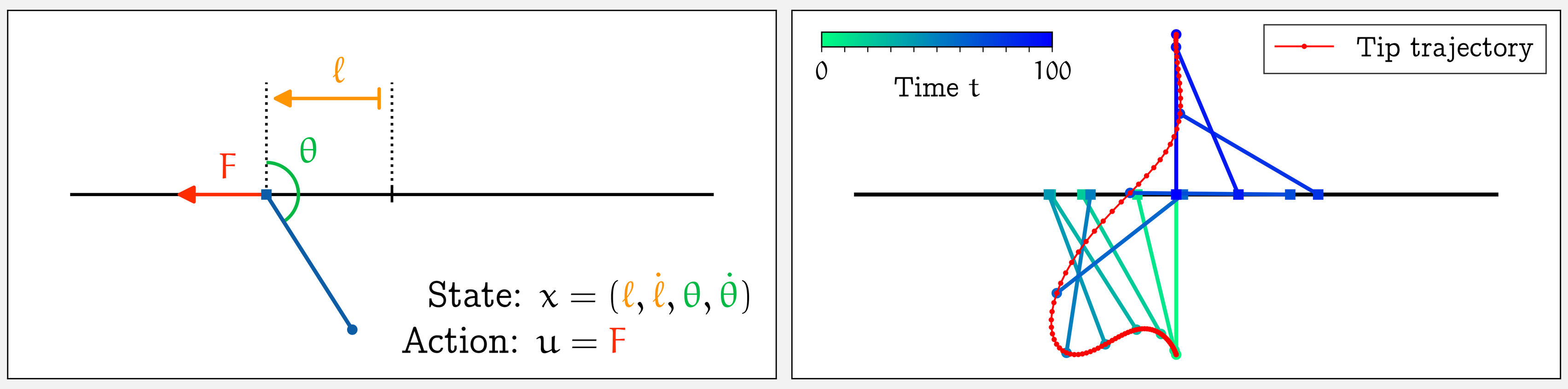
A Pontryagin Perspective on Reinforcement Learning

Onno Eberhard[1,2] Claire Vernade[2] Michael Muehlebach[1]

[1]Max Planck Institute for Intelligent Systems [2]University of Tübingen

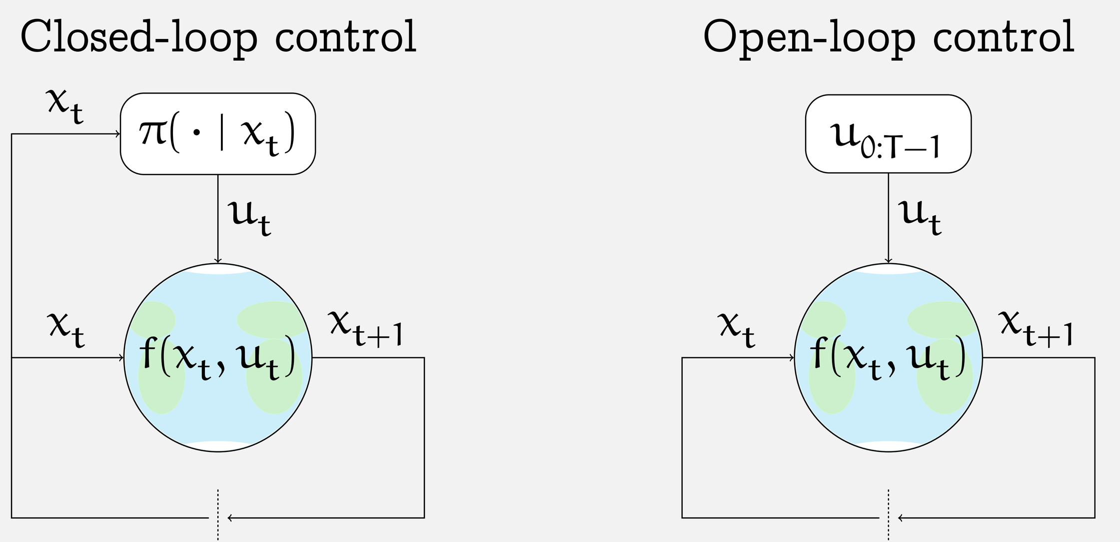
We introduce *open-loop reinforcement learning* by replacing Bellman with Pontryagin

Motivation



- Some behavior is best represented as a sequence of actions, *not as a policy*
- Open-loop methods are commonplace in control, but largely ignored in RL
- In applications where sensors are not viable, an open-loop solution is required

