DiffOG: Differentiable Policy Trajectory Optimization with Generalizability

Zhengtong Xu, Zichen Miao, Qiang Qiu, Zhe Zhang, Yu She*

***Abstract*—Imitation learning-based visuomotor policies excel at manipulation tasks but often produce suboptimal action trajectories compared to model-based methods. Directly mapping camera data to actions via neural networks can result in jerky motions and difficulties in meeting critical constraints, compromising safety and robustness in real-world deployment. For tasks that require high robustness or strict adherence to constraints, ensuring trajectory quality is crucial. However, the lack of interpretability in neural networks makes it challenging to generate constraint-compliant actions in a controlled manner. This paper introduces differentiable policy trajectory optimization with generalizability (DiffOG), a learning-based trajectory optimization framework designed to enhance visuomotor policies. By leveraging the proposed differentiable formulation of trajectory optimization with transformer, DiffOG seamlessly integrates policies with a generalizable optimization layer. Visuomotor policies enhanced by DiffOG generate smoother, constraint-compliant action trajectories in a more interpretable way. DiffOG exhibits strong generalization capabilities and high flexibility. We evaluated DiffOG across 11 simulated tasks and 2 real-world tasks. The results demonstrate that DiffOG significantly enhances the trajectory quality of visuomotor policies while having minimal impact on policy performance, outperforming trajectory processing baselines such as greedy constraint clipping and penalty-based trajectory optimization. Furthermore, DiffOG achieves superior performance compared to existing constrained visuomotor policy.**

I. Introduction

Imitation learning [15] has emerged as a popular paradigm for endowing robots with complex manipulation skills by leveraging human demonstrations. By formulating a supervised learning problem that maps sensor observations directly to actions via neural networks, imitation learning-based visuomotor policies have demonstrated effectiveness across a broad spectrum of tasks [5, 6, 8, 21, 35]. Yet, these learned policies often exhibit notable limitations when applied to real-world settings, where safety, robustness, and strict constraint satisfaction are paramount. In contrast to classical model-based trajectory optimization methods which explicitly account for motion constraints and can ensure smooth and reliable trajectories [19], imitation learning approaches may produce suboptimal or jerky actions. Such trajectories can reduce the robustness of the whole robotic systems and raise critical safety concerns.

Meanwhile, various challenges in imitation learning, such as the representation of multi-modal action distributions [5, 9] and training large, generalized policies on extensive datasets [3, 13, 14], have been the focus of ongoing research. Despite these advancements, ensuring the generation of high-quality and constraint-compliant trajectories for neural network-based policies remains a significant and open challenge. Unlike traditional model-based motion generation methods, learning-based approaches tend to be less controlled and more difficult to interpret. For instance, learning-based policies lack mechanisms to directly control trajectory smoothness or impose motion constraints through tunable parameters.

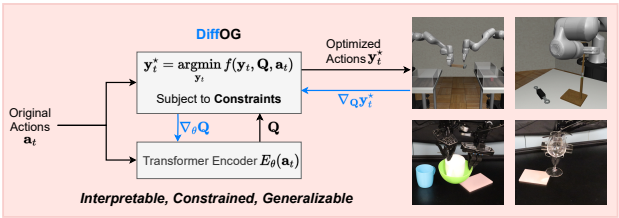


Fig. 1: We introduce differentiable policy trajectory optimization with generalizability (DiffOG). Visuomotor policies enhanced by DiffOG generate smoother, constraint-compliant action trajectories in a more interpretable way. DiffOG introduces a novel transformer-based differentiable trajectory optimization framework tailored for action refinement in imitation learning.

Instead, current approaches to improve trajectory quality are mainly based on post-hoc processing, such as applying trajectory optimization methods to refine the outputs of the learned policy. However, because constraints or smoothness objectives are not inherently considered during the policy training process, these adjustments can result in trajectories that deviate from those in the demonstration dataset. This misalignment can ultimately degrade the policy's performance. This often makes such post-hoc processing methods difficult to generalize across a wide range of tasks, particularly in scenarios involving high-dimensional action spaces, such as dual-arm manipulation, or long-horizon tasks with complex and highly varied trajectories.

In this paper, we introduce differentiable policy trajectory optimization with generalizability (DiffOG), a generalizable framework that seamlessly integrates transformer-based trajectory optimization with imitation learning to produce smooth, constraint-compliant action trajectories for robotic manipulation, as shown in Fig. 1.

The core contributions of DiffOG can be summarized as follows:

1. We propose a differentiable trajectory optimization framework designed to refine the actions generated by robot policies. The optimized trajectories are capable of accom-

*Address all correspondence to this author.
Zhengtong, Zichen, Qiang, Zhe, and Yu are with Purdue University, West Lafayette, USA xu1703, miaoz, qqiu, zhan5111, shey@purdue.edu

plishing the demonstrated tasks while satisfying hard constraints and exhibiting improved smoothness. Built upon a supervised learning paradigm, this framework offers high flexibility and generalizes well across diverse tasks and policy architectures. Compared to traditional approaches that enforce constraints through post-hoc processing, which often lead to actions deviating from the demonstration distribution, our method improves trajectory quality while maintaining alignment with the demonstrated behavior, thereby better preserving policy performance.

2. We integrate a transformer-based trajectory encoder into the proposed differentiable trajectory optimization framework, which substantially enhances its representational capacity. This design allows DiffOG to adapt trajectory optimization to the specific characteristics of diverse tasks, including long-horizon dual-arm manipulation, with strong generalization ability. Furthermore, through a series of deliberate design choices and theoretical analyses, our transformer-based differentiable optimization layer aligns with interpretable theoretical principles while ensuring feasibility and stable training. By combining model-based formulations with neural networks, this approach fully exploits differentiability to improve both the effectiveness and flexibility of DiffOG.

3. Through rigorous analysis, we establish the interpretability of DiffOG under this training paradigm and elucidate the underlying principles that guide its operation. Extensive validation in both simulation and real-world settings demonstrates the broad effectiveness of DiffOG, as well as the flexibility and generalization capabilities of its transformer-based trajectory optimization. Overall, DiffOG provides a practical and scalable example of integrating visual policies with a rigorous and interpretable optimization layer.

We evaluated DiffOG across 11 simulated tasks and 2 real-world tasks. The results demonstrate that DiffOG significantly enhances the trajectory quality of visuomotor policies with generalizability while having minimal impact on policy performance, outperforming trajectory processing baselines such as constraint clipping and penalty-based trajectory optimization. Furthermore, DiffOG achieves superior performance compared to existing constrained visuomotor policy [32].

II. Method

In this section, the details of the DiffOG will be introduced.

A. Action Space Assumption

We first analyze the action space to introduce the trajectory formulation of the actions generated by visuomotor policies.

The datasets used in imitation learning adhere to the format of

$$\mathcal{D} = \left\{ \left(o_0^i, a_0^i, o_1^i, a_1^i, \ldots, o_{T^i}^i, a_{T^i}^i \right) \right\}_{i=1}^{N_d},$$

where o represents the observations of the robot, such as images of the cameras with different views and robot states, and $a \in \mathbb{R}^{D_a}$ represents the demonstrated action aligned with the observation o. N_d is the total number of demonstrated trajectories, T is the total length of a trajectory, and D_a is the dimension of action a.

Action can be categorized into two types: discrete actions, such as controlling grasping and releasing using discrete values [12], and continuous actions, such as the motion of the robot. Discrete actions, by their nature, cannot form a continuous trajectory and therefore do not require trajectory optimization. To address this distinction, we introduce a selection matrix $S \in \mathbb{R}^{D_c \times D_a}$ that identifies the action dimensions to be considered for trajectory optimization. Specifically, S is used to select: 1) degrees of freedoms (DOFs) that are inherently continuous and capable of forming a trajectory, and 2) DOFs that require trajectory optimization in practical applications. The resulting filtered action variable is expressed as $c = Sa \in \mathbb{R}^{D_c}$. In this paper, we filter out the grasping action using S, as grasping typically does not require trajectory motion constraints and optimization.

Recently, considerable papers demonstrate that using the training objective of predicting a sequence of actions results in a more effective policy compared to predicting a single action [5, 9, 35]. Therefore, at time step t, the action sequence output of policies adhere to the format of

$$\mathbf{a}_t = \left[a_t^{\mathrm{T}}, a_{t+1}^{\mathrm{T}}, \ldots, a_{t+T_p-1}^{\mathrm{T}} \right]^{\mathrm{T}} \in \mathbb{R}^{T_p D_a}, \tag{1}$$

where T_p is the length of the predicted action sequence and $\mathbf{a}_t$ is a sequence of actions a. Apply the selection matrix $S \in \mathbb{R}^{D_c \times D_a}$ to $\mathbf{a}_t$, we have

$$\mathbf{c}_t = \mathbf{S}\mathbf{a}_t = \left[c_t^{\mathrm{T}}, c_{t+1}^{\mathrm{T}}, \ldots, c_{t+T_p-1}^{\mathrm{T}} \right]^{\mathrm{T}} \in \mathbb{R}^{T_p D_c}, \tag{2}$$

where $\mathbf{S} = \mathrm{blkdiag}\left(S, \ldots, S\right) \in \mathbb{R}^{T_p D_c \times T_p D_a}$ is a sequence of the selection matrix S and $\mathbf{c}_t$ is a sequence of filtered actions c.

In discrete trajectory form (2), the time derivative of $\mathbf{c}_t$ is expressed as

$$\begin{aligned}
\frac{d\mathbf{c}_t}{dt} &= \left[(\frac{dc_t}{dt})^{\mathrm{T}}, (\frac{dc_{t+1}}{dt})^{\mathrm{T}}, \ldots, (\frac{dc_{t+T_p-2}}{dt})^{\mathrm{T}} \right] \\
&= \frac{1}{\Delta t} \left[c_{t+1}^{\mathrm{T}} - c_t^{\mathrm{T}}, \ldots, c_{t+T_p-1}^{\mathrm{T}} - c_{t+T_p-2}^{\mathrm{T}} \right]^{\mathrm{T}} \\
&= \frac{1}{\Delta t} \begin{bmatrix} -1 & 1 & & & \\ & -1 & 1 & & \\ & & \ddots & \ddots & \\ & & & -1 & 1 \end{bmatrix} \mathbf{c}_t \\
&= \mathbf{A}_{\text{diff}} \mathbf{c}_t \in \mathbb{R}^{T_p D_c - D_c}.
\end{aligned} \tag{3}$$

Assumption 1. *The proposed method assumes that the discrete-time derivative $\frac{dc_t}{dt}$ in* (3) *represents decoupled explicit accelerations or velocities.*

Specifically, among commonly used action spaces for robot policies, Assumption 1 holds under the following conditions:

1. The action space consists of joint angles, where the derivative corresponds to joint velocities. This type of action space is commonly used in systems that collect data through leader-follower arms, such as ALOHA [35] and GELLO [28].

2. The action space consists of the robot end-effector linear position in xyz, where the derivative corresponds to linear velocity. This action space is used in the push-T task in [5] and the Meta-World benchmark [33].

Another widely adopted policy action space is $[v_x, v_y, v_z, \dot{r}_x, \dot{r}_y, \dot{r}_z]^{\mathrm{T}} \in \mathbb{R}^6$, where $\mathbf{v}_{xyz} = [v_x, v_y, v_z]^{\mathrm{T}}$ denotes the translational velocity, and $\dot{\mathbf{r}} = [\dot{r}_x, \dot{r}_y, \dot{r}_z]^{\mathrm{T}}$ is the time derivative of the axis–angle rotation vector [12, 26, 32, 36]. Taking the time derivative gives $[\dot{v}_x, \dot{v}_y, \dot{v}_z, \ddot{r}_x, \ddot{r}_y, \ddot{r}_z]^{\mathrm{T}}$. The first three elements are explicit linear accelerations, whereas the last three are axis-angle second derivatives, not the physical angular acceleration $\dot{\boldsymbol{\omega}}$. Therefore, this type of action space is not covered by Assumption 1. However, we will show in experiments that under this type of action representation, constraints on the action derivative can still effectively regulate the robot's physical motion.

B. Overview

As shown in Fig. 2, DiffOG leverages the differentiability of the optimization problem to directly learn a trajectory optimizer within a supervised learning framework. This trajectory optimizer is designed to balance objectives including constraint satisfaction, smoothness, and fidelity to the dataset, such that the resulting actions remain within the demonstrated distribution. Moreover, by incorporating transformer into the differentiable optimization layer, DiffOG significantly enhances both the generalization ability and representational power of trajectory optimization, making it applicable to a wide range of tasks, including long-horizon manipulation tasks with high-dimensional action spaces.

C. Differentiable Trajectory Optimization

In this section, we introduce the formulation of the differentiable trajectory optimization problem, which is the optimization process that converts an unoptimized action trajectory to the optimized action trajectory. We first define the optimized trajectory output by DiffOG as:

$$\mathbf{y}_t = \left[\hat{a}_t^{\mathrm{T}}, \hat{a}_{t+1}^{\mathrm{T}}, \ldots, \hat{a}_{t+T_p-1}^{\mathrm{T}}\right]^{\mathrm{T}} \in \mathbb{R}^{T_p D_a}.$$

The definition of $\mathbf{y}_t$ follows (1) and $\hat{a}$ here simply represents optimized actions. Similarly, from equations (2) and (3), the following equations of $\mathbf{y}_t$ can be derived

$$\hat{\mathbf{c}}_t = \mathbf{S}\mathbf{y}_t = \left[\hat{c}_t^{\mathrm{T}}, \hat{c}_{t+1}^{\mathrm{T}}, \ldots, \hat{c}_{t+T_p-1}^{\mathrm{T}}\right]^{\mathrm{T}} \in \mathbb{R}^{T_p D_c}, \quad (4)$$

$$\frac{d\hat{\mathbf{c}}_t}{dt} = \mathbf{A}_{\text{diff}}\mathbf{S}\mathbf{y}_t. \quad (5)$$

Based on all notations we defined, we can write the following optimization problem which take $\mathbf{a}_t$ as input and outputs the optimized $\mathbf{y}_t^\star$

$$\mathbf{y}_t^\star = \underset{\mathbf{y}_t}{\operatorname{argmin}} \frac{1}{2}\mathbf{y}_t^{\mathrm{T}}\mathbf{Q}\mathbf{y}_t + \mathbf{a}_t^{\mathrm{T}}\mathbf{y}_t + \frac{\alpha}{2}\left(\frac{d\hat{\mathbf{c}}_t}{dt}\right)^{\mathrm{T}}\frac{d\hat{\mathbf{c}}_t}{dt}, \quad (6)$$

$$\text{subject to} \quad d_{\min}\Delta t \le \hat{c}_{t+k+1} - \hat{c}_{t+k} \le d_{\max}\Delta t, \quad (7)$$
$$k = 0, 1, \ldots, T_p - 2.$$

In the optimization problem (6), $\mathbf{Q}$ is a weight matrix, α is a scalar, and $d_{\max}, d_{\min}$ are constraint bounds for the derivatives of the robot actions. Intuitively, α reflects the degree to which the trajectory smoothness is emphasized. When the sum of the derivatives of the robot actions $\hat{\mathbf{c}}_t$ in the generated trajectory $\mathbf{y}_t$ is large, it indicates that the trajectory is less smooth. For instance, excessive acceleration or velocity contributes to an increase in the cost function. Therefore, we call α as smoothing weight. Optimization problem (6) aims to solve for an action trajectory that satisfies the constraints on action derivatives while achieving greater smoothness, making it a trajectory optimization problem. $\mathbf{Q}$ also has a intuitive theoretical interpretation, which will be elaborated in detail in Section V-C. In Section V-C, we will delve deeper into the trajectory optimization objectives by incorporating learning-based perspectives.

The optimization problem (6) is a quadratic program, which is a class of optimization problems. In DiffOG, we instantiate this class by proposing a specific, learnable trajectory optimization formulation tailored for imitation learning.

D. Parameterize $\mathbf{Q}$ with Transformer

So far, both the differentiability discussed in Remark 2 and the derivation of the interpretable learning objectives in Remark 3 require $\mathbf{Q}$ to be symmetric positive definite. This section will introduce our method for constructing a symmetric positive definite $\mathbf{Q}$.

$\mathbf{Q}$ is the learnable linear transformation. Since actions are highly diverse after sampling from dataset $\mathcal{D}$, different action samples $\mathbf{a}_t$ may require distinct $\mathbf{Q}$ matrices to construct the trajectory optimization problem, ensuring an optimal balance of the three interpretable learning objectives described in Remark 3 under the presence of smoothing terms and constraints. When the task is long-horizon, complex, and involves a high number of robot DOFs, $\mathbf{Q}$ needs to be more generalizable.

Thanks to the well-posed properties, namely feasibility and differentiability, of our proposed trajectory optimization formulation, we are able to incorporate a transformer module to significantly enhance its capacity through a series of careful design choices. This enhancement enables the optimization process to generalize to a wider range of more complex tasks, while preserving interpretability (see Remark 3) and maintaining feasibility and training stability (see Remark 1). We use a transformer encoder E_θ to construct a symmetric positive definite $\mathbf{Q}$, as detailed in Algorithm 1 and Fig. 1. The encoder used in Algorithm 1 which computes the embedding $\mathbf{e}_t = E_\theta(\mathbf{a}_t)$ is shown in Fig. 3. In Algorithm 1, steps 3)–5) construct a lower triangular matrix with positive diagonal entries from the embedding output by the transformer. Additionally, the use of clamping prevents value explosion caused by the exponential operation. Based on Cholesky factorization, Step 6) results in a symmetric positive definite $\mathbf{Q}$. Step 7) further reinforces the symmetry of $\mathbf{Q}$ to mitigate the impact of minor numerical errors introduced during computation, ensuring the desired properties of $\mathbf{Q}$ are preserved.

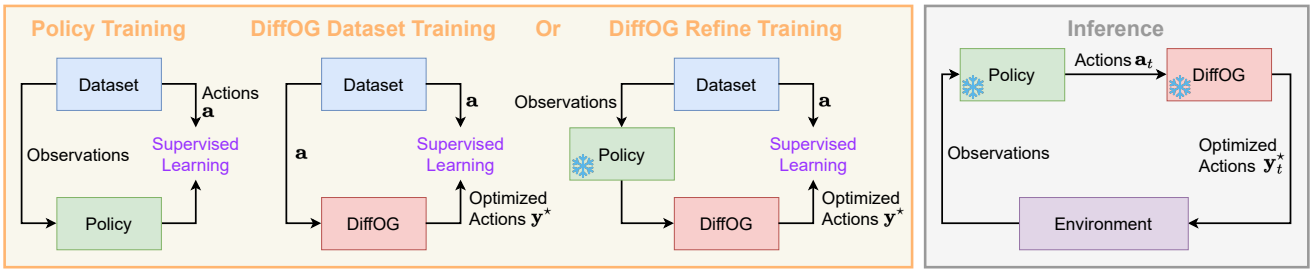


Fig. 2: High-level Overview of training and inference of DiffOG. A detailed illustration of DiffOG (red block) can be found in Fig. 1.

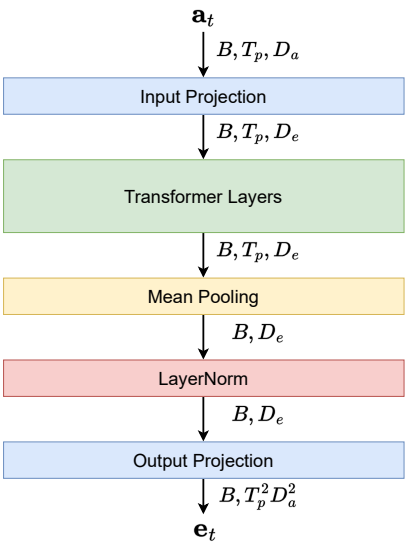


Fig. 3: Pipeline of the transformer encoder.

Algorithm 1 Constructing a symmetric positive definite matrix $\mathbf{Q}$ from $\mathbf{a}_t$

Input: $\mathbf{a}_t \in \mathbb{R}^{T_p D_a}$, small constant $\epsilon > 0$.
Output: $\mathbf{Q} \in \mathbb{R}^{T_p D_a \times T_p D_a}$

1) $\mathbf{e}_t \leftarrow \text{Encoder}(\mathbf{a}_t)$
2) $\mathbf{L} \leftarrow \mathbf{e}_t.\text{reshape}(T_p D_a, T_p D_a)$
3) **for each diagonal entry** $\mathbf{L}_{ii}$:

$$\mathbf{L}_{ii} \leftarrow \exp(\mathbf{L}_{ii}) + \epsilon$$

4) $\mathbf{L} \leftarrow \text{clamp}(\mathbf{L})$ \\ prevent numerical explosion
5) $\mathbf{L} \leftarrow \text{tril}(\mathbf{L})$ \\ set upper-triangular elements to zero
6) $\mathbf{Q} \leftarrow \mathbf{L}\mathbf{L}^{\mathrm{T}} + \epsilon \mathbf{I}$
7) $\mathbf{Q} \leftarrow \frac{1}{2}(\mathbf{Q} + \mathbf{Q}^{\mathrm{T}})$ \\ enhance symmetry

In practice, we clamp the values to the range [-10, 10]. For the small constant ϵ, we use 1×10^{-4}.

By Remark 2, we know that optimization problem (6) is differentiable, and $\nabla_{\mathbf{Q}} \mathbf{y}_t^\star$ can be computed. The operations in Algorithm 1 are also differentiable. Then $\nabla_\theta \mathbf{Q}$ can be computed. Therefore, based on the chain rule, DiffOG can be fully trained in an end-to-end manner. For quadratic programming solver, we utilize qpth [1], which enables batch-form quadratic programming computation along with efficient backpropagation.

III. Experiments

We validated DiffOG on 13 different tasks as shown in Fig. 4. Experiments involve two base policies, Diffusion Policy and 3D Diffusion Policy (DP3), along with a comprehensive comparison of trajectory processing baselines, including constraint clipping and penalty-based optimization. Additionally, we compare DiffOG with the constrained visuomotor policy [32]. We also conduct ablation studies on DiffOG.

For the hyperparameters of Diffusion Policy and DP3, we use the default parameters provided in their respective works [5, 34]. DiffOG, penalty-based optimization, and constraint clipping use same constraints and same base policy, either DP3 or Diffusion Policy, depending on the type of camera input used for the task. For details on the parameters of DiffOG and trajectory processing baselines, as well as information on the datasets, please refer to Appendix.

We divide the results of our experiments into four sections for presentation. The detailed results can be found in Appendix (Section VII).

Benchmark with Baselines Across Tasks: We demonstrate how DiffOG optimizes the trajectories generated by Diffusion Policy and DP3 across 13 tasks. Additionally, we compare its performance with baseline methods, including constraint clipping and penalty-based trajectory optimization.

Ablation Study on Adjustability of Trajectory: We show how adjusting the smoothing weight and constraint bounds allows us to control the properties of the optimized actions produced by DiffOG. This section further demonstrates the interpretability of DiffOG.

Ablation Study on Static Q, Matrix Learning, and Transformer: Through experiments, we demonstrate the significance of the transformer-based design in DiffOG by comparing it with static $\mathbf{Q}$ and matrix-learning-based $\mathbf{Q}$ components. The results emphasize that the transformer architecture plays a crucial role in achieving superior performance. Specifically, it enables DiffOG to fully leverage differentiability, effectively balance trajectory optimization learning objectives, and maintain high fidelity to demonstration data trajectories.

Comparison with a Constrained Visuomotor Policy: We present comparative experiments between DiffOG and an constrained visuomotor policy Leto [32], showing that DiffOG achieve superior performance.

References

[1] Brandon Amos and J Zico Kolter. OptNet: Differentiable optimization as a layer in neural networks. In *Proc. 34th Int. Conf. Mach. Learn.*, pages 136–145, 2017.

[2] Brandon Amos, Ivan Jimenez, Jacob Sacks, Byron Boots, and J Zico Kolter. Differentiable MPC for end-to-end planning and control. *Advances in Neural Information Processing Systems*, 2018.

[3] Kevin Black, Noah Brown, Danny Driess, Adnan Esmail, Michael Equi, Chelsea Finn, Niccolo Fusai, Lachy Groom, Karol Hausman, Brian Ichter, et al. π_0: A vision-language-action flow model for general robot control. *arXiv preprint arXiv:2410.24164*, 2024.

[4] Ching-An Cheng, Mustafa Mukadam, Jan Issac, Stan Birchfield, Dieter Fox, Byron Boots, and Nathan Ratliff. Rmp flow: A computational graph for automatic motion policy generation. In *Algorithmic Foundations of Robotics XIII: Proceedings of the 13th Workshop on the Algorithmic Foundations of Robotics 13*, pages 441–457. Springer, 2020.

[5] Cheng Chi, Siyuan Feng, Yilun Du, Zhenjia Xu, Eric Cousineau, Benjamin Burchfiel, and Shuran Song. Diffusion policy: Visuomotor policy learning via action diffusion. In *Proceedings of Robotics: Science and Systems*, 2023.

[6] Cheng Chi, Zhenjia Xu, Chuer Pan, Eric Cousineau, Benjamin Burchfiel, Siyuan Feng, Russ Tedrake, and Shuran Song. Universal manipulation interface: In-the-wild robot teaching without in-the-wild robots. In *Proceedings of Robotics: Science and Systems*, 2024.

[7] Pete Florence, Corey Lynch, Andy Zeng, Oscar A Ramirez, Ayzaan Wahid, Laura Downs, Adrian Wong, Johnny Lee, Igor Mordatch, and Jonathan Tompson. Implicit behavioral cloning. In *Proceedings of Conference on Robot Learning*, 2022.

[8] Zipeng Fu, Tony Z. Zhao, and Chelsea Finn. Mobile aloha: Learning bimanual mobile manipulation with low-cost whole-body teleoperation. In *Proceedings of Conference on Robot Learning*, 2024.

[9] Seungjae Lee, Yibin Wang, Haritheja Etukuru, H Jin Kim, Nur Muhammad Mahi Shafiullah, and Lerrel Pinto. Behavior generation with latent actions. In *Proceedings of International Conference on Machine Learning*, 2024.

[10] Anqi Li, Ching-An Cheng, M Asif Rana, Man Xie, Karl Van Wyk, Nathan Ratliff, and Byron Boots. RMP2: A structured composable policy class for robot learning. In *Proceedings of Robotics: Science and Systems*, 2021.

[11] Songming Liu, Lingxuan Wu, Bangguo Li, Hengkai Tan, Huayu Chen, Zhengyi Wang, Ke Xu, Hang Su, and Jun Zhu. Rdt-1b: a diffusion foundation model for bimanual manipulation. *arXiv preprint arXiv:2410.07864*, 2024.

[12] Ajay Mandlekar, Danfei Xu, Josiah Wong, Soroush Nasiriany, Chen Wang, Rohun Kulkarni, Li Fei-Fei, Silvio Savarese, Yuke Zhu, and Roberto Martín-Martín. What matters in learning from offline human demonstrations for robot manipulation. In *Proc. Conf. Robot Learn.*, pages 1678–1690, 2022.

[13] Octo Model Team, Dibya Ghosh, Homer Walke, Karl Pertsch, Kevin Black, Oier Mees, Sudeep Dasari, Joey Hejna, Charles Xu, Jianlan Luo, Tobias Kreiman, You Liang Tan, Lawrence Yunliang Chen, Pannag Sanketi, Quan Vuong, Ted Xiao, Dorsa Sadigh, Chelsea Finn, and Sergey Levine. Octo: An open-source generalist robot policy. In *Proceedings of Robotics: Science and Systems*, 2024.

[14] Abby O'Neill, Abdul Rehman, Abhinav Gupta, