Sobolev Diffusion Policy

Théotime Le Hellard[1,*], Franki Nguimatsia Tiofack[1,*], Quentin Le Lidec[1,2] and Justin Carpentier[1]

[1]Willow, Inria - Département d'Informatique de l'École normale supérieure ENS, PSL Research University,
[2]New York University, theotime.le-hellard@inria.fr, franki.nguimatsia-tiofack@inria.fr

Overview

Trajectory optimization (TO) and policy learning (PL) offer complementary strengths for controlling complex dynamic systems.

Control problem:

$$\min_{\theta} \mathop{\mathbb{E}}_{\substack{\xi\sim\mathcal{P},\\ X,U\sim\pi_\theta}} J(X,U;\xi) = \sum_{t=0}^{T-1} \ell_t(x_t,u_t;\xi) + \ell_T(x_T;\xi) \quad \text{s.t.} \quad x_{t+1} = f(x_t,u_t), \quad x_0 = \hat{x}(\xi),$$

First idea:

- For a fix ξ, TO solvers find local minima from initial guesses

(1) Collect some trajectories using a TO solver [1]

(2) Train a diffusion policy [2, 3]

(1') Repeat, but use the policy to get initial guesses for the solver

Second idea: gradient based solvers generate feedback gains [1], we integrate this first-order information in the policy training loss, similar to what [4] did with standard policies

$$\mathcal{L} = \mathop{\mathbb{E}}_{(\xi,\tau_0,\frac{\partial\tau_0}{\partial x_{hist}}),\varepsilon,k} \left[\left\| \tau_0 - \tau_0^{\theta,k} \right\|^2 + \left\| \frac{\partial \tau_0}{\partial x_{hist}} - \frac{\partial \tau_0^{\theta,k}}{\partial x_{hist}} \right\|^2 \right]$$

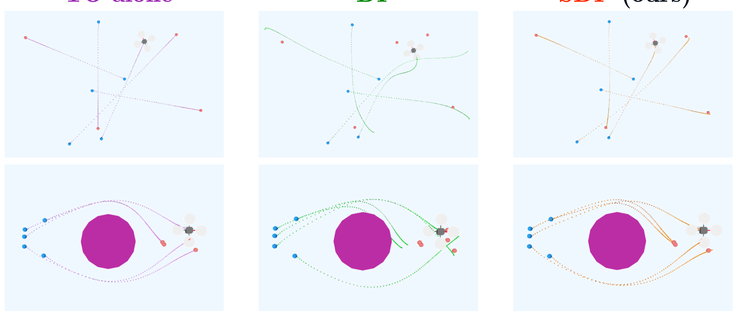
For stochastic Sobolev [5], using n_{proj} random vectors v_i:

$$\boldsymbol{\mathcal{L}^{SDP}} = \mathbb{E}\left[\cdots + \frac{\alpha_{Sobolev}}{n_{proj}} \sum_{i=1}^{n_{proj}} \left\| v_i^\top \cdot \frac{\partial \tau_0}{\partial x_{hist}} - \frac{\partial}{\partial x_{hist}} \left(v_i^\top \cdot \tau_0^{\theta,k} \right) \right\|^2 \right]$$

Compounding error issue

We compare our Sobolev Diffusion Policy (SDP), to standard Diffusion Policy (DP)[2, 3], and to ProxDDP[1] with interpolation-based initial guesses, referring to it as *TO alone*.

TO alone | **DP** | **SDP (ours)**



Sobolev diffusion policy

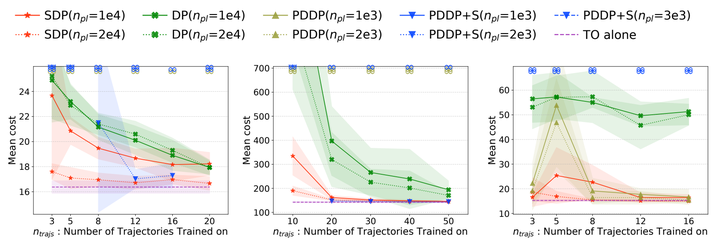
Training:

Initialize: Diffusion policy network τ_θ; Buffer $\mathcal{D} = \emptyset$
for n_{algo} **do**
  // Collect new trajectories
  if *reset buffer* **then** $\mathcal{D} = \emptyset$
  for n_{traj} **do**
    Sample $\xi \sim \mathcal{P}$
    $X^{\text{naive}}, U^{\text{naive}} = Interpolate(x_0, x_{target})$
    $X^{\text{pol}}, U^{\text{pol}} =$ **PolicyRollout**$(\tau_\theta, \xi, T_a, K_{rollout})$
    $X, U, \frac{\partial u_t}{\partial x_t}, \frac{\partial x_{t+1}}{\partial x_t} = ArgminCost\,($
      TrajOpt$(X^{\text{naive}}, U^{\text{naive}}, \xi)$,
      TrajOpt$(X^{\text{pol}}, U^{\text{pol}}, \xi)$
    $) \mathcal{D} \leftarrow \mathcal{D} \cup \{(X, U, \frac{\partial u_t}{\partial x_t}, \frac{\partial x_{t+1}}{\partial x_t})\}$
  // Policy learning
  for n_{pl} *epochs* **do**
    // Sample by batches
    $t \sim \mathcal{U}(0, T - T_h)$
    Sample ξ, τ_0, x_{hist}, o_{hist} and derivatives in $\mathcal{D}$
    $\frac{\partial \tau_0}{\partial x_{hist}} = ChainRule(\frac{\partial x_{t+1}}{\partial x_t}, \frac{\partial u_t}{\partial x_t})$
    $k \sim \mathcal{U}(1, K_{train})$; $\varepsilon \sim \mathcal{N}(0, I)$
    // Apply noise and inpainting
    $\tau_k = \sqrt{\bar{\alpha}_k}\tau_0 + \sqrt{1-\bar{\alpha}_k}\varepsilon$
    $\tau_k[1:T_o] = a_{hist}$
    // Compute loss $\mathcal{L}^{SDP}$
    $\tau_0^{\theta,k} = \tau_\theta(\tau_k, k, \xi, o_{hist})$
    Take gradient descent step on $\nabla_\theta \mathcal{L}^{SDP}(\theta)$

