Sobolev Diffusion Policy

Théotime Le Hellard, Franki Nguimatsia Tiofack, Quentin Le Lidec, Justin Carpentier

Sobolev Diffusion Policy

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

***Abstract*— We present Sobolev diffusion policy (SDP), a novel framework to combine the strengths of policy learning and trajectory optimization effectively. On the one hand, we build upon diffusion policy, an expressive imitation learning method based on diffusion probabilistic generative models. On the other hand, we use gradient-based trajectory optimization solvers to generate locally optimal trajectories and leverage their associated feedback gains to enrich Sobolev training with first-order information. Combining both, we introduce a first-order loss for diffusion-based policies. The framework alternates between collecting trajectories using a solver warm-started by the policy and training. Through comprehensive experiments, we demonstrate how the Sobolev component significantly reduces the number of trajectories required for the policy to converge globally. First-order information both avoids overfitting, despite the use of very few samples, and mitigates the compounding error issue of imitation-based policies, even when predicting torques for tasks requiring high-frequency control. We benchmark the benefits of SDP on various robotics tasks of increasing complexity. In particular, SDP shows to be stable over extended horizons, with fewer diffusion steps, shrinking the overall rollout time compared to vanilla diffusion models. And when used to compute initial guesses for trajectory optimization, it reduces the solving time by a factor of 2 to 20.**

I. Introduction

Trajectory optimization (TO) and policy learning (PL) offer complementary strengths for controlling complex dynamic systems. On the one hand, TO leverages information from known system dynamics to compute locally optimal trajectories, but starts over the optimization on each problem instance. On the other hand, PL learns global control policies from data, but often requires numerous samples to converge. Intuitively, policies can be trained on trajectories found by a TO solver [1], and once trained, they can be rolled out to provide initial guesses to the solver. In a virtuous circle, one can alternate between collecting trajectories and training. The policy provides initial guesses, and the TO solver refines them towards optimal solutions.

In supervised learning, diffusion models [2]–[5] recently stood out for their generalizing capacities. Initially developed for image generation, diffusion-based policies (DP) [6]–[11] quickly became the go-to approach in policy learning, driving significant investments in robotics. These approaches often involve a costly data collection process, typically countless human demonstrations, *e.g.*, via teleoperation or virtual reality VR. For this, using TO solvers to generate [12] or retarget [13] demonstrations is promising to get enhanced trajectories.

Fortunately, research in TO is also thriving. The development of general differentiable physical simulators [14]–[17], empowers gradient-based TO solvers [18]–[21]. In addition to finding trajectories, these solvers compute some first-order information. These feedback gains come at no additional cost and can improve the stability of MPC approaches on locomotion tasks [22]. They can also be leveraged when training a policy on TO-generated trajectories, for first-order policy learning [23], [24]. The latter corresponds to Sobolev training [25], [26], a first-order supervised learning method that accelerates convergence and mitigates overfitting, thereby reducing the number of trajectories required for the policy to generalize.

In this letter, we propose adapting the Sobolev training formulation to diffusion-based policy learning, which we refer to as the Sobolev diffusion policy (SDP). Our framework alternates between collecting trajectories using a gradient-based TO solver, and first-order policy learning. Compared to standard, non-diffusion-based, first-order policies [23], [24], our method scales to complex tasks, and compared to standard, non-Sobolev, diffusion methods, it requires fewer trajectories to learn the diffusion models. In particular, SDP exhibits remarkable resilience to the compounding error issue that imitation-based policies often struggle with, where small errors not observed during training cause the policy to fall out of distribution and enter a downward spiral. Unlike human demonstrations, TO generates locally optimal trajectories, which by definition do not contain small missteps, so the compounding error risk might be exacerbated. Thankfully, first-order information guides the policy on how to recover from small deviations. [7] advocates for torque control, but due to a lack of smoothness, it predicts velocities, while [9] does position control to avoid the compounding error issue. In contrast, SDP can reliably do velocity control and even directly predict torques for tasks requiring high-frequency control. Our policy can predict over longer horizons, with fewer diffusion steps, hence mitigating the slow inference time of diffusion. When used to generate initial guesses for TO, SDP leads to more optimal trajectories and greatly reduces the solving time.

In this letter, we make the following contributions:

- we introduce a first-order loss for diffusion-based policy learning, improving sample efficiency, and enhancing policy expressiveness and stability
- we propose an interplay algorithm, alternating between collecting trajectories and training, to solve hard tasks that TO struggle on without proper initial guesses,

[*] Equal contributions

[1] Inria - Département d'Informatique de l'École normale supérieure, PSL Research University, France. `{firstname.surname}@inria.fr`
[2] New York University, USA. `ql2867@nyu.edu`

- we present comprehensive experiments evaluating the sensitivity of (i) the number of collected trajectories, (ii) the number of training epochs, and (iii) the prediction horizon, across 3 different robotics tasks with 8 variants.

The rest is structured as follows. Sec. II provides a background on trajectory optimization and diffusion at the root of our approach. Sec. III introduces our Sobolev diffusion policy algorithm. Sec. IV presents our extensive set of experiments. Finally, Sec. V discusses related works and future ideas to explore.

II. Background

A. Optimal control problems

We consider discretized dynamics for robotic systems: at time step $0 \leq t \leq T$, q_t and v_t represent the position and velocity of a robot, respectively, and $x_t = (q_t, v_t)$ is the state. Function f represents the dynamics of the system (deterministic and known), for $t \in [0, T-1]$, u_t is the control input, such that $x_{t+1} = f(x_t, u_t)$. We extend these notations to time segments with a subscript: $q_{t_1:t_2}$ is the matrix whose rows are $q_{t_1}, q_{t_1+1}, \ldots, q_{t_2}$. The state and control trajectories are $X = x_{0:T}$ and $U = u_{0:T-1}$. Let us consider the following formulation of constrained optimal control problems (OCP):

$$\begin{aligned} \min_{X,U} \quad & J(X, U; \xi) = \sum_{t=0}^{T-1} \ell_t(x_t, u_t; \xi) + \ell_T(x_T; \xi) \\ \text{s.t.} \quad & x_{t+1} = f(x_t, u_t), \quad x_0 = \hat{x}(\xi), \\ & h_t(x_t, u_t; \xi) \leq 0, \quad h_T(x_T; \xi) \leq 0. \end{aligned} \tag{1}$$

ξ are task parameters defining the initial position $\hat{x}$ and parameterizing both the constraints h_t and h_T, and J the objective function to minimize, which is given as the sum of costs ℓ_t at each time step t and the final cost ℓ_T.

