau = \mathbf{g}(\mathbf{x}) - \mathbf{D}\dot{\mathbf{x}} - \bar{\mathbf{L}}\bar{\mathbf{J}}(\bar{\mathbf{L}}\mathbf{x})^\top \bar{\mathbf{S}}\bar{\phi}(\bar{\mathbf{L}}\mathbf{x})$$

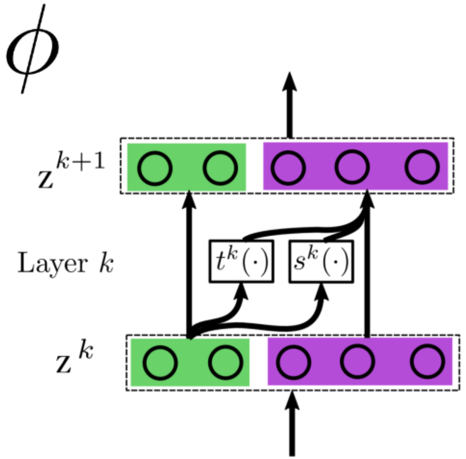
- Need antipodal-equivariant normalizing-flow

$$\phi(*) = -\phi(-*)$$



Antipodal-Equivariant Normalizing-Flows

$$\mathbf{z}^{k+1} = \phi^k(\mathbf{z}^k)$$
$$\mathbf{z}^{k+1}_{1:d^k} = \mathbf{z}^k_{1:d^k}$$
$$\mathbf{z}^{k+1}_{d^k+1:d} = \mathbf{z}^k_{d^k+1:d} \odot \exp(\mathbf{s}^k(\mathbf{z}^k_{1:d^k})) + \mathbf{t}^k(\mathbf{z}^k_{1:d^k})$$

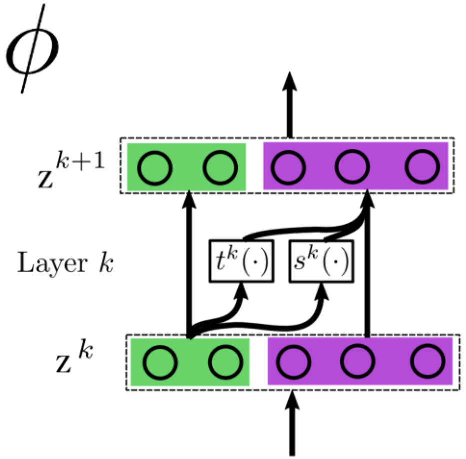


Antipodal-Equivariant Normalizing-Flows

$$\mathbf{z}^{k+1} = \phi^k(\mathbf{z}^k)$$

$$\mathbf{z}^{k+1}_{1:d^k} = \boxed{\mathbf{z}^k_{1:d^k}}$$

$$\mathbf{z}^{k+1}_{d^k+1:d} = \boxed{\mathbf{z}^k_{d^k+1:d}} \odot \exp\left(\mathbf{s}^k(\mathbf{z}^k_{1:d^k})\right) + \mathbf{t}^k(\mathbf{z}^k_{1:d^k})$$



Equivariant

Need to be equivariant

Need to be invariant

Antipodal-Equivariant Normalizing-Flows

$$\mathbf{z}^{k+1} = \phi^k(\mathbf{z}^k)$$
$$\mathbf{z}^{k+1}_{1:d^k} = \boxed{\mathbf{z}^k_{1:d^k}}$$
$$\mathbf{z}^{k+1}_{d^k+1:d} = \boxed{\mathbf{z}^k_{d^k+1:d}} \odot \exp\left(\boxed{\mathbf{s}^k(\mathbf{z}^k_{1:d^k})}\right) + \boxed{\mathbf{t}^k(\mathbf{z}^k_{1:d^k})}$$

Equivariant

Need to be equivariant

Need to be invariant

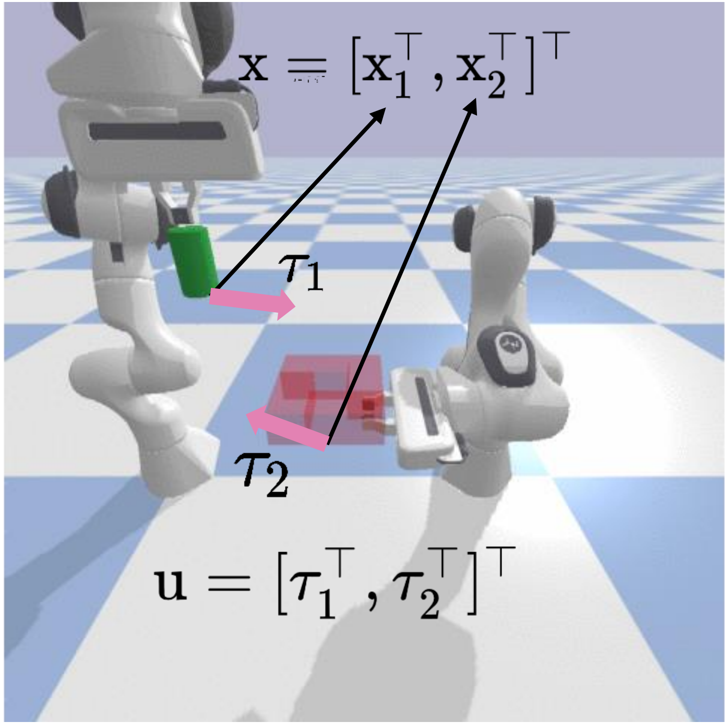
Construction with even and odd functions

