Accelerating trajectory optimization with Sobolev-trained diffusion policies

Théotime Le Hellard[1,⋆][0009−0008−4880−6341], Franki Nguimatsia Tiofack[1,⋆][0009−0006−4227−2241], Quentin Le Lidec[1,2][0000−0001−7973−1030], and Justin Carpentier[1][0000−0001−6585−2894]

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

Abstract. Trajectory Optimization (TO) solvers exploit known system dynamics to compute locally optimal trajectories through iterative improvements. A downside is that each new problem instance is solved independently; therefore, convergence speed and quality of the solution found depend on the initial trajectory proposed. To improve efficiency, a natural approach is to warm-start TO with initial guesses produced by a learned policy trained on trajectories previously generated by the solver. Diffusion-based policies have recently emerged as expressive imitation learning models, making them promising candidates for this role. Yet, a counterintuitive challenge comes from the local optimality of TO demonstrations: when a policy is rolled out, small non-optimal deviations may push it into situations not represented in the training data, triggering compounding errors over long horizons. In this work, we focus on learning-based warm-starting for gradient-based TO solvers that also provide feedback gains. Exploiting this specificity, we derive a first-order loss for Sobolev learning of diffusion-based policies using both trajectories and feedback gains. Through comprehensive experiments, we demonstrate that the resulting policy avoids compounding errors, and so can learn from very few trajectories to provide initial guesses reducing solving time by 2× to 20×. Incorporating first-order information enables predictions with fewer diffusion steps, reducing inference latency.

Keywords: Machine Learning in Robotics · Control Theory and Optimization · Diffusion Models

1 Introduction

Given an instance of a control problem and an initially proposed sequence of states and controls, trajectory optimization (TO) methods iteratively refine the trajectory to meet problem constraints and minimize an associated cost function [12, 22, 31]. Being local methods, their solving time and the quality of the solution found directly depend on the initial trajectory. To complement TO

⋆ Equal contributions

with learning approaches, a natural application is to train a global policy from a batch of solved instances [10, 19, 20, 27]. The resulting policy can then either be deployed or used to warm-start the solver on other instances. In this paper, we study interplay loops alternating between collecting trajectories and training, close to DAgger [32]. The policy provides initial guesses, the TO solver refines them towards locally optimal solutions, which are then used to train the policy.

For the imitation learning part, we leverage diffusion models [11, 34, 35] for their generalizing capacities. Initially developed for image generation, diffusion-based policies [1, 6, 15, 38] are now the favored approach in imitation learning, driving significant investments in robotics. While commonly used to imitate human demonstrations, in this work, we use diffusion models to imitate trajectories generated by a solver [21, 39].

We focus on gradient-based TO solvers [12, 13, 22, 26], which leverage the derivatives of underlying dynamics algorithms [4, 9, 18] and the development of differentiable simulators [3, 17, 23]. These solvers base their iterative steps on the computation of feedback gains, first-order information that can then be exploited at no additional cost [8]. In particular, they can help train a policy to imitate trajectories generated by gradient-based TO [2, 16, 27], using derivatives matching [7, 25, 30, 33, 36]. We use the Sobolev learning [7] terminology, a first-order supervised method that accelerates convergence and mitigates overfitting, reducing the number of trajectories needed for the policy to generalize.

We propose adapting the Sobolev learning formulation to diffusion-based policies, in connection with a gradient-based TO solver. Our framework alternates between collecting trajectories using a gradient-based TO solver and first-order policy learning. Compared to non-diffusion-based first-order policies [16, 27], our method scales to more complex tasks, and compared to non-Sobolev methods, it requires fewer trajectories. The resulting policy can predict over longer horizons, with fewer diffusion steps, hence accelerating diffusion inference, which in our context corresponds to the latency of predicting an initial guess. Remarkably, it exhibits great 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. In particular, we highlight a counterintuitive challenge when learning from TO-generated trajectories: their local optimality entails no small missteps, which might exacerbate the risk of compounding errors. In fact, when training a diffusion policy from human demonstrations, it is common practice to include non-optimal, or even failing, trajectories, a recommendation opposite to learning from locally optimal trajectories. Close to our work, [21] had to generate hundreds of thousands of trajectories in order to train diffusion policies from locally optimal trajectories. In contrast, our Sobolev method benefits from the feedback gains, and only needs at most few hundred generated trajectories to provide effective initial guesses.

In this work, we make the following contributions:

- we introduce a first-order loss for diffusion-based policy learning, to efficiently train with trajectories generated by gradient-based TO

- we propose an interplay algorithm, alternating between collecting trajectories and training, to solve hard tasks that TO struggles with without proper initial guesses,
- 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. 2 provides background on trajectory optimization and diffusion, including prior works and challenges. Sec. 3 introduces our Sobolev approach to diffusion policies. Sec. 4 presents our set of experiments. Finally, Sec. 5 discusses related work, limitations, and future ideas.

2 Background

2.1 Optimal control problems

We consider discrete-time dynamical systems. At time step t, we denote by x_t the state of the system and u_t the control input. The system dynamic is defined by the differentiable function f, such that $x_{t+1} = f(x_t, u_t)$. In the context of robotics, x_t is composed of joint positions q_t and velocities v_t. For indices $t_1 \leq t_2$, we denote by $x_{t_1:t_2}$ the matrix whose rows are $x_{t_1}, x_{t_1+1}, \ldots, x_{t_2}$. The state and control trajectories are $X = x_{0:T}$ and $U = u_{0:T-1}$.

We consider the following constrained optimal control problems (OCPs):

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

ξ denotes task parameters specifying the initial position $\hat{x}(\xi)$ and parameterizing both the differentiable constraints h_t, h_T and the twice differentiable cost J. The latter is defined as the sum of stage costs ℓ_t and a terminal cost ℓ_T.

2.2 Gradient-based trajectory optimization

To efficiently find local optimum to OCPs, gradient-based TO solvers, such as iterative linear quadratic regulator (iLQR) [22] and differential dynamic programming (DDP) [12, 13], leverage the derivatives of stage costs, constraints, and of the dynamics function. In the case of robotic systems, these solvers exploit the derivatives of rigid body algorithms [4, 9] and differentiable physics simulators [3, 17, 23]. The present paper uses iLQR-like methods, yielding local *feedback gains* that we exploit in the policy learning part through Sobolev training. In practice, we use the ProxDPP variant [13], which 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 $J_t(x_{\geq t}, u_{\geq t}) = \sum_{i \geq t}^{T-1} \ell_i(x_i, u_i) + \ell_T(x_T)$ 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, these solvers 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 estimates of $\frac{\partial x_{t+1}}{\partial x_t}$ using the chain rule:

$$\frac{\partial x_{t+1}}{\partial x_t} = \frac{\partial f}{\partial x_t}(x_t, u_t) + \frac{\partial f}{\partial u_t}(x_t, u_t) \cdot \frac{\partial u_t}{\partial x_t} \tag{2}$$

2.3 Policy learning with diffusion models

Related works. Policy learning trains policy π_θ, with parameters $\theta \in \Theta$, to minimize the constrained objective function (1) in expectation over task parameters $\xi \sim \mathcal{P}$. Common Gaussian stochastic policies learn a conditional distribution of control inputs given the current state: $u_t \sim \pi(u_t | x_t; \xi)$. Recently, Janner et al [15] instead proposed training diffusion models to control robots.

Diffusion models [11, 24, 34, 35] are probabilistic approaches that seek to generate elements from a desired distribution $p_{\mathcal{D}}$, given a training dataset $\mathcal{D} = \{\tau_0\}$. Diffusion proceeds by progressively disrupting data through noise injection, then a model is trained to reverse this process for sample generation. For diffusion policies [1, 6, 15] $\mathcal{D}$ is built using successful trajectories of a desired control task. Trajectories are sequences of states x_t and/or controls u_t, with definitions varying across papers. For unified notations, we denote by a_t the action variable generated by the diffusion model, and by o_t the observation variable used to condition the diffusion process, *e.g.*, the current state. When a_t is x_t, or even q_t, u_t is deduced by a low-level controller (PD control, IK solver, etc).

Diffusion policies are trained to model the distribution of trajectory chunks. Each training sample is a sequence $\tau_0 = a_{t_1:t_1+T_h-1}$, where t_1 is the start time and T_h is the prediction horizon. For example, in robot control, τ_0 may be a sequence of joint positions over the next T_h time steps. The model learns to generate a T_h-step control sequence conditioned on task parameters ξ and the most recent T_o states (including the current state), i.e., $o_{t-T_o+1:t}$, where T_o is the history length. At inference time, the policy predicts a T_h-step sequence, executes the first T_a steps, and then replans, where T_a is the action length.

While diffusion models may be used as black box generative models, the present paper modifies their training loss by adding first-order information, so for completeness we detail an introduction to the diffusion framework used, denoising diffusion probabilistic models (DDPM) [11].

DDPM. A diffusion process is defined 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 $\beta_1, ..., \beta_K$ the noising schedule. A sample τ_k can be directly drawn from τ_0, by sampling a noise $\varepsilon \sim \mathcal{N}(0, \mathbf{I})$ and

$$\tau_k := \sqrt{\bar{\alpha}_k}\tau_0 + \sqrt{1-\bar{\alpha}_k}\varepsilon \tag{3}$$

with $\bar{\alpha}_k := \prod_{s=1}^{k} 1-\beta_s$. Thus, forward posteriors conditioned on τ_0 are tractable:

$$p_{\mathcal{D}}(\tau_{k-1}|\tau_k, \tau_0) = \mathcal{N}(\tau_{k-1}; \tilde{\mu}_k(\tau_k, \tau_0), \tilde{\beta}_k\mathbf{I}) \tag{4}$$

$$\tilde{\mu}_k(\tau_k, \tau_0) = \frac{\sqrt{\bar{\alpha}_{k-1}}\beta_k}{1-\bar{\alpha}_k}\tau_0 + \frac{\sqrt{\bar{\alpha}_k}(1-\bar{\alpha}_{k-1})}{1-\bar{\alpha}_k}\tau_k \text{ and } \tilde{\beta}_k = \frac{1-\bar{\alpha}_{k-1}}{1-\bar{\alpha}_k}\beta_k \tag{5}$$

In the other direction, 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 $\mu_\theta(\tau_k, k)$.

A neural network τ_θ is trained 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 $\tilde{\mu}_k$ (5) to sample τ_{k-1}, *i.e.* $\tau_{k-1} \sim \mathcal{N}(\tilde{\mu}_k(\tau_k, \tau_\theta(\tau_k, k)), \tilde{\beta}_k\mathbf{I})$. For diffusion policies, $\tau_0 = a_{t_1:t_1+T_h-1}$, and τ_θ is additionally conditioned on ξ and $o_{hist} = o_{t:t+T_o-1}$. To further condition the generative process, [15] uses inpainting, overwriting $\tau_k[1:T_o] = a_{hist}$ at each denoising step, and if applicable $\tau_k[T_h] = x_{target}$ (a target position). Finally, DDPM authors [11] recommend discarding the weighting when averaging over $k \in [1, K]$; using (3):

$$\mathcal{L}^{Diff} = \mathop{\mathbb{E}}_{(\xi, \tau_0, o_{hist}), \varepsilon, k} \left[\|\tau_0 - \tau_\theta(\tau_k, k, \xi, o_{hist})\|^2 \right] \tag{6}$$

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 we will stick with the τ_0 parametrization. Since τ_0 can be deduced from ε as $\tau_0 = \frac{1}{\sqrt{\bar{\alpha}_k}}(\tau_k - \sqrt{1-\bar{\alpha}_k}\varepsilon)$, everything could be derived using ε_θ instead of τ_θ.

The compounding error challenge. A bottleneck in diffusion-based policies is the number of trajectories needed. In imitation learning, not having enough trajectories to cover the task space comes with the compounding error risk. In particular, if the dataset does not contain missteps, small deviations during rollouts may lead the policy to unseen scenarios; without knowing how to recover, the policy may worsen the situation and enter a downward spiral. To avoid this effect, the authors of diffusion policy [6] later highlighted the need to include non-perfect trajectories in the dataset. While it may turn the data collection process non-intuitive, having to predict how the policy could fail, similar to the DAgger method [32], human teleoperated trajectories eventually bring enough diversity and imperfections. In contrast, when using solvers for data collection, the local

optimality may counterintuitively hurt. In a pioneer work on diffusion for control, decision diffuser [1] uses 10000 trajectories for the Kuka block stacking task. Close to our work, DiffuSolve [21] uses about 200k locally optimal trajectories for a quadrotor navigation task. While using a solver to generate demonstrations is usually cheaper, in [21], collecting 200k trajectories took 20 hours on 200 CPU cores. In this work, we will show how integrating the feedback gains by Sobolev learning mitigates this issue, exploiting the particularity of gradient-based TO.