B. Second-order trajectory optimization

To efficiently find local optimum to OCPs, gradient-based TO solvers, such as iterative linear quadratic regulator (iLQR) [18] and differential dynamic programming (DDP) [19], [21], leverage derivatives of rigid body algorithms [14] and differentiable physics simulators [15]–[17]. The present paper makes use of iLQR, as it yields local *feedback gains* that we exploit in the policy learning part through Sobolev training (Sec. II-D). We use the ProxDPP variant [21], which correctly handles constrained OCPs of the form (1).

iLQR iteratively optimizes a trajectory by (i) deriving a second-order Taylor expansion of the objective and first-order Taylor expansion of the dynamics near the current trajectory, and (ii) taking a step in the appropriate direction. Let $J_t(x_{\geq t}, u_{\geq t}) = \sum_{i \geq t}^{T-1} \ell_t(x_t, u_t) + \ell_T(x_T)$ be the objective function from step t onwards, $V_t(x_t) = \min_{x_{\geq t+1}, u_{\geq t}} J_t$ the optimal value starting from a given x_t. The Bellman principle states that $V_t(x_t) = \min_{u_t} \ell_t(x_t, u_t) + V_{t+1}(f_t(x_t, u_t))$, starting with $V_T(x_T) = \ell_T(x_T)$ and then t from T–1 down to 0. So, the Taylor expansion of V_t around a nominal trajectory can be expressed using the second-order derivatives of the costs ℓ_t and dynamic f_t, and the Taylor expansion of the next value function V_{t+1}. This expansion results in the computation of *feedback gains* which correspond to $\frac{\partial u_t}{\partial x_t}$.

Thereupon convergence, in addition to the produced trajectories X and U, DDP-like methods estimate $\frac{\partial u_t}{\partial x_t}$, for all $t \in [0, T-1]$. Then, as $x_{t+1} = f(x_t, u_t)$, by composing these derivatives with the differentiable dynamics, we get $\frac{\partial x_{t+1}}{\partial x_t}$.

C. Diffusion models for policy learning

Diffusion models [2] are probabilistic generative models that progressively disrupt data by injecting noise, then learn to reverse this process for sample generation. Various theoretical formulations of diffusion models have been derived, either based on the theory of stochastic differential equations, such as score-based approaches, using Stein scores and Langevin dynamics [2], [4], and denoising diffusion probabilistic models (DDPM) [3], or based on the theory of ordinary differential equations, such as flow matching methods [5]. Following the emergence of these various formulations and successful image generation applications, control policies for robotics have been derived. Here, we use the DDPM approach [3] and follow how it is used in the original diffusion policy papers [6], [7], [9].

Denoising diffusion probabilistic models. For $\mathcal{D} = \{\tau_0\}$ a dataset, and $p_{\mathcal{D}}$ the underlying distribution, a probabilistic generation model aims at sampling from $p_{\mathcal{D}}$. DDPM [3] defines a diffusion process using two Markov chains with Gaussian transitions. The *forward noising* chain starts from $\tau_0 \sim p_{\mathcal{D}}$ and gradually add noise over K steps: $p_{\mathcal{D}}(\tau_k | \tau_{k-1}) := \mathcal{N}(\tau_k; \sqrt{1-\beta_k}\tau_{k-1}, \beta_k \mathbf{I})$ for $k \in [1, K]$, with $\beta_1, \ldots, \beta_K$ the noising schedule.

Chaining the Gaussians, one can directly sample τ_k from τ_0, by sampling a noise $\varepsilon \sim \mathcal{N}(0, \mathbf{I})$ and $\tau_k := \sqrt{\bar{\alpha}_k}\tau_0 + \sqrt{1-\bar{\alpha}_k}\varepsilon$, with $\bar{\alpha}_k := \prod_{s=1}^{k} 1 - \beta_s$. Thus, forward posteriors conditioned on τ_0 are tractable: $p_{\mathcal{D}}(\tau_{k-1} | \tau_k, \tau_0) = \mathcal{N}(\tau_{k-1}; \tilde{\mu}_k(\tau_k, \tau_0), \tilde{\beta}_k \mathbf{I})$ with $\tilde{\mu}_k(\tau_k, \tau_0) = \frac{\sqrt{\bar{\alpha}_{k-1}}\beta_k}{1-\bar{\alpha}_k}\tau_0 + \frac{\sqrt{\bar{\alpha}_k}(1-\bar{\alpha}_{k-1})}{1-\bar{\alpha}_k}\tau_k$ and $\tilde{\beta}_k = \frac{1-\bar{\alpha}_{k-1}}{1-\bar{\alpha}_k}\beta_k$.

On the opposite, the *reverse* process starts from pure noise $\tau_K \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ and is trained to recover distribution $p_{\mathcal{D}}$ through a chain of Gaussians parameterized by θ: $p_\theta(\tau_{k-1} | \tau_k) = \mathcal{N}(\tau_{k-1}; \mu_\theta(\tau_k, k), \tilde{\beta}_k \mathbf{I})$ using the same $\tilde{\beta}_k$ as the forward posterior. This way, the KL divergence between $p_{\mathcal{D}}$ and p_θ boils down to a weighted sum over $k \in [1, K]$ of the difference between $\tilde{\mu}_k(\tau_k, \tau_0)$ and $\mu_\theta(\tau_k, k)$.

The idea is to train a neural network τ_θ to predict τ_0 from τ_k and k, then at inference, sample $\tau_K \sim \mathcal{N}(0, \mathbf{I})$ and for k from K down to 1, use the close form of $\tilde{\mu}_k$ to sample τ_{k-1}, *i.e.* $\tau_{k-1} \sim \mathcal{N}(\tilde{\mu}_k(\tau_k, \tau_\theta(\tau_k, k)), \tilde{\beta}_k \mathbf{I})$. Finally, DDPM authors recommend discarding the weighting when averaging over $k \in [1, K]$:

$$\mathcal{L}^{DDPM} = \mathop{\mathbb{E}}_{\tau_0, \varepsilon, k} \left[\|\tau_0 - \tau_\theta(\tau_k, k)\|^2 \right] \tag{2}$$

As an alternative parameterization, one can also choose to train ε_θ to predict the noise ε. In our experiments, predicting τ_0 performed better than predicting ε, so for the rest of the paper, we stick with the τ_0 parametrization. Since τ_0

can be deduced from ε using $\tau_0 = \frac{1}{\sqrt{\bar{\alpha}_k}}(\tau_k - \sqrt{1-\bar{\alpha}_k}\varepsilon)$, everything could be derived using ε_θ instead of τ_θ.

Diffusion policy (DP). Policy learning trains policy π_θ, with parameters $\theta \in \Theta$, to minimize the constrained objective function (1) in expectation over task parameters $\xi \sim \mathcal{P}$. Stochastic policies learn a conditional distribution of torque given the current state: $u_t \sim \pi(u_t|x_t;\xi)$, typically modeling the distribution as Gaussian. But recently, Janner et al [6] proposed to train diffusion models to control robots.

Given a dataset $\mathcal{D}$ of successful trajectories of a desired task, they learned the distribution of chunks of these trajectories. In previous works [6], [7], [9], trajectories are sequences of states s_t and/or actions a_t, but what these variables refer to may vary. Following the notations of diffusion policy [9], in this paper, we denote by a_t what comprises the trajectories being diffused, and o_t the observations conditioning the diffusion process. o_t may be composed of both proprioceptive feedback, *e.g.*, joint positions q_t, velocities v_t, coordinate positions x,y,z of a key component of the robot (typically, the end-effector) and perhaps associated velocities Δx, Δy, Δz; and exteroceptive sensors, *e.g.*, cameras or the position of some obstacles. As for a_t, it is either q_t or x,y,z for position control, v_t or Δx, Δy, Δz, for velocity control, or u_t for torque control.

Training a diffusion model over the distribution of chunks of trajectories means $\tau_0 = a_{t_1:t_1+T_h-1}$ is sampled in $\mathcal{D}$, with t_1 a starting time step and T_h the horizon. For instance, when $a_t = q_t$, the diffused variable τ is a sequence of joint positions over T_h time steps. The idea consists of conditioning the diffusion process on the current state, observations, and task parameters ξ, and predicting the next steps. To condition the diffusion process, Janner et al [6] uses inpainting: for $a_t = q_t$, they impose $\tau_k[1] = q_{t_1}$ (the current position) and $\tau_k[T_h] = q_{target}$ (the target position), at all diffusion steps $k \in [1, K]$ through the denoising process. To generalize to more diverse tasks, [7] and [9] use classifier-free guidance diffusion, which consists in conditioning the denoising neural network not only on the diffusion step k, but also ξ (for instance, a target position) and the last T_o observations o_t (including the current state). At inference time, the policy predicts T_h time steps, out of which T_a steps are rolled out. If $a_t = u_t$, it means applying the predicted torques over the next T_a time steps; if instead $a_t = q_t$ or v_t, then u_t is deduced by inverse dynamics. In [6] and [7], $T_a = 1$, *i.e.*, only one action is rolled out, despite predicting $T_h \gg 1$ steps. This leads to a slow inference time, where, for each time step, a chunk of size T_h is generated, resulting in the computational cost of K calls to τ_θ for the K denoising steps. To mitigate this, [9] sets T_a to 8, replanning every 8 steps.

D. *Sobolev training for policy learning*

To combine the strengths of TO and PL, one can alternate between (i) collecting a dataset $\mathcal{D}$ of locally optimal trajectories using TO ; (ii) training a policy on $\mathcal{D}$; then repeat: (i') collecting new trajectories, but this time starting the optimization from trajectories obtained using the policy. Leveraging feedback gains $K_t \sim \frac{\partial u_t}{\partial x_t}$ produced by some TO methods (Sec. II-B), [23] proposed using tangent propagation [25], equivalently described as training in Sobolev spaces. The Sobolev training loss for a standard policy is:

$$\mathcal{L}^{\textit{Sobolev}} = \mathop{\mathbb{E}}_{(\xi, x_t, u_t, \frac{\partial u_t}{\partial x_t}) \sim \mathcal{D}} \Bigg[\|u_t - \pi_\theta(x_t,\xi)\|^2 + \left\| \frac{\partial u_t}{\partial x_t} - \frac{\partial \pi_\theta}{\partial x}(x_t,\xi) \right\|^2 \Bigg] \quad (3)$$

Gradient descend of $\mathcal{L}^{\textit{Sobolev}}$ on θ requires computing the expensive $\partial_{\theta,x}\pi_\theta$ Hessian. Instead, [24] uses stochastic Sobolev training [26] to boost computational efficiency. This stochastic variant samples $n_{\textit{proj}}$ random vectors v_i on the unit sphere (with $n_{\textit{proj}} = 1$ turning out to be sufficient in our case), on which the control gradients are projected:

$$\mathcal{L}^{\textit{Sobolev}}_{\textit{Stochastic}} = \mathop{\mathbb{E}}_{(\xi, x_t, u_t, \frac{\partial u_t}{\partial x_t}) \sim \mathcal{D}} \Bigg[\|u_t - \pi_\theta(x_t,\xi)\|^2 + \frac{1}{n_{\textit{proj}}} \sum_{i=1}^{n_{\textit{proj}}} \left\| v_i^\top \cdot \frac{\partial u_t}{\partial x_t} - \frac{\partial}{\partial x}(v_i^\top \cdot \pi_\theta(x_t,\xi)) \right\|^2 \Bigg] \quad (4)$$

III. Sobolev Diffusion Policy

A bottleneck in diffusion policies is the number of trajectories that need to be collected. Decision diffuser [7] uses 10000 trajectories for the Kuka block stacking task, diffusion policy [9] requires hundreds of human teleoperated trajectories. Even more concerning is the compounding error effect, where small deviations during rollouts make the policy go out of distribution and enter a downward spiral. To avoid this effect, diffusion policy [9] authors later highlighted the need to include non-perfect trajectories in the dataset. Training on a mixture of more or less good trajectories mitigates the risk of following out-of-distribution, but at a cost of training a flawed policy. It also makes the data collection process non-intuitive, having to predict how the policy could fail, similar to the DAgger method [27]. Training a policy on locally optimal trajectories goes straight against this advice. By definition, trajectories obtained from optimal control do not have small errors (being locally optimal). This effect can eventually be mitigated by getting enough trajectories, and some works successfully trained diffusion policies using TO [12], [13].

We propose using stochastic Sobolev training to enhance diffusion policies trained on TO-generated trajectories, making use of first-order information. We show how this reduces the number of trajectories needed, improves control precision (easing the compounding error effect), and allows increasing T_a - the number of steps applied per diffusion inference, hence mitigating the slow rollout time of diffusion policies. We name *Sobolev diffusion policy* both our policy learning method and the algorithm alternating between trajectory collection using TO and training.

To recap all notations, a_t is the action variable; in our implementation, either q_t, v_t, x_t, or u_t. In our case, the observation variable o_t includes x_t, u_{t-1} and task parameters ξ. Diffusion is done on chunks of variables: $\tau_0 = a_{t_1:t_1+T_h-1}$. The horizon T_h is the length of this chunk, T_a is the action length, *i.e.* the number of applied actions before replanning, and finally, T_o is the history length. The first T_o elements of τ_0 are the current and past actions (by inpainting), only then come the T_a used ones, hence $T_a \leq T_h - T_o$.

As explained in Sec. II-B, by chaining the feedback gains $K_t \sim \frac{\partial u_t}{\partial x_t}$ produced by the solvers, with dynamic derivatives, we get $\frac{\partial x_{t+1}}{\partial x_t}$. Further chaining these derivatives, we can estimate $\frac{\partial a_{t+h}}{\partial x_{t+o}}$ for $t \in [0, T-1]$, $0 \leq o \leq h < T_h$. Finally, stacking these derivatives we get $\frac{\partial a_{t_1:t_1+T_h-1}}{\partial x_{t_1:t_1+T_o-1}}$, *i.e.* $\frac{\partial \tau_0}{\partial x_{hist}}$, with $x_{hist} = x_{t:t+T_o-1}$, $o_{hist} = o_{t:t+T_o-1}$, $a_{hist} = a_{t:t+T_o-1}$.

From this point, one could adapt $\mathcal{L}^{Sobolev}$ (3) by comparing the original τ_0 and its derivatives, to $\bar{\tau}_0^\theta$, the final output after all K denoising steps, and the derivatives of the entire diffusion process:

$$\mathcal{L}^{SDP}_{full} = \underset{(\xi, \tau_0, \frac{\partial \tau_0}{\partial x_{hist}}), \tau_K}{\mathbb{E}} \left[\left\| \tau_0 - \bar{\tau}_0^\theta \right\|^2 + \left\| \frac{\partial \tau_0}{\partial x_{hist}} - \frac{\partial \bar{\tau}_0^\theta}{\partial x_{hist}} \right\|^2 \right] \quad (5)$$

However, this is not how diffusion models are typically trained, and it did not work in practice. Instead of training over the whole diffusion process at once, DDPM samples ε and k, and trains the model to predict τ_0 from τ_k. We name $\tau_0^{\theta,k}$ this estimate of τ_0 from step k, not to be confused with $\bar{\tau}_0^\theta$, the latter being the final output of the whole diffusion process. To appropriately combine $\mathcal{L}^{SDP}$ (2) and $\mathcal{L}^{Sobolev}_{Stochastic}$ (4), we propose to guide the derivatives of each denoising step.

$$\boldsymbol{\mathcal{L}}^{\boldsymbol{SDP}} = \underset{(\xi, \tau_0, \frac{\partial \tau_0}{\partial x_{hist}}), \varepsilon, k}{\mathbb{E}} \left[\left\| \tau_0 - \tau_0^{\theta,k} \right\|^2 + \frac{\alpha_{Sobolev}}{n_{proj}} \sum_{i=1}^{n_{proj}} \left\| v_i^\top \cdot \frac{\partial \tau_0}{\partial x_{hist}} - \frac{\partial}{\partial x_{hist}} \left(v_i^\top \cdot \tau_0^{\theta,k} \right) \right\|^2 \right] \quad (6)$$

The full training procedure, Algorithm 1, alternates between collecting trajectories with their derivatives, using TO and the policy, and training the network just as DP but using the first-order loss $\boldsymbol{\mathcal{L}}^{\boldsymbol{SDP}}$ (6). For policy rollout, Algorithm 2 iteratively generates T_h actions using the DDPM inference scheme, out of which T_a actions are rolled out.

Algorithm 1: Sobolev diffusion policy - Training

```
Initialize: Diffusion policy network $\tau_\theta$; 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 $\xi$, $\tau_0$, $x_{hist}$, $o_{hist}$ 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 $\mathcal{L}^{SDP}$ (6)
        $\tau_0^{\theta,k} = \tau_\theta(\tau_k, k, \xi, o_{hist})$
        Sample $n_{proj}$ vectors $v_i$ in the unit sphere
        Take gradient descent step on $\nabla_\theta \mathcal{L}^{SDP}(\theta)$
```

Algorithm 2: Sobolev diffusion policy - Rollout

```
Input: $\tau_\theta$, $\xi$, $T_a$, $K_{rollout}$
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
        $\tau_0^{\theta,k} = \tau_\theta(\tau_k, k, \xi, o_{hist})$
        $\tau_{k-1} \sim \mathcal{N}(\tilde{\mu}_k(\tau_k, \tau_0^{\theta,k}), \tilde{\beta}_k \mathbf{I})$

    // 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 $u_{t+s-1} = InvDynamics(x_{t+s-1}, \tau_0[s])$
        $x_{t+s} = Dynamic(x_{t+s-1}, u_{t+s-1})$
    Update $o_{hist}$ and $a_{hist}$
    $t = t + T_a$
Output: $X$, $U$
```

IV. Experiments

Our **SDP** is evaluated against alternative policy learning methods such as *projected DDP* (**PDDP**) and its Sobolev variant (**PDDP+S**) from [24], and **DP** [9]. Both position control and torque control are evaluated. We first fix n_{algo} to 1, focusing solely on learning capacities (no further collecting trajectories). We show how SDP can generalize based on fewer trajectories, while achieving better control and being less prone to overfitting. Then, we experiment on the full algorithm ($n_{algo} > 1$), alternating between trajectory collection and policy learning, as described in Algo 1. Again, SDP is compared to PDDP, PDDP+S, and DP. To our knowledge, using DP to power such an interplay loop is original, but for clarity, we unify the name of the interplay loop with the name of the policy loss used. We demonstrate that warm starting the TO solver with the trained SDP policy drastically reduces the solving time and leads to more optimal trajectories. Compared to DP, SDP allows for an increased action length T_a, mitigating the slow inference time of diffusion-based methods.

A. Experimental setup and implementation details

Our code will be made public upon paper acceptance. We use Pinocchio [28] and ALIGATOR [29] to define and solve the constrained OCPs (1), Candlewick [30] for the visualization. For the learning part, we use PyTorch with the same conditional U-Net as [6] and [9], the DDPM scheduler is square cosine from improved DDPM [31] and HuggingFace implementation. All experiments were run on a single laptop, with an Intel Core i9-13950HX and an Nvidia RTX2000 ADA. The number of iterations the TO solver can do is restricted to 1000. We let $n_{proj} = 1$, sampling only one vector for the stochastic Sobolev training, so training SDP on one epoch takes about twice more time as DP. We fix the hidden dimensions of the conditional U-Net to $[24, 24, 32, 32]$, resulting in approximately 300k parameters. From our experiments, we found that fixing K, the number of diffusion steps, to 5 was sufficient (compared to 100 in diffusion policy [9]). By default, $T_h = 32$, $T_o = 1$ and all generated actions are applied, *i.e.* $T_a = T_h - T_o = 31$.

To compare the training time fairly, we adapt the size E of an "*epoch*" to the method. For standard policies, predicting u_t given x_t (*e.g.*, PDDP), $E = |\mathcal{D}| \times T$, the total number of state-action pairs in the dataset. But for diffusion-based policies, predictions are made over horizon length T_h, covering T_h steps, so we chose to define $E = |\mathcal{D}| \times (T - T_h)/T_h$, which is the number of chunks.

To avoid learning on bad trajectories, we reject trajectories when the TO solver still has not converged after 1000 iterations, and we keep trying until $|\mathcal{D}| = n_{traj}$. We find it beneficial to reset $\mathcal{D}$ after each iteration (Algo 1, line 2); still, as the policy improves, we may choose to increase n_{traj}, since trajectories are faster to collect.