$$\tilde{\mathrm{s}}_k(*) = \mathrm{s}_k(*) + \mathrm{s}_k(-*)$$

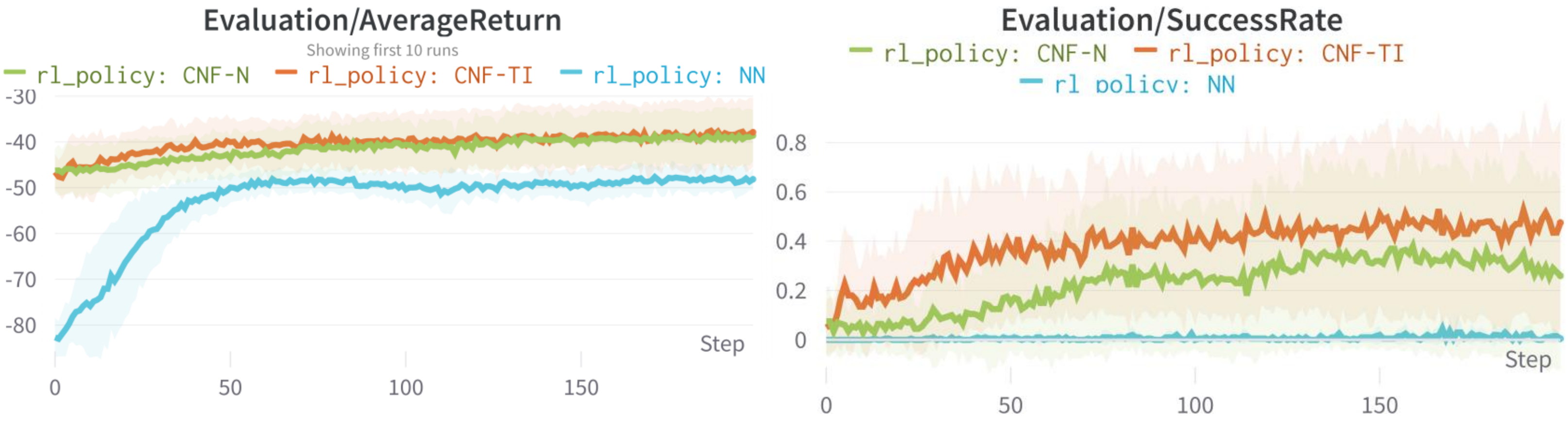
$$\tilde{\mathrm{t}}_k(*) = \mathrm{t}_k(*) - \mathrm{t}_k(-*)$$

How Does It Work

- Bimanual insertion with a clearance of 2mm
  - Observation/Action: translational position and force
  - Fixed stiffness controller for end-effector orientations
- PPO
  - Stochastic policy with controller adding Gaussian noise
  - 1e-4 learning rate
  - 10^6 environment steps
  - Randomized initial positions 10cmx10cm area
  - 10 random seeds