Open-loop control



- Closed-loop control: learn a policy π that maximizes the sum of rewards

$$\pi^\star = \operatorname*{arg\,max}_{\pi : \mathcal{X} \to \Delta_{\mathcal{U}}} \mathbb{E}_\pi \left[\sum_{t=0}^{T-1} r(x_t, u_t) + r_T(x_T) \right]$$

- Open-loop control: learn a sequence of actions instead of a policy

$$u^\star_{0:T-1} = \operatorname*{arg\,max}_{u_{0:T-1} \in \mathcal{U}^T} \underbrace{\sum_{t=0}^{T-1} r(x_t, u_t) + r_T(x_T)}_{J(u_{0:T-1})} \quad \text{s.t.} \quad x_{t+1} = f(x_t, u_t)$$

- The open-loop problem is often much easier (optimize over $\mathcal{U}^T$ instead of $\Delta_{\mathcal{U}}^{\mathcal{X}}$)
- We can optimize J with gradient ascent and *Pontryagin's principle*

Pontryagin's principle for computing ∇J

1. Forward pass: $x_{t+1} = f(x_t, u_t)$, where x_0 is given
2. Backward pass: $\lambda_t = \nabla_x r(x_t, u_t) + \nabla_x f(x_t, u_t)\lambda_{t+1}$, where $\lambda_T = \nabla r_T(x_T)$
3. Gradient: $\nabla_{u_t} J(u_{0:T-1}) = \nabla_u r(x_t, u_t) + \nabla_u f(x_t, u_t)\lambda_{t+1}$

Open-loop reinforcement learning

- In RL, we don't know the dynamics f, but Pontryagin requires $\nabla_x f_t$ and $\nabla_u f_t$

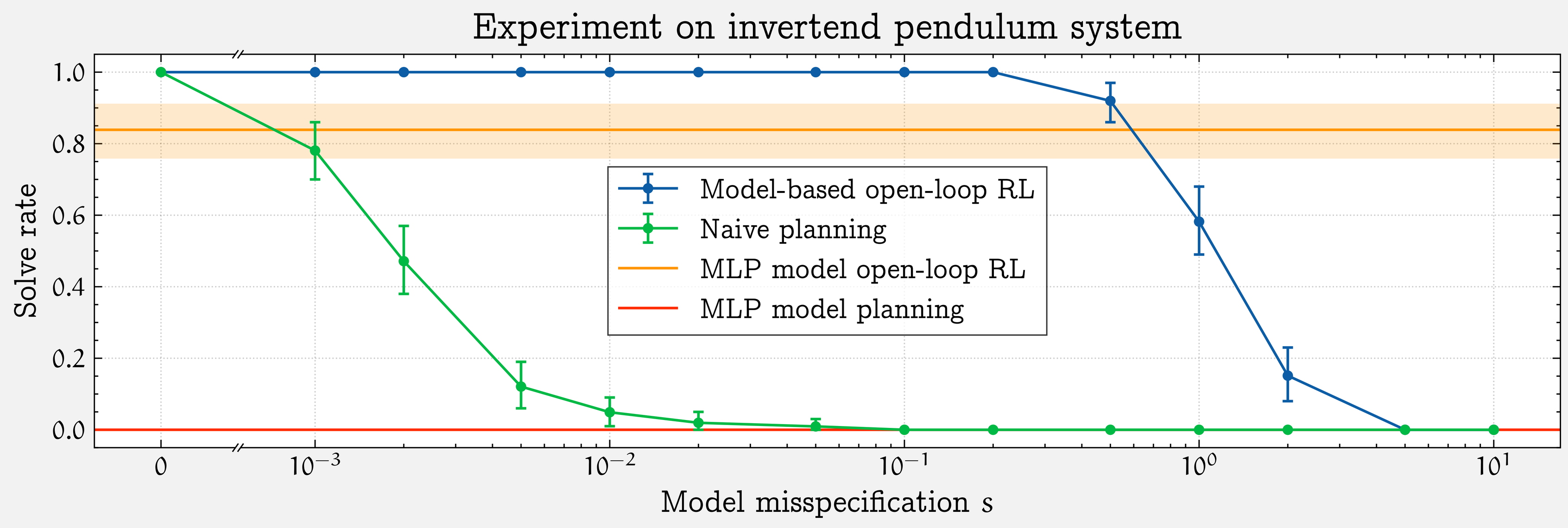
Theorem (informal)