All results are averaged over 5 random seeds, displayed with confidence intervals. On each seed, the average cost J is estimated using 50 randomly sampled $\xi \sim \mathcal{P}$. When the mean cost is higher than 10^5, we put an ∞ sign at the top of the plot. We assigned colors to methods for visual consistency across figures: SDP, DP, PDDP, and PDDP+S. As a reference, we evaluate the performance of TO with interpolation-based initial guesses, reported as TO alone.

B. *Tasks descriptions*

Methods are evaluated on goal-reaching tasks, with a UR5, a quadrotor, and inverted pendulums. The implementation is modular and works on any problem defined using ALIGATOR [29]. We use $\alpha_{Sobolev} = 1.0$, except for the quadrotor task where $\alpha_{Sobolev} = 10^{-3}$ performs better.

Inverted pendulums. The goal consists of swinging a single or double pendulum from a randomly sampled downward position to the upright unstable equilibrium point. To avoid ill-conditioned solutions, $|u_t|$ is bounded by 25 and a strong regularization on u makes the task harder for the TO solver, $\ell_t(x_t, u_t) = 10 \left\| \textit{end-effector}(x_t) - \textit{goal} \right\|^2 + 0.1 \left\| u_t \right\|^2$. The chaotic properties of this task impose torque control, $a_t = u_t$.

UR5. To gradually increase the complexity of the goal-reaching task with a UR5 robotic arm, we tried 4 variants. We define two boolean variables: *fully random init* r_{init}, whether the initial position is sampled in a small reasonable area ($r_{init} = 0$) or anywhere is the robot space ($r_{init} = 1$); and *random target* r_{tgt}, whether the end-effector target position is fixed (r_{tgt}=0) or random (r_{tgt}=1). As a reference, in [24], PDDP+S was tested on *UR5*($r_{init} = 0, r_{tgt} = 0$). q_t is composed of the 6 joint angles, and following [24], q_t is preprocessed to $(\cos q_t, \sin q_t)$, derivatives are adjusted accordingly (Algo 1, line 14). For this task, we set $a_t = x_t$, doing state control. u_t is computed using RNEA [28], so only v_t is used, but predicting q_t too stabilizes the training. The only constraints are torque limits.

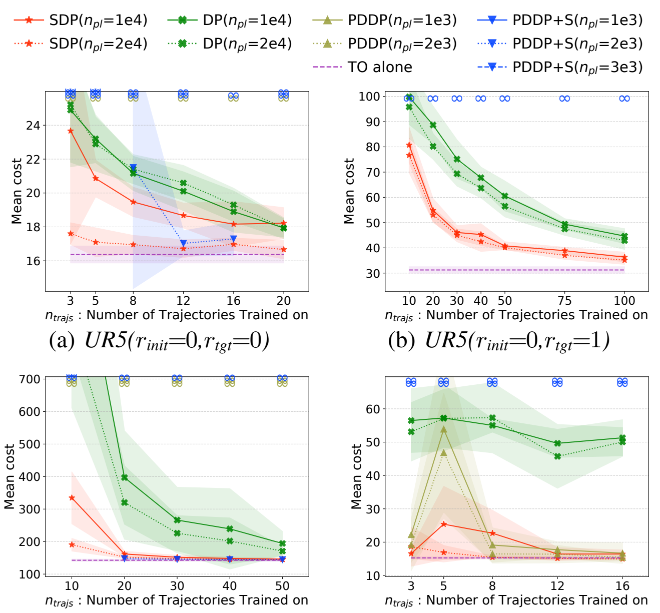
Quadrotor. We first evaluate all methods on controlling a quadrotor to go from a random initial position to a random goal, at the same height, with torque limits, ceiling, and floor constraints. Then, up to 9 columns are added to the environment as obstacles for the quadrotor to avoid. q_t is the SE3 placement of the quadrotor, preprocessed by projecting it to the tangent space $dq_t = q_t \ominus_{SE3} q_{ref}$, and since state control works better on this task too, $a_t = (dq_t, v_t)$.

C. *Learning capacities and sample efficiency*

In this subsection, we study the learning capacities of our Sobolev diffusion policy, so n_{algo} is fixed to 1 (*i.e.* no further collecting trajectories). On Fig. 1, PL methods are evaluated on their sample efficiency, plotting the average cost depending on the size of the dataset $n_{traj} = |\mathcal{D}|$ (results are collected independently, with a fixed dataset for each run). To investigate the dependence on the number of training epochs n_{pl}, each method is evaluated at different times during training. Both SDP and DP are evaluated at $n_{pl} = 10^4$ and $n_{pl} = 2 \cdot 10^4$ (except on the single pendulum, where 10^3 is enough), while PDDP and PDDP+S are evaluated at $n_{pl} = 10^3$ and $n_{pl} = 2 \cdot 10^3$. On all tasks, our SDP reaches the performance of the TO solver, achieving both state control and torque control, even when trained with very few trajectories. Compared to DP, SDP usually needs between 5 to 10 times fewer trajectories. On the inverted pendulum task, SDP succeeds with $n_{traj} = 3$, while DP systematically fails to operate torque control. Overall, PDDP and PDDP+S are unstable; they often diverge on some trajectories, which shifts their mean cost to extreme values. On the *UR5*($r_{init} = 0, r_{tgt} = 0$) task, Fig. 1a, PDDP+S is additionally evaluated with $n_{pl} = 3 \cdot 10^3$, it reveals that PDDP+S only succeeds with $8 \leq n_{traj} \leq 16$ and $n_{pl} = 2 \cdot 10^3$, otherwise following into overfitting.

D. *Full algorithm evaluation*

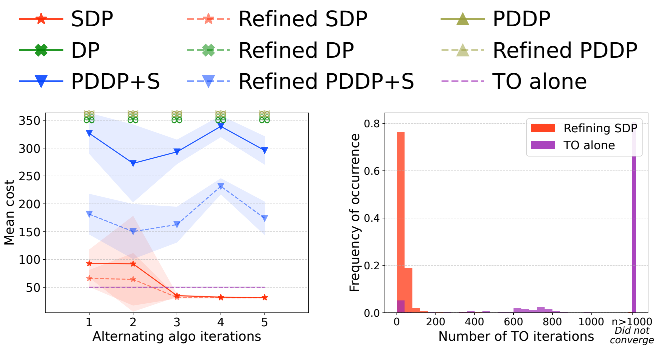
The alternating loop (Algo 1, with $n_{algo} > 1$) is tested on harder tasks, where the TO solver needs good initial guesses. After the first training iteration, studied in the previous subsection, new trajectories are collected using the policy to generate initial guesses for the TO solver. We refer to the process of doing TO on a trajectory produced by a policy as *refining*. As shown in Fig. 2, only SDP can solve the inverted double pendulum task, while DP consistently fail to operate torque control, and both PDDP and PDDP+S strongly diverge. Not only does SDP lead to more optimal trajectories compared to TO alone, but it also drastically reduces the solving time. On this challenging task, without a proper initial guess, the TO solver fails to converge 80%



(a) *UR5(r_{init}=0,r_{tgt}=0)* (b) *UR5(r_{init}=0,r_{tgt}=1)*

(c) Quadrotor without obstacles (d) Inverted single pendulum

Fig. 1: **Policy learning capacities.** Mean cost on test instances *w.r.t* n_{traj}, the number of trajectories in the dataset. Curves with the same color but different line styles differ in the number of training epochs n_{pl}. SDP reaches the performance of the TO solver on all tasks, even when trained with very few trajectories, while DP needs between 5 to 10 times more trajectories, and both PDDP and PDDP+S are very unstable. Our SDP successfully operates the inverted pendulum with torque control (actually, on this task $n_{pl} = 10^3$ and $n_{pl} = 2 \cdot 10^3$).