3 Interplay between gradient-based TO solvers and diffusion-based policies

We propose using Sobolev training to enhance diffusion policies trained on trajectories generated by gradient-based TO. For complex tasks, our algorithm alternates between trajectory collection and training.

To recap all notations, a_t is the action variable, either x_t, u_t or a sub part, *e.g.* q_t. If a_t is not u_t, the latter is estimated by inverse methods. 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 T_o is the history length, with $x_{hist} = x_{t:t+T_o-1}$, $o_{hist} = o_{t:t+T_o-1}$, $a_{hist} = a_{t:t+T_o-1}$. The first T_o elements of τ_0 are the current and past actions (for inpainting), only then come the T_a used ones, hence $T_a \leq T_h - T_o$.

As explained Sec. 2.2, the TO solver produces feedback gains $\frac{\partial u_t}{\partial x_t}$, and by chaining with dynamics derivatives, we get $\frac{\partial x_{t+1}}{\partial x_t}$ (2). As a_t is either u_t, x_t, or a subpart of it, we have $\frac{\partial a_t}{\partial x_t}$, and by further chaining these derivatives:

$$\frac{\partial a_{t+h}}{\partial x_{t+o}} = \frac{\partial x_{t+o+1}}{\partial x_{t+o}} \cdot \frac{\partial x_{t+o+2}}{\partial x_{t+o+1}} \cdot \dots \cdot \frac{\partial x_{t+h}}{\partial x_{t+h-1}} \cdot \frac{\partial a_{t+h}}{\partial x_{t+h}} \tag{7}$$

for $t \in [0, T-1]$ and $0 \leq o \leq h < T_h$. Stacking derivatives gives $\frac{\partial \tau_0}{\partial x_{hist}} = \frac{\partial a_{t_1:t_1+T_h-1}}{\partial x_{t_1:t_1+T_o-1}}$.

3.1 Sobolev learning for first-order diffusion policies

In general, training a regression problem $f_\theta(x) = y$ in Sobolev spaces [7], also known as derivatives matching [25, 30, 36] or tangent propagation [33], consists of adding a first-order term in the regression loss, under a Frobenius norm:

$$\mathcal{L}^{Sobolev} = \mathbb{E}_x \left[\left\| f_\theta(x) - y \right\|^2 + \alpha_{Sob} \left\| \frac{\partial f_\theta(x)}{\partial x} - \frac{\partial y}{\partial x} \right\|_F^2 \right] \tag{8}$$

where $\frac{\partial y}{\partial x}$ is supposed known, and α_{Sob} weights the first-order term. Gradient descend of $\mathcal{L}^{Sobolev}$ on θ requires computing the expensive $\partial_{\theta,x} f_\theta$ Hessian. As an alternative, stochastic Sobolev training [7] samples n_{proj} random vectors v_i on the unit sphere (with $n_{proj} = 1$ turning out to be sufficient in our case), on which the gradients are projected:

$$\mathcal{L}^{Sobolev}_{Stochastic} = \mathbb{E}_x \left[\left\| f_\theta(x) - y \right\|^2 + \frac{\alpha_{Sob}}{n_{proj}} \sum_{i=1}^{n_{proj}} \left\| v_i^\top \cdot \frac{\partial y}{\partial x} - \frac{\partial}{\partial x}(v_i^\top \cdot f_\theta(x, \xi)) \right\|^2 \right] \tag{9}$$

[27] and [16] respectively use (8) and (9), to train a direct policy $\pi_\theta(x_t) = u_t$ using the feedback gains $\frac{\partial u_t}{\partial x_t}$.

One way to adapt (8) to diffusion policies could be to consider the whole generative process. Comparing the original τ_0 and its derivatives, to $\bar{\tau}_0^\theta$, the final output of the diffusion policy after all K denoising steps, and the chained derivatives of all K steps:

$$\mathcal{L}_{full}^{Sob+Diff} = \mathbb{E}_{(\xi, \tau_0, \frac{\partial \tau_0}{\partial x_{hist}}), \tau_K} \left[\left\| \tau_0 - \bar{\tau}_0^\theta \right\|^2 + \alpha_{Sob} \left\| \frac{\partial \tau_0}{\partial x_{hist}} - \frac{\partial \bar{\tau}_0^\theta}{\partial x_{hist}} \right\|^2 \right] \tag{10}$$

However, this opposes how diffusion models are typically trained, and it did not work in practice. Rather than training over the whole diffusion process at once, DDPM trains each denoising step separately, sampling ε and k, and training τ_θ 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 $\mathcal{L}^{Diff}$ (6) and $\mathcal{L}_{Stochastic}^{Sobolev}$ (9), we propose to guide the derivatives of each denoising step:

$$\boldsymbol{\mathcal{L}^{Sob+Diff}} = \mathbb{E}_{(\xi, \tau_0, \frac{\partial \tau_0}{\partial x_{hist}}), \varepsilon, k} \left[\left\| \tau_0 - \tau_0^{\theta,k} \right\|^2 + \frac{\alpha_{Sob}}{n_{proj}} \sum_{i=1}^{n_{proj}} \left\| v_i^\top \cdot \frac{\partial \tau_0}{\partial x_{hist}} - \frac{\partial}{\partial x_{hist}} \left(v_i^\top \cdot \tau_0^{\theta,k} \right) \right\|^2 \right] \tag{11}$$

Eq. (11) is not equivalent to Eq. (10), as it steers the derivatives of each step toward the desired derivatives of the full process. Moreover, the Sobolev weight α_{Sob} is fixed for all diffusion steps. This might be counterintuitive, as early steps (k close to K) are stochastic, while late steps (k close to 1) focus the distribution to one mode. But oppositely, for $k < K$, at inference, $\tau_0^{\theta,k} = \tau_\theta(\tau_k^\theta, k, \xi, o_{hist})$ has a dependence to x_{hist} through τ_k^θ, as it was obtained from a previous prediction. While one could define k-dependent weights, *e.g.* increasing $\alpha_{Sob}^k = \alpha_{Sob}\frac{k}{K}$ or decreasing $\frac{K-k}{K}$, we observed counteracting effects and let $\alpha_{Sob}^k = \alpha_{Sob}$.

3.2 Alternating loop

The full training procedure, Algorithm 1, repeats over n_{algo} iterations: (i) collect a dataset $\mathcal{D}$ of trajectories using TO ; (ii) train a policy on $\mathcal{D}$ using $\boldsymbol{\mathcal{L}^{Sob+Diff}}$ (11); (i') collect new trajectories, but using the policy to provide initial guesses to the solver. For policy rollout, Algorithm 2 iteratively generates T_h actions using the DDPM inference scheme, out of which T_a actions are rolled out. For simple tasks, collecting one fixed dataset $\mathcal{D}$ using TO is enough ($n_{algo} = 1$), but for harder ones, further collecting trajectories using the policy to provide initial guesses ($n_{algo} > 1$) helps reduce the solving time and can lead to better local minima. This is due to **three key choices in Alg. 1**: Line 7 calls the solver twice, both with and without warm-starting with the policy. Line 9 rejects trajectories where the solver failed to converge after n_{max} solving iterations. Finally, Line 2, we chose to reset $\mathcal{D}$ between each algorithm iteration. While more complex heuristics could be beneficial, through our experiments we found these designed choices to be effective. On hard tasks, using TO alone may lead to high *rejection*-rates (Line 9), so the policy reduces the total time needed to collect n_{traj} trajectories.

Algorithm 1: Training - Interplaying collection and learning

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 $\xi \sim \mathcal{P}$
    $X^0, U^0 = Interpolate(x_0, x_{target})$
    $X^\pi, U^\pi = \textbf{PolicyRollout}(\tau_\theta, \xi, T_a, K_{rollout})$
    $X, U, \frac{\partial u_t}{\partial x_t}, \frac{\partial x_{t+1}}{\partial x_t} = ArgminCost\,(\textbf{TO}(X^0, U^0, \xi),\ \textbf{TO}(X^\pi, U^\pi, \xi))$
    if both **TO** calls did not converge in n_{max} iterations
    then *reject - restart Line 4*
    else $\mathcal{D} \leftarrow \mathcal{D} \cup \{(X, U, \frac{\partial u_t}{\partial x_t}, \frac{\partial x_{t+1}}{\partial x_t})\}$
  // Policy learning
  for n_{pl} *epochs* **do**
    // Sample by batches
    Sample ξ, τ_0, x_{hist}, o_{hist} and derivatives in $\mathcal{D}$
    $\frac{\partial \tau_0}{\partial x_{hist}} = ChainRule(\frac{\partial x_{t+1}}{\partial x_t}, \frac{\partial u_t}{\partial x_t})$ // using (7)
    $k \sim \mathcal{U}(1, K_{train}),\ \varepsilon \sim \mathcal{N}(0, I)$
    // 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 $\mathcal{L}^{Sob+Diff}$ (11)
    $\tau_0^{\theta,k} = \tau_\theta(\tau_k, k, \xi, o_{hist})$
    Sample n_{proj} vectors v_i in the unit sphere
    Take gradient descent step on $\nabla_\theta \mathcal{L}^{Sob+Diff}(\theta)$

Algorithm 2: 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
    $\tau_0^{\theta,k} = \tau_\theta(\tau_k, k, \xi, o_{hist})$
    $\tau_{k-1} \sim \mathcal{N}(\tilde{\mu}_k(\tau_k, \tau_0^{\theta,k}), \tilde{\beta}_k \mathbf{I})$
  // 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 $u_{t+s-1} = Inverse(x_{t+s-1}, \tau_0[s])$ // e.g. PD control, RNEA etc
    $x_{t+s} = f(x_{t+s-1}, u_{t+s-1})$ // dynamics
  Update o_{hist} and a_{hist}
  $t = t + T_a$

Output: X, U

4 Experiments

In this section, we refer to our method as *Sob+Diff*, and evaluate it against alternative policy learning architectures. Projected DDP (*PDDP*) and its Sobolev variant (*PDDP+S*) from [16] use a similar interplay loop and correspond to using a direct MLP (multilayer perceptron) instead of a diffusion model. For simplicity, we will refer to PDDP and PDDP+S as *MLP* and *Sob+MLP*, respectively. *DiffuSolve* [21] trains diffusion policies over trajectories generated by a solver. While setup and implementation differ (in particular, they do not use a gradient-based solver), DiffuSolve is close to ablating the Sobolev component, which we refer to by *Diff* in this section. DiffuSolve does not have an interplay loop to further collect trajectories, but we unify the name of the method with the learning components used. The experiments are separated into two parts. n_{algo} is first fixed to 1 (no further collecting trajectories) to study how many trajectories are needed for the different policies to learn. Then, for harder tasks, we study the full algorithm ($n_{algo} > 1$), alternating between trajectory collection and policy learning, as described in Alg 1.

4.1 Experimental setup and implementation details

Our code will be made public upon paper acceptance. We use Pinocchio [5] and Aligator [14] to define and solve the constrained OCPs (1). For the learning part, we use PyTorch with the same conditional U-Net as [15] and [6], the DDPM scheduler is square cosine from improved DDPM [29] 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 $n_{max} = 1000$. While we find it beneficial to reset $\mathcal{D}$ after each iteration (Alg 1, Line 2), still, as the policy improves, we may choose to increase n_{traj}, since trajectories are faster to collect (as the rejection-rate Line 9 decreases). We let $n_{proj} = 1$, sampling only one vector for the stochastic Sobolev training, so training *Sob+Diff* on one epoch takes about twice as much time as *Diff*. We fix the hidden dimensions of the conditional U-Net to $[24, 24, 32, 32]$, resulting in approximately 300k parameters. We fixed the number of diffusion steps K to 5. By default, $T_h = 32$, $T_o = 1$ and $T_a = T_h - T_o = 31$.

To compare the training time fairly, we adapt the size E of an "*epoch*" to the method. For direct policies, predicting u_t given x_t (*e.g.*, *MLP*), $E = |\mathcal{D}| \times T$, 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.

All results are averaged over 5 random seeds, displaying confidence intervals. On each seed, the mean trajectory cost $J(X, U; \xi)$ is estimated using 50 random instances $\xi \sim \mathcal{P}$. 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: *Sob+Diff* (red), *Diff* (green), *MLP* (gold), and *Sob+MLP* (blue). As a reference, we evaluate the performance of TO with interpolation-based initial guesses, reported as *TO alone* (purple).

4.2 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 [14]. We use $\alpha_{Sob} = 1.0$, except for the quadrotor task where $\alpha_{Sob} = 10^{-3}$ 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 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, $a_t = u_t$.

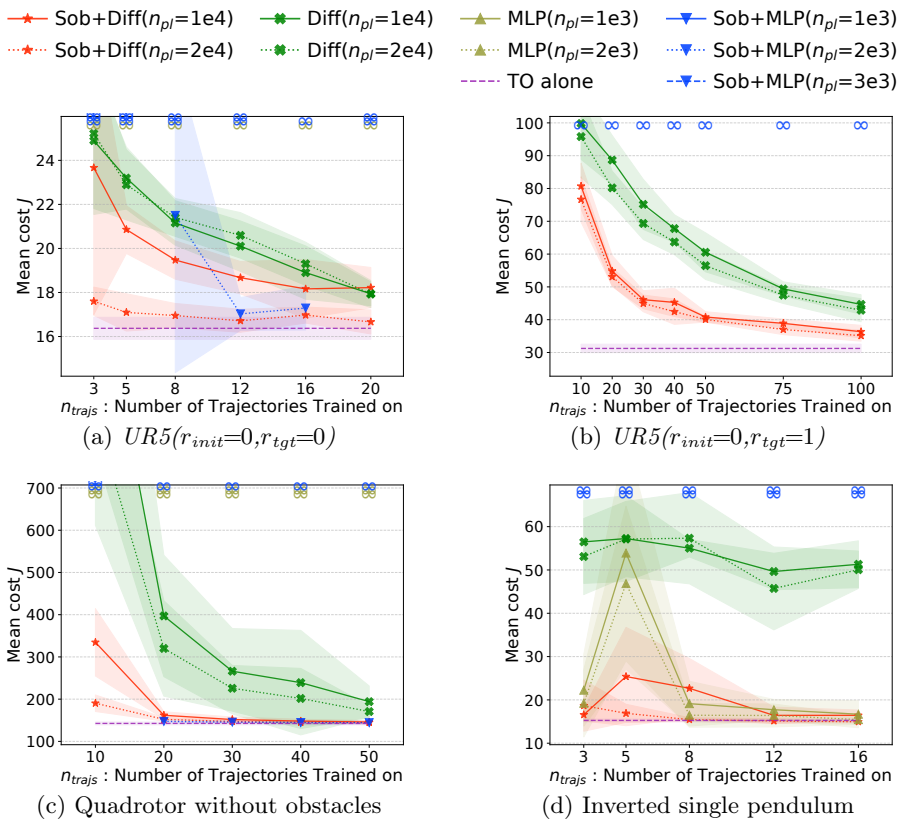
UR5. To gradually increase the complexity of the goal-reaching task with a UR5 robotic arm, we tried 4 variants. We define two boolean variables: *fully random init* r_{init}, whether the initial position is sampled in a small reasonable area (r_{init}=0) or anywhere in the robot space (r_{init}=1); 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 [16], PDDP+S (*i.e. Sob+MLP*) was tested on *UR5(*r_{init}=0,r_{tgt}=0*)*. q_t is composed of the 6 joint angles, and following [16], q_t is preprocessed to $(\cos q_t, \sin q_t)$, derivatives are adjusted accordingly (Alg 1, Line 13). For this task, we set $a_t = x_t$, and u_t is computed using RNEA [5] for Alg 1 Line 10, 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, $a_t = (dq_t, v_t)$.

4.3 Learning capacities and sample efficiency

This subsection focuses on the learning capacities, n_{algo} is fixed to 1 (*i.e.* no further trajectory collection), to see how many trajectories are needed for the policy to produce initial guesses with average costs close to typical ones the solver reaches without warm-starting. On Fig. 1, the sample efficiency is evaluated by plotting the average cost with respect to the dataset size $n_{traj} = |\mathcal{D}|$. Results are collected independently, with a fixed dataset for each run, for example, one run had 5 trajectories, another one had 5 more etc. To investigate the dependence on the number of training epochs n_{pl}, each method is evaluated at different times during training. Both *Sob+Diff* and *Diff* are evaluated at $n_{pl} = 10^4$ and $n_{pl} = 2 \cdot 10^4$ (except for the single pendulum, where 10^3 is enough), while *MLP* and *Sob+MLP* are evaluated at $n_{pl} = 10^3$ and $n_{pl} = 2 \cdot 10^3$. On all tasks, *Sob+Diff* reaches the average performance of the TO solver, achieving both

state control and torque control, even when trained with very few trajectories. Compared to *Diff*, *Sob+Diff* usually needs between 5 to 10 times fewer trajectories. On the inverted pendulum task, *Sob+Diff* succeeds with $n_{traj} = 3$, while *Diff* systematically fails to operate torque control. Overall, *MLP* and *Sob+MLP* 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, *Sob+MLP* is additionally evaluated with $n_{pl} = 3 \cdot 10^3$, it reveals that *Sob+MLP* only succeeds with $8 \leq n_{traj} \leq 16$ and $n_{pl} = 2 \cdot 10^3$, otherwise following into overfitting.



(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.** Plot mean cost $J(X, U; \xi)$ from (1) on test instances ξ *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}. *Sob+Diff* reaches the performance of the TO solver on all tasks, even when trained with very few trajectories, while *Diff* needs between 5 to 10 times more trajectories, and both *MLP* and *Sob+MLP* are very unstable. Our *Sob+Diff* successfully operates the inverted pendulum with torque control (actually, on this task $n_{pl} = 10^3$ and $n_{pl} = 2 \cdot 10^3$).

4.4 Full algorithm evaluation

The alternating loop (Alg 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 *Sob+Diff* can solve the inverted double pendulum task, while *Diff* consistently fails to operate torque control, and both *MLP* and *Sob+MLP* strongly diverge. Not only does *Sob+Diff* 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% of the time, while refining from *Sob+Diff* takes 35 iterations on average. The optimality of the trajectories produced by *Sob+Diff* 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 is 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.

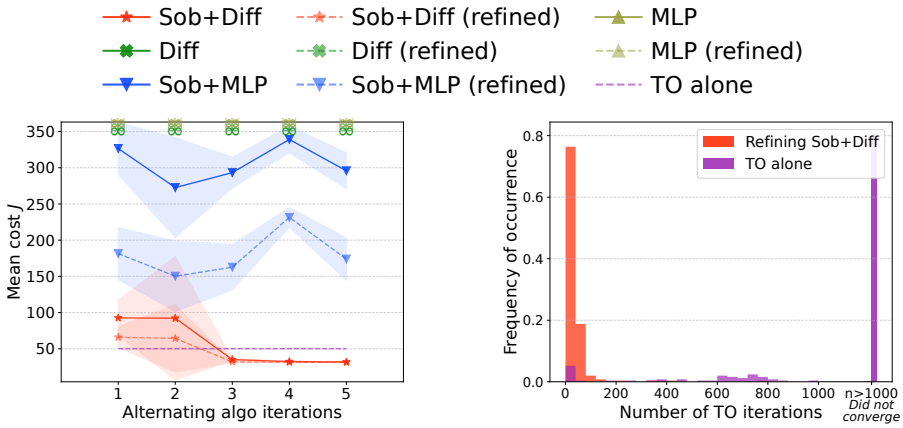
Sob+Diff and *Diff* are further evaluated on the most challenging variants of the UR5 task, using various action lengths. *Sob+Diff* consistently shows great control, even with a very long action length $T_a = 63$. While a small T_a may still be preferred for reactivity, we show the resilience of *Sob+Diff* to large T_a, for instance, due to limited computational resources. On these tasks, 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.

These experiments show that, given a TO solver and tasks it underperforms on, a learned policy can help to find initial guesses. We do not claim that these tasks are insolvable using dedicated modified iLQR algorithms [37].

4.5 Tasks involving constraints

$\mathcal{L}^{Sob+Diff}$ 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 harder, because when the constraints are not satisfied, J should be $+\infty$, hiding how much the constraints are violated. Fig. 4 illustrates the performances of *Sob+Diff* and *Diff*. In particular, the compounding error issue of *Diff* is visible: from a small variation, *Diff* may go straight into an obstacle, thus violating the constraint. This may be due to $\mathcal{D}$ containing trajectories brushing the obstacles without ever touching them, so touching an obstacle is out of distribution, and *Diff* gets lost. On the contrary, we observe that *Sob+Diff* adapts smoothly. *Sob+Diff* exhibits good precision on all trajectories, while *Diff* rarely finishes at the target position.

On the layout with 9 obstacles, the average solving time of the TO solver without warm-start is 16.3 seconds, while a diffusion policy reduces the solving time to 9.0 seconds, and *Sob+Diff* further reduces it to 7.8 seconds, *i.e.* by 53%.



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

(b) Histogram of the number of TO iterations taken to converge, with *Sob+Diff* to warm-start 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 *Sob+Diff* one obtained at the end of training from plot **(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: **Interplay algorithm on the inverted double pendulum.** For this task, we use $T_h = 16$ and at training time $T_a = 4$ (Alg 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 *Sob+Diff* 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)**.

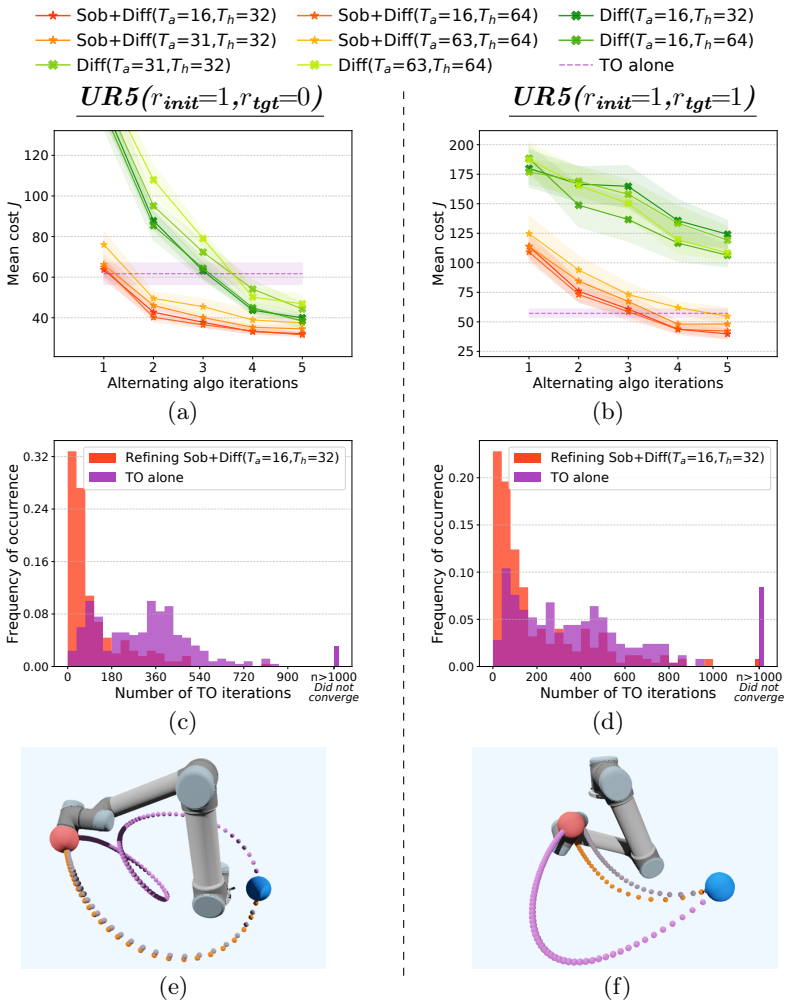


Fig. 3: **Interplay algorithm on the UR5.** *Sob+Diff* and *Diff* are evaluated on the challenging *UR5*(r_{init}=1,r_{tgt}=0) (left column), and *UR5*(r_{init}=1,r_{tgt}=1) (right column). Methods are tested with varying prediction horizons, $T_h = 32$ and $T_h = 64$, and action lengths T_a: 16, 31, and 63 (as $T_a \leq T_h - T_o$ and $T_o = 1$). For *UR5*(r_{init}=1,r_{tgt}=0), $n_{traj} = 50$ in the first three iterations and 100 after, for *UR5*(r_{init}=1,r_{tgt}=1) n_{traj} is doubled, in both cases $n_{pl} = 10^4$. *Sob+Diff* solves both tasks, even with a large number of applied actions ($T_a = 63$) for fast inference. **(c)** and **(d)** report the number of TO iterations taken to converge, with *Sob+Diff* and without, showing that *Sob+Diff* helps the solver quickly converge. **(e)** and **(f)** show an instance of each task, in blue the initial position, in red the goal, in pink trajectories obtained by TO alone (representing trajectories used at the beginning of training), in orange *Sob+Diff* trajectories, in gray refined ones. *Sob+Diff* trajectories are more direct, avoiding local minima.

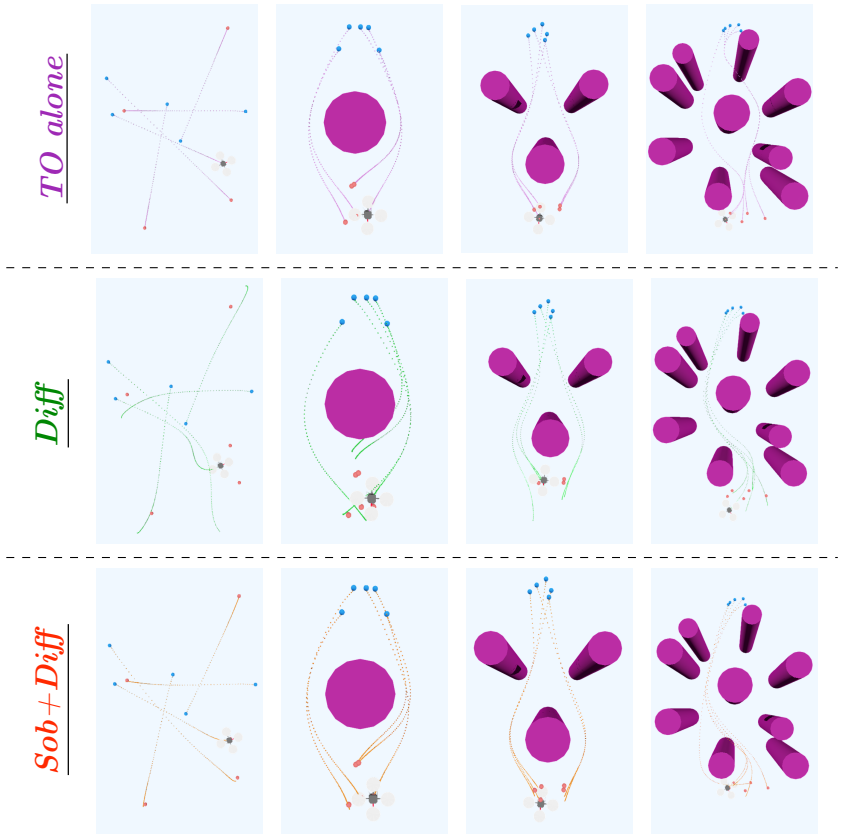


Fig. 4: **Constrained task - Quadrotor with obstacles.** *Sob+Diff*, third row, is compared against *Diff*, second row, and TO as a reference on the first row. We test four variants of the quadrotor task, starting from zero obstacles to nine. The obstacles are fixed, for changing number of obstacles one could use a transformer or a recurrent policy. 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, from 30 to 100. On the other layouts (up to 3 obstacles), $n_{algo} = 1$ and $n_{traj} = 50$, no further collecting 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 *Diff* is visible on the second row: when there are no obstacles, whenever the quadrotor starts to deviate from a straight trajectory, *Diff* only gets worse. When there is one obstacle, if the quadrotor gets too close to the obstacle, *Diff* is not able to recover, while *Sob+Diff* adapts smoothly. On all trajectories, *Sob+Diff* exhibits great precision, while *Diff* nearly never finishes at the target position.

5 Discussion and Related Works

We focused on using diffusion policies over trajectories generated by gradient-based TO. Close to our work, [21] trains a diffusion model on trajectories obtained from optimal control (DiffuSolve), and, for constrained tasks, a penalty term is added (DiffuSolve+). As detailed Sec. 4, in our terms, DiffuSolve is close to *Diff* with $n_{algo} = 1$ (no further collecting trajectories). Adding a similar penalty term to *Sob+Diff* for constrained problems is interesting future work.

For derivatives matching [7, 25, 30, 33, 36], TaSIL [30] is highly relevant as it expands the theory to higher order terms, with bounds in the context of imitation learning and some experiments. Our alternating loop is close to DAgger [32], but unlike the assumption of these frameworks, in our case, the TO solver is not a perfect global expert: new trajectories are not just meant to show how to recover from the policy mistakes, the goal is to find better trajectories over time.

Reinforcement learning (RL) is promising to improve the generality of our approach. On the tasks studied Sec. 4.4, *Sob+Diff* worked thanks to the *Argmin-Cost* term (Alg 1 Line 7). This filter was sufficient here, but for more challenging tasks, advanced techniques from offline-RL [28, 38], learning a critic, could be leveraged to better exploit the collected dataset, in particular to filter out low-quality local minima or to estimate the final cost, as in CACTO [2, 10]. Likewise, curriculum learning could be used to progressively focus on hard task instances.

The goal of this paper is to work along gradient-based TO, by providing initial guesses it reduces the solving time and avoids poor local minima; still, it inherits the limitations of these solvers. We chose ProxDDP [13] as it directly yields feedback gains, for a first implementation of our method, but we will benefit from developments in this research field. In particular, leveraging fully differentiable simulators [23] to enable contact-rich tasks. In particular, [27] proposed a way to adapt any TO solver to retrieve the feedback gains we need. While promising, research remains to be conducted as their approach is hard to reproduce.

6 Conclusion

We propose a Sobolev approach to train diffusion policies to warm start gradient-based trajectory optimization. It leverages the first-order information available in this context, for improved sample efficiency, higher policy precision, and longer prediction horizons. This approach empowers gradient-based TO with an expressive policy to generate initial guesses, reducing the solving time and leading to better solutions. Future work will focus on contact-rich scenarios, working with new TO solvers, and combining our learning framework with offline RL.

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 references Organic Robotics Program (PEPR O2R) and "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.
Note on LLM usage. We only used LLMs for basic grammar and spelling corrections.

References

1. Ajay, A., Du, Y., Gupta, A., Tenenbaum, J., Jaakkola, T., Agrawal, P.: Is conditional generative modeling all you need for decision-making? arXiv preprint arXiv:2211.15657 (2022)
2. Alboni, E., Grandesso, G., Papini, G.P.R., Carpentier, J., Del Prete, A.: Cacto-sl: Using sobolev learning to improve continuous actor-critic with trajectory optimization. In: 6th Annual Learning for Dynamics & Control Conference. pp. 1452–1463. PMLR (2024)
3. de Avila Belbute-Peres, F., Smith, K., Allen, K., Tenenbaum, J., Kolter, J.Z.: End-to-end differentiable physics for learning and control. Advances in neural information processing systems **31** (2018)
4. Carpentier, J., Mansard, N.: Analytical derivatives of rigid body dynamics algorithms. In: Robotics: Science and systems (RSS 2018) (2018)
5. Carpentier, J., Saurel, G., Buondonno, G., Mirabel, J., Lamiraux, F., Stasse, O., Mansard, N.: 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)
6. Chi, C., Feng, S., Du, Y., Xu, Z., Cousineau, E., Burchfiel, B., Song, S.: Diffusion policy: Visuomotor policy learning via action diffusion. arXiv preprint arXiv:2303.04137 (2023)
7. Czarnecki, W.M., Osindero, S., Jaderberg, M., Swirszcz, G., Pascanu, R.: Sobolev training for neural networks. Advances in neural information processing systems **30** (2017)
8. Dantec, E., Taix, M., Mansard, N.: First order approximation of model predictive control solutions for high frequency feedback. IEEE Robotics and Automation Letters **7**(2) (2022)
9. Giftthaler, M., Neunert, M., Stäuble, M., Frigerio, M., Semini, C., Buchli, J.: Automatic differentiation of rigid body dynamics for optimal control and estimation. Advanced Robotics **31**(22), 1225–1237 (2017)
10. Grandesso, G., Alboni, E., Papini, G.P.R., Wensing, P.M., Del Prete, A.: Cacto: Continuous actor-critic with trajectory optimization—towards global optimality. IEEE Robotics and Automation Letters **8**(6), 3318–3325 (2023)
11. Ho, J., Jain, A., Abbeel, P.: Denoising diffusion probabilistic models. Advances in neural information processing systems **33** (2020)
12. Jacobson, D.H., Mayne, D.Q.: Differential dynamic programming (1970)
13. Jallet, W., Bambade, A., Arlaud, E., El-Kazdadi, S., Mansard, N., Carpentier, J.: ProxDDP: Proximal constrained trajectory optimization. IEEE Transactions on Robotics (2025)
14. Jallet, W., Bambade, A., El Kazdadi, S., Carpentier, J., Nicolas, M.: aligator, https://github.com/Simple-Robotics/aligator
15. Janner, M., Du, Y., Tenenbaum, J., Levine, S.: Planning with diffusion for flexible behavior synthesis. In: International Conference on Machine Learning. PMLR (2022)
16. Le Lidec, Q., Jallet, W., Laptev, I., Schmid, C., Carpentier, J.: 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)
17. Le Lidec, Q., Kalevatykh, I., Laptev, I., Schmid, C., Carpentier, J.: Differentiable simulation for physical system identification. IEEE Robotics and Automation Letters **6**(2) (2021)

18. Lee, S.H., Kim, J., Park, F.C., Kim, M., Bobrow, J.E.: Newton-type algorithms for dynamics-based robot movement optimization. IEEE Transactions on robotics **21**(4), 657–667 (2005)
19. Levine, S., Finn, C., Darrell, T., Abbeel, P.: End-to-end training of deep visuomotor policies. The Journal of Machine Learning Research **17**(1) (2016)
20. Levine, S., Koltun, V.: Guided policy search. In: International conference on machine learning. PMLR (2013)
21. Li, A., Ding, Z., Dieng, A.B., Beeson, R.: Diffusolve: Diffusion-based solver for non-convex trajectory optimization. arXiv preprint arXiv:2403.05571 (2024)
22. Li, W., Todorov, E.: 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)
23. Lidec, Q.L., Montaut, L., de Mont-Marin, Y., Schramm, F., Carpentier, J.: End-to-end and highly-efficient differentiable simulation for robotics. arXiv preprint arXiv:2409.07107 (2024)
24. Lipman, Y., Chen, R.T., Ben-Hamu, H., Nickel, M., Le, M.: Flow matching for generative modeling. In: 11th International Conference on Learning Representations, ICLR 2023 (2023)
25. Mitchell, T.M., Thrun, S.B.: Explanation-based neural network learning for robot control. Advances in neural information processing systems **5** (1992)
26. Mordatch, I., Todorov, E., Popović, Z.: Discovery of complex behaviors through contact-invariant optimization. ACM Transactions on Graphics (ToG) **31**(4) (2012)
27. Mordatch, I., Todorov, E.: Combining the benefits of function approximation and trajectory optimization. In: Robotics: Science and Systems. vol. 4 (2014)
28. Nguimatsia Tiofack, F., Le Hellard, T., Schramm, F., Perrin-Gilbert, N., Carpentier, J.: Guided flow policy: Learning from high-value actions in offline reinforcement learning. arXiv preprint arXiv:2512.03973 (2025)
29. Nichol, A.Q., Dhariwal, P.: Improved denoising diffusion probabilistic models. In: International conference on machine learning. PMLR (2021)
30. Pfrommer, D., Zhang, T., Tu, S., Matni, N.: Tasil: Taylor series imitation learning. Advances in Neural Information Processing Systems **35**, 20162–20174 (2022)
31. Posa, M., Cantu, C., Tedrake, R.: A direct method for trajectory optimization of rigid bodies through contact. The International Journal of Robotics Research **33**(1), 69–81 (2014)
32. Ross, S., Gordon, G., Bagnell, D.: 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)
33. Simard, P.Y., LeCun, Y.A., Denker, J.S., Victorri, B.: Transformation invariance in pattern recognition—tangent distance and tangent propagation. In: Neural networks: tricks of the trade. Springer (2002)
34. Sohl-Dickstein, J., Weiss, E., Maheswaranathan, N., Ganguli, S.: Deep unsupervised learning using nonequilibrium thermodynamics. In: International conference on machine learning. PMLR (2015)
35. Song, Y., Sohl-Dickstein, J., Kingma, D.P., Kumar, A., Ermon, S., Poole, B.: Score-based generative modeling through stochastic differential equations. In: 9th International Conference on Learning Representations, ICLR 2021 (2021)
36. Srinivas, S., Fleuret, F.: Knowledge transfer with jacobian matching. In: International conference on machine learning. pp. 4723–4731. PMLR (2018)

37. Wang, R., Sharma, A., Parunandi, K.S., Goyal, R., Mohamed, M.N.G., Chakravorty, S.: The search for feedback in reinforcement learning. Journal of Dynamic Systems, Measurement, and Control **147**(6), 061002 (2025)
38. Wang, Z., Hunt, J.J., Zhou, M.: Diffusion policies as an expressive policy class for offline reinforcement learning. arXiv preprint arXiv:2208.06193 (2022)
39. Yang, L., Suh, H., Zhao, T., Graesdal, B.P., Kelestemur, T., Wang, J., Pang, T., Tedrake, R.: Physics-driven data generation for contact-rich manipulation via trajectory optimization. arXiv preprint arXiv:2502.20382 (2025)