Replace $\nabla_x f_t$ and $\nabla_u f_t$ in Pontryagin's equations by estimates A_t and B_t with sufficiently small errors $\|A_t - \nabla_x f_t\|$ and $\|B_t - \nabla_u f_t\|$ to get an approximate gradient $g \simeq \nabla J(u_{0:T-1})$. Then, gradient ascent on g produces iterates $u^{(0)}_{0:T-1}, \dots, u^{(N-1)}_{0:T-1}$ that satify, for some learning rate η and constant $\alpha > 0$,

$$\frac{1}{N}\sum_{k=0}^{N-1} \|\nabla_{u_t} J(u^{(k)}_{0:T-1})\|^2 \leqslant \frac{J^\star - J(u^{(0)}_{0:T-1})}{\alpha \eta N}.$$

Model-based open-loop RL

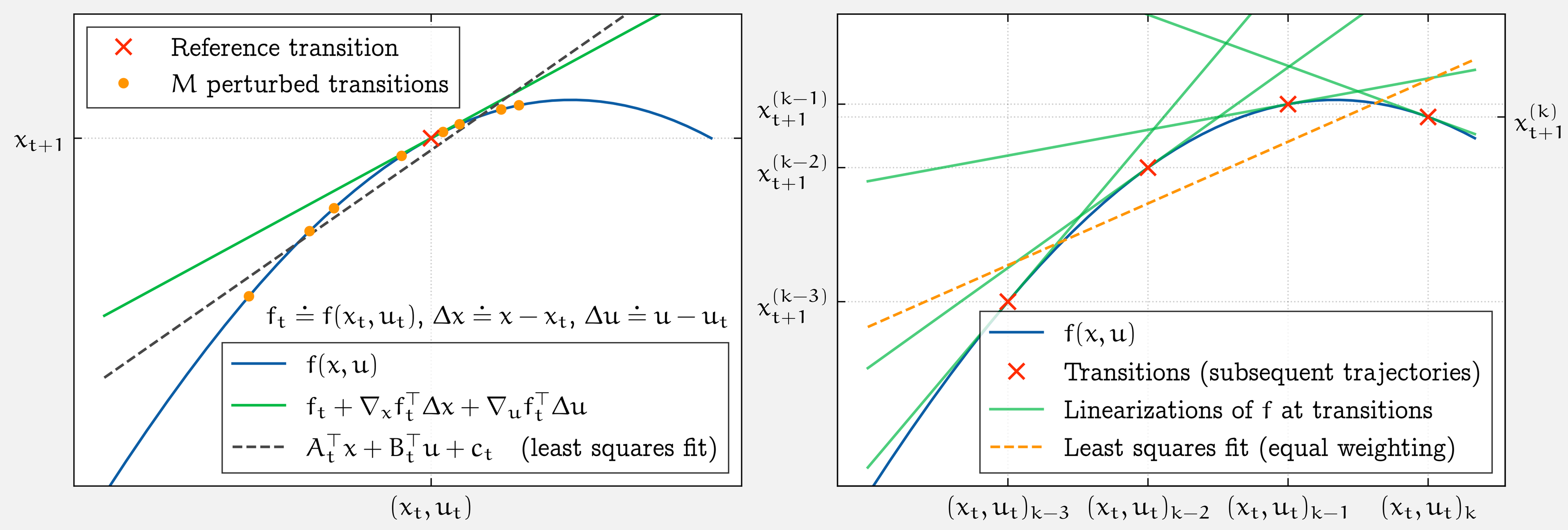
- We can learn a dynamics model $\tilde{f} \simeq f$ and set $A_t \doteq \nabla_x \tilde{f}_t$ and $B_t \doteq \nabla_u \tilde{f}_t$
- This method is remarkably robust against modeling errors



Model-free open-loop RL

- The Jacobians $\nabla_x f_t$ and $\nabla_u f_t$ measure how x_{t+1} changes if (x_t, u_t) is perturbed
- We can estimate them directly from M rollouts with perturbed actions:

$$\operatorname*{arg\,min}_{[A_t^\top \; B_t^\top \; c_t] \in \mathbb{R}^{D \times (D+K+1)}} \sum_{i=1}^{M} \|A_t^\top x_t^{(i)} + B_t^\top u_t^{(i)} + c_t - x_{t+1}^{(i)}\|^2$$