(a) Mean cost *w.r.t.* algo iters, both policy and *refined*, *i.e.* TO starting from the policy solution.

(b) Histogram of the number of TO iterations taken to converge, with SDP and without.

T_a	1	2	4	6	**9**	12	15	TO Alone
Policy mean cost	31.7	31.7	31.8	32.5	33.5	85.1	137	-
Refined mean cost	31.3	31.3	31.2	31.7	32.3	48.1	59.7	50.1
Traj Opt iters	27	35	35	43	78	405	690	896
Rollout time (s)	9.2	4.7	2.4	1.6	1.1	0.80	0.66	-
Solving time (s)	0.22	0.30	0.29	0.36	0.56	2.5	4.4	4.1
Total time (s)	9.4	5.0	2.6	1.9	**1.6**	3.3	5.1	4.1

(c) **Performances at inference when varying T_a.** The first two lines report the average trajectory cost J. The policy is the SDP one obtained at the end of the training, shown in **(a)**. The third line is the mean number of TO iterations taken to converge when refining (early stopped at 1000). As a reference, the last column shows the cost and solving time when TO is not warm-started. For our setup, $T_a = 9$ appears to be a good trade-off between policy rollout time and TO solving time.

Fig. 2: **Alternating algorithm performances on the inverted double pendulum.** For this task, we use $T_h = 16$ and at training time $T_a = 4$ (Algo 1, line 6), $n_{traj} = 30$ and $n_{pl} = 3 \cdot 10^3$ for all methods. **(a)** shows the evolution of the mean cost during training; after 3 iterations of the alternating loop, our SDP leads to near-optimal trajectories while all other methods fail (on this task, a cost higher than 200 corresponds to a complete fail). As the policy improves, the trajectories collected get more optimal, in a virtuous cycle. After 5 iterations, refining policy trajectories takes only 35 TO iterations on average, with the full histogram **(b)**. The action length T_a can be increased to accelerate the policy, inducing a trade-off between policy inference time and TO solving time, as reported in table **(c)**.

of the time, while refining from SDP takes 35 iterations on average. The optimality of the trajectories produced by SDP depends on T_a, the number of actions applied before replanning. As shown in Table 2c, with $T_a = 1$, refining a trajectory requires on average 0.22 seconds, compared to 4.1 seconds when solving from scratch (actually, the gap could be even wider, as we early stop after 1000 iterations). $T_a = 9$ appears to be a good trade-off, with a policy rollout time of 1.1 seconds and a solving time of 0.56 seconds, but this depends on the GPU and CPU used.

SDP and DP are further evaluated on the most challenging variants of the UR5 task, using various action lengths. SDP consistently shows great control, even with a very long action length $T_a = 63$, which implies a fast inference time. On these tasks too, warm starting the TO solver with the policy drastically reduces the number of TO iterations, and leads to better trajectories, as illustrated by Fig. 3e and Fig. 3f.

E. *Tasks involving constraints*

$\mathcal{L}^{SDP}$ does not have an explicit term to handle constraints; still, we experimented with adding obstacles to the quadrotor task. On a constrained OCP, reporting an average cost J is meaningless, because when the constraints are not satisfied, J should be $+\infty$. Fig. 4 illustrates the performances of SDP and DP. In particular, the compounding error issue of DP is visible: from a small variation, DP may go straight into an obstacle, thus violating the constraint. This may be due to $\mathcal{D}$ not containing trajectories where the quadrotor gets too close to an obstacle, so getting close to an obstacle is out of distribution, and DP gets lost. On the contrary, we observe that SDP adapts smoothly. On all trajectories, SDP exhibits great precision, while DP nearly never finishes at the target position.

On the layout with 9 obstacles, the average solving time of the TO solver with a naive initial guess is 16.3 seconds, but refining the SDP trajectory only requires 7.8 seconds, while with DP it takes 9.0 seconds. On this very challenging task for local optimizers, our SDP reduces the solving time by 53%, thus showing promising results to combine the strengths of PL and TO.

V. Discussion and Related Works

Numerous works have been published on diffusion-based policies. Close to our work, [12] trains a diffusion model on trajectories obtained from optimal control (DiffuSolve), and,

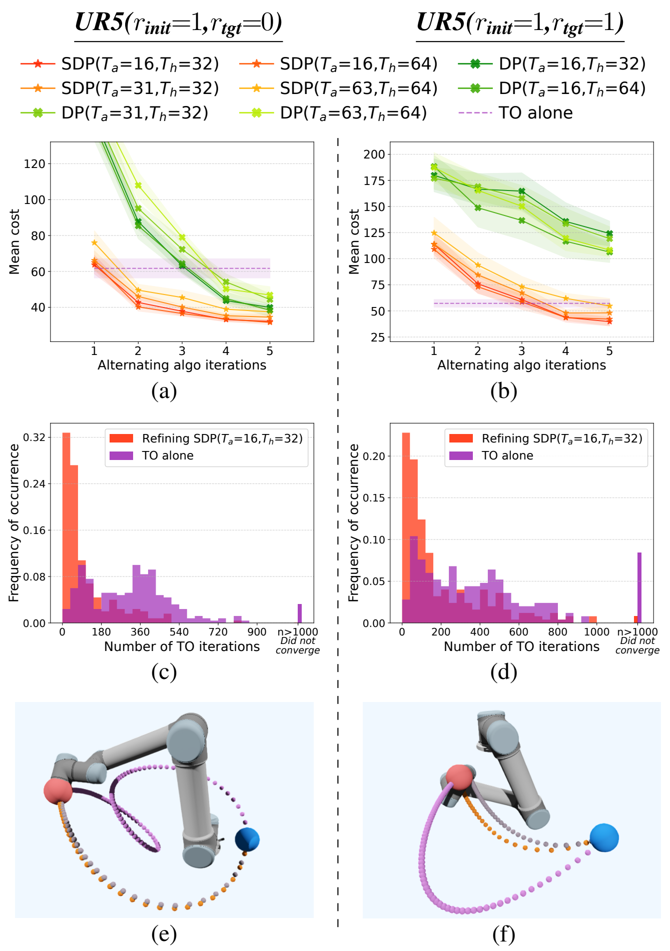


Fig. 3: **Alternating algorithm performances on the UR5.** SDP and DP are tested on the most challenging variants of the UR5 task, *UR5($r_{init}=1,r_{tgt}=0$)* (left column), and *UR5($r_{init}=1,r_{tgt}=1$)* (right column). To study the impact of the prediction horizon, both methods are tested with $T_h = 32$ and $T_h = 64$, and various action lengths T_a: 16, 31, and 63 (as $T_a \leq T_h - T_o$ and $T_o = 1$). For *UR5(r_{init}=1,r_{tgt}=0)*, n_{traj} is 50 in the first three iterations and 100 in the last ones, for *UR5(r_{init}=1,r_{tgt}=1)* n_{traj} is doubled, in both cases $n_{pl} = 10^4$. SDP solves both tasks, even with a large number of applied actions ($T_a = 63$), leading to fast inference time. SDP finds good initial guesses that the solver can quickly refine. **(c)** and **(d)** are histograms of the number of TO iterations taken to converge, with SDP and without. **(e)** and **(f)** show an instance of each task, in blue the initial position of the end-effector, in red the goal, in pink trajectories obtained by the TO solver alone, in orange SDP trajectories, in gray refined trajectories. SDP trajectories are more direct, while the TO solver may be stuck in local minima.

for constrained tasks, a penalty term is added to the training loss (DiffuSolve+). In terms of the present paper, DiffuSolve is *refined* DP (conditioned as in [7]) with $n_{algo} = 1$ (no further collecting trajectories), so it corresponds to DP as studied in the experiment section IV-C. Adding a penalty term to SDP for constrained problems, as DiffuSolve+, could be an interesting extension of our work, and we leave it as future work.

Adding a reinforcement learning (RL) component could be promising to improve the generality of our approach. On the inverted double pendulum (Fig. 2) and the more challenging UR5 tasks (Fig. 3), SDP worked thanks to the *ArgminCost* term (Algo 1 line 7), keeping the best trajectory between the policy-refined and the TO alone. This simple filter was sufficient here, but for more challenging tasks, advanced techniques from offline-RL, relying on a learned critic to better exploit the collected dataset, could be leveraged. By weighting actions to imitate based on their estimated value, the policy would avoid bad local minima, as proposed in recent works [8], [10].

To reduce the number of human-collected trajectories on real hardware, [13] proposes collecting a few trajectories using a VR headset and transferring them to a simulator using a TO solver, thereby minimizing the distance between the recorded data and the simulated control, while respecting physical constraints. By using a gradient-based TO solver in this pipeline, one could get feedback gains and use SDP instead of DP, to further reduce the number of trajectories needed, improve the policy precision, and recover all the advantages of SDP.

VI. Conclusion

In this paper, we have introduced Sobolev diffusion policy, a framework that combines diffusion policies with trajectory optimization methods, leveraging first-order information for faster convergence, improved sample efficiency, higher policy precision, and longer prediction horizons. This approach empowers TO solvers with an expressive policy to generate initial guesses, significantly reducing the solving time and leading to better solutions. Our extensive set of experiments suggests promising deployment on real robots.

Acknowledgments

This work has received support from the French government, managed by the National Research Agency, through the INEXACT project (ANR-22-CE33-0007-01) and under the France 2030 program with the reference "PR[AI]RIE-PSAI" (ANR-23-IACL-0008). This work was also supported by the European Union through the AGIMUS project (GA no.101070165). Views and opinions expressed are those of the author(s) only and do not necessarily reflect those of the European Union or the European Commission. Neither the European Union nor the European Commission can be held responsible for them.

References

[1] S. Levine and V. Koltun, "Guided policy search," in *International conference on machine learning*. PMLR, 2013.
[2] J. Sohl-Dickstein, E. Weiss, N. Maheswaranathan, and S. Ganguli, "Deep unsupervised learning using nonequilibrium thermodynamics," in *International conference on machine learning*. PMLR, 2015.
[3] J. Ho, A. Jain, and P. Abbeel, "Denoising diffusion probabilistic models," *Advances in neural information processing systems*, vol. 33, 2020.
[4] Y. Song, J. Sohl-Dickstein, D. P. Kingma, A. Kumar, S. Ermon, and B. Poole, "Score-based generative modeling through stochastic differential equations," in *9th International Conference on Learning Representations, ICLR 2021*, 2021.
[5] Y. Lipman, R. T. Chen, H. Ben-Hamu, M. Nickel, and M. Le, "Flow matching for generative modeling," in *11th International Conference on Learning Representations, ICLR 2023*, 2023.

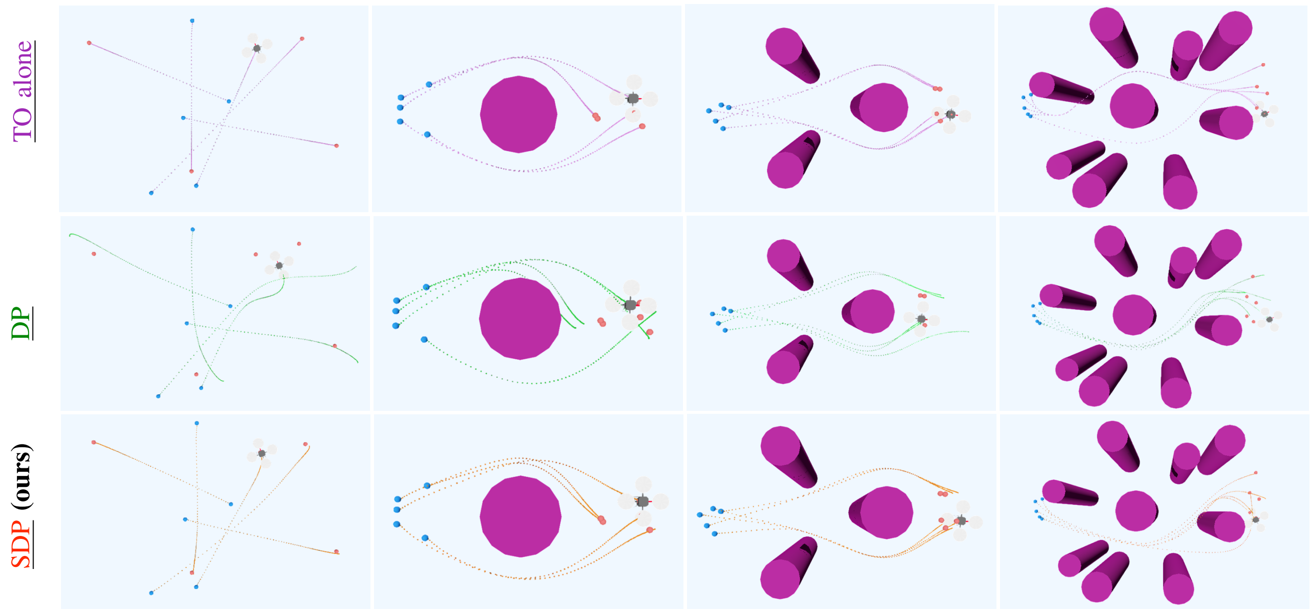


Fig. 4: **Constrained task - Quadrotor with obstacles.** SDP, third row, is compared against DP, second row, and TO as a reference on the first row. We test four variants of the quadrotor task, starting from zero obstacles to nine obstacles. Each illustration is done using 5 test instances, hence 5 trajectories (in blue, the initial position, in red, the target one). Here we fix $T_h = 32$ and $T_a = 16$. On the layout with 9 obstacles, we use $n_{algo} = 5$, $n_{pl} = 10^4$, and n_{traj} gradually increase at each iteration, starting at 30, ending at 100. On the other layouts (up to 3 obstacles), $n_{algo} = 1$ and $n_{traj} = 50$, no need to further collect trajectories. For the training time, $n_{pl} = 10^4$ when there are no obstacles, then $n_{pl} = 2 \cdot 10^4$ when there are 1 or 3 obstacles. The compounding error issue of DP is visible on the second row: when there are no obstacles, whenever the quadrotor starts to deviate from a straight trajectory, DP only gets worse. When there is one obstacle, if the quadrotor gets too close to the obstacle, DP will not be able to recover, while SDP adapts smoothly. On all trajectories, SDP exhibits great precision, while DP nearly never finishes at the target position.

[6] M. Janner, Y. Du, J. Tenenbaum, and S. Levine, “Planning with diffusion for flexible behavior synthesis,” in *International Conference on Machine Learning*. PMLR, 2022.

[7] A. Ajay, Y. Du, A. Gupta, J. Tenenbaum, T. Jaakkola, and P. Agrawal, “Is conditional generative modeling all you need for decision-making?” *arXiv preprint arXiv:2211.15657*, 2022.

[8] Z. Wang, J. J. Hunt, and M. Zhou, “Diffusion policies as an expressive policy class for offline reinforcement learning,” *arXiv preprint arXiv:2208.06193*, 2022.

[9] C. Chi, S. Feng, Y. Du, Z. Xu, E. Cousineau, B. Burchfiel, and S. Song, “Diffusion policy: Visuomotor policy learning via action diffusion,” *arXiv preprint arXiv:2303.04137*, 2023.

[10] C. Lu, H. Chen, J. Chen, H. Su, C. Li, and J. Zhu, “Contrastive energy prediction for exact energy-guided diffusion sampling in offline reinforcement learning,” in *International Conference on Machine Learning*. PMLR, 2023.

[11] S. Park, Q. Li, and S. Levine, “Flow q-learning,” *arXiv preprint arXiv:2502.02538*, 2025.

[12] A. Li, Z. Ding, A. B. Dieng, and R. Beeson, “Diffusolve: Diffusion-based solver for non-convex trajectory optimization,” *arXiv preprint arXiv:2403.05571*, 2024.

[13] L. Yang, H. Suh, T. Zhao, B. P. Graesdal, T. Kelestemur, J. Wang, T. Pang, and R. Tedrake, “Physics-driven data generation for contact-rich manipulation via trajectory optimization,” *arXiv preprint arXiv:2502.20382*, 2025.

[14] J. Carpentier and N. Mansard, “Analytical derivatives of rigid body dynamics algorithms,” in *Robotics: Science and systems (RSS 2018)*, 2018.

[15] F. de Avila Belbute-Peres, K. Smith, K. Allen, J. Tenenbaum, and J. Z. Kolter, “End-to-end differentiable physics for learning and control,” *Advances in neural information processing systems*, vol. 31, 2018.

[16] Q. Le Lidec, I. Kalevatykh, I. Laptev, C. Schmid, and J. Carpentier, “Differentiable simulation for physical system identification,” *IEEE Robotics and Automation Letters*, vol. 6, no. 2, 2021.

[17] Q. L. Lidec, L. Montaut, Y. de Mont-Marin, F. Schramm, and J. Carpentier, “End-to-end and highly-efficient differentiable simulation for robotics,” *arXiv preprint arXiv:2409.07107*, 2024.

[18] W. Li and E. Todorov, “Iterative linear quadratic regulator design for nonlinear biological movement systems,” in *First International Conference on Informatics in Control, Automation and Robotics*, vol. 2. SciTePress, 2004.

[19] D. H. Jacobson and D. Q. Mayne, “Differential dynamic programming,” 1970.

[20] I. Mordatch, E. Todorov, and Z. Popović, “Discovery of complex behaviors through contact-invariant optimization,” *ACM Transactions on Graphics (ToG)*, vol. 31, no. 4, 2012.

[21] W. Jallet, A. Bambade, E. Arlaud, S. El-Kazdadi, N. Mansard, and J. Carpentier, “ProxDDP: Proximal constrained trajectory optimization,” *IEEE Transactions on Robotics*, 2025.

[22] E. Dantec, M. Taix, and N. Mansard, “First order approximation of model predictive control solutions for high frequency feedback,” *IEEE Robotics and Automation Letters*, vol. 7, no. 2, 2022.

[23] I. Mordatch and E. Todorov, “Combining the benefits of function approximation and trajectory optimization.” in *Robotics: Science and Systems*, vol. 4, 2014.

[24] Q. Le Lidec, W. Jallet, I. Laptev, C. Schmid, and J. Carpentier, “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.

[25] P. Y. Simard, Y. A. LeCun, J. S. Denker, and B. Victorri, “Transformation invariance in pattern recognition—tangent distance and tangent propagation,” in *Neural networks: tricks of the trade*. Springer, 2002.

[26] W. M. Czarnecki, S. Osindero, M. Jaderberg, G. Swirszcz, and R. Pascanu, “Sobolev training for neural networks,” *Advances in neural information processing systems*, vol. 30, 2017.

[27] S. Ross, G. Gordon, and D. Bagnell, “A reduction of imitation learning and structured prediction to no-regret online learning,” in *Proceedings of the fourteenth international conference on artificial intelligence and statistics*. JMLR Workshop and Conference Proceedings, 2011.

[28] J. Carpentier, G. Saurel, G. Buondonno, J. Mirabel, F. Lamiraux, O. Stasse, and N. Mansard, “The pinocchio c++ library – a fast and flexible implementation of rigid body dynamics algorithms and their analytical derivatives,” in *IEEE International Symposium on System Integrations (SII)*, 2019.

[29] W. Jallet, A. Bambade, S. El Kazdadi, J. Carpentier, and

M. Nicolas, “aligator.” [Online]. Available: https://github.com/Simple-Robotics/aligator

[30] W. Jallet, “Candlewick,” 2025. [Online]. Available: https://github.com/Simple-Robotics/candlewick

[31] A. Q. Nichol and P. Dhariwal, “Improved denoising diffusion probabilistic models,” in *International conference on machine learning*. PMLR, 2021.