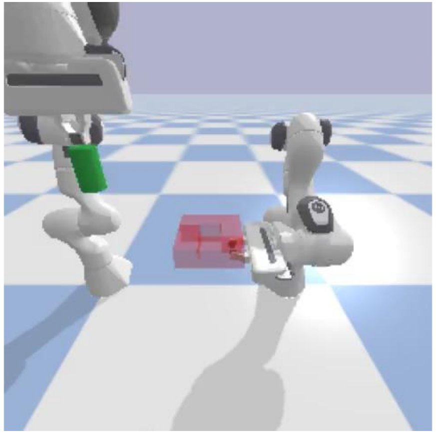


How Does It Work- Training Curves

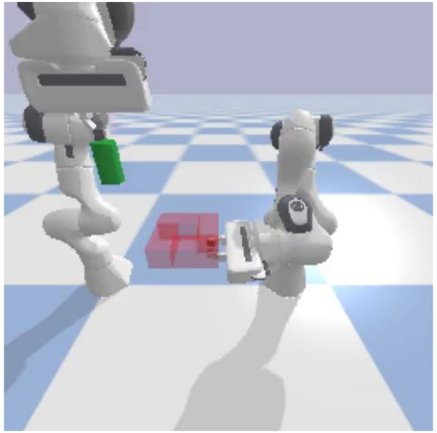


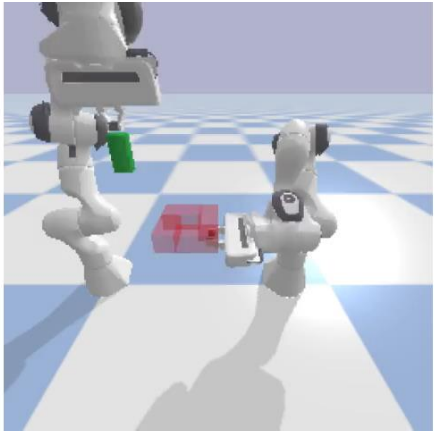
How Does It Work - Results

Basic consensus-based policy (CNF-N)



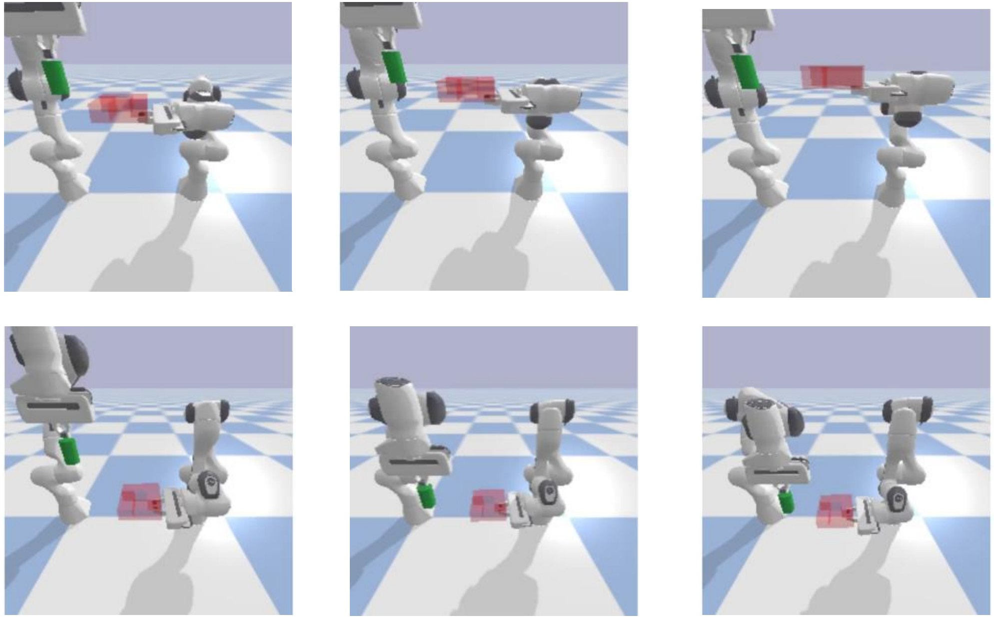
Translation-invariant policy (CNF-TI)



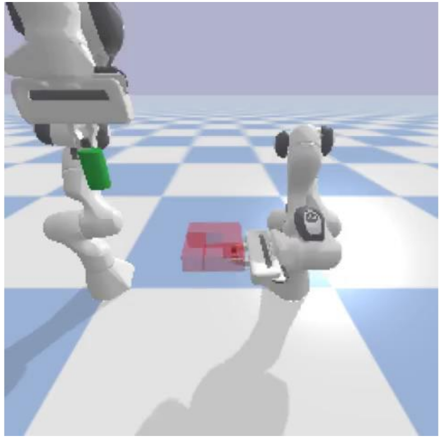


How Does It Work - Generalization and Failure Case

Generalize to novel positions



Failure case



Closing Notes

- RL (real) data efficiency gained from prior knowledge
  - In form of synthetic simulation data: zero/few-shot sim-to-real
  - In form of structured policy design
- Choosing “right” policy
  - Optimal control perspectives
  - Structure from robotics literature
  - Consideration on Guarantees
  - Train on real robots
- Design RL system with provable performance guarantees
  - Computational tractability
  - Efficiency from regulated searching space
  - Theory-grounded and certified algorithms

Closing Notes

- Connection to Chris and Alexis lectures
  - RL exploration with provable properties
  - Stability does not necessarily mean safety
  - Mostly in robot control application: continuous, steady, constraints
  - Need more powerful description of desired system behaviours

Supplemental Materials

- Benjamin Recht – An outsider's tour of reinforcement learning
  - http://www.argmin.net/2018/06/25/outsider-rl/

- Underactuated Robotics
  - Course notes (https://underactuated.mit.edu/)

- Dimitri Bertserkas
  - Reinforcement Learning and Optimal Control (http://web.mit.edu/dimitrib/www/RLbook.html)
  - Lessons from AlphaZero for Optimal, Model Predictive, and Adaptive Control (http://web.mit.edu/dimitrib/www/abstractdp_MIT.html)