- This an *on-trajectory* method: data is discarded after each update

Off-trajectory open-loop RL

- If subsequent trajectories are similar, we can reuse previous Jacobian estimates
- We can solve the regression problem with *recursive least squares*:

$$Q_t^{(k)} = \alpha Q_t^{(k-1)} + (1-\alpha) q_0 I + z_t^{(k)} \{z_t^{(k)}\}^\top$$

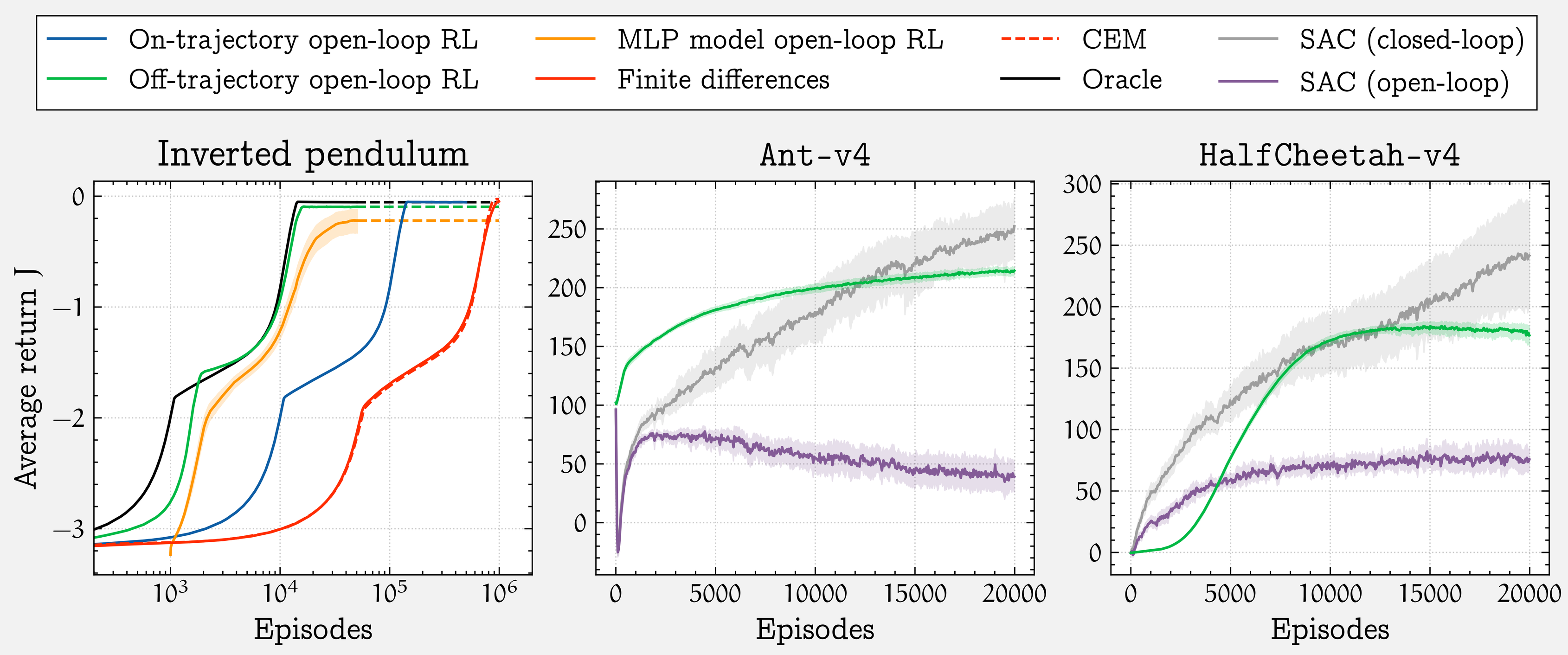
$$F_t^{(k)} = F_t^{(k-1)} + \{Q_t^{(k)}\}^{-1} z_t^{(k)} \{x_{t+1}^{(k)} - F_t^{(k-1)} z_t^{(k)}\}^\top,$$

where $F_t \doteq [A_t^\top \; B_t^\top \; c_t]$, $z_t \doteq (x_t, u_t, 1) \in \mathbb{R}^{D+K+1}$, and $Q_t^{(0)} \doteq q_0 I$

- Here, α is a *forgetting factor*: recent transitions are given more weight

Experiments

- Our method works in high-dimensional, stochastic, non-smooth environments



Open-loop reinforcement learning is an effective strategy to solve challenging tasks without function approximation!

Paper
Code
Video