Policy rollout:

Input: τ_θ, ξ, T_a, $K_{rollout}$
Initialize: $t = 0$, o_{hist}, a_{hist}
while $t < T$ **do**
  // Full reverse denoising process
  $\tau_K \sim \mathcal{N}(0, I)$
  for $k = K_{rollout}$ **to** 1 **do**
    $\tau_k[1:T_o] = a_{hist}$ // inpainting
    $\tau_0^{\theta,k} = \tau_\theta(\tau_k, k, \xi, o_{hist})$
    $\tau_{k-1} \sim \mathcal{N}(\bar{\mu}_k(\tau_k, \tau_0^{\theta,k}), \tilde{\beta}_k \mathbf{I})$
  // Play predicted actions
  for $s = 1$ **to** T_a **do**
    if $t + s > T$ **then** ***stop***
    if a is u **then** $u_{t+s-1} = \tau_0[s]$
    else $u_{t+s-1} = InvDynamics(x_{t+s-1}, \tau_0[s])$
    $x_{t+s} = Dynamics(x_{t+s-1}, u_{t+s-1})$
  Update o_{hist} and a_{hist} : $t = t + T_a$
Output: X, U

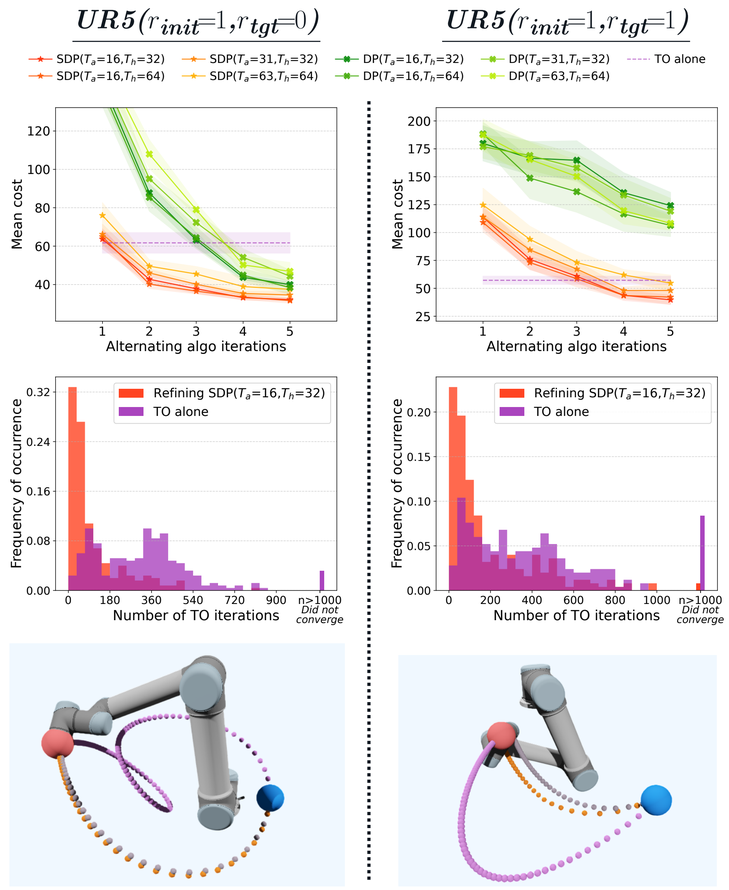
Learning capacities



UR5(r_{init}=0,r_{tgt}=0) | Quadrotor | Inverted pendulum

Mean cost on test instances *w.r.t* n_{traj}, the number of trajectories in the dataset. Here n_{algo} is fixed to 1. Curves with the same color but different line styles differ in the number of training epochs n_{pl}.

Full algorithm evaluation



To study the impact of the prediction horizon, both DP and SDP are tested with $T_h = 32$ and $T_h = 64$, and various action lengths T_a.

References

[1] Wilson Jallet et al. "ProxDDP: Proximal constrained trajectory optimization". In: *IEEE Transactions on Robotics* (2025).

[2] Michael Janner et al. "Planning with Diffusion for Flexible Behavior Synthesis". In: *International Conference on Machine Learning*. PMLR. 2022.

[3] Cheng Chi et al. "Diffusion policy: Visuomotor policy learning via action diffusion". In: *arXiv preprint arXiv:2303.04137* (2023).

[4] Quentin Le Lidec et al. "Enforcing the consensus between Trajectory Optimization and Policy Learning for precise robot control". In: *2023 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2023.

[5] Wojciech M Czarnecki et al. "Sobolev training for neural networks". In: *Advances in neural information processing systems* 30 (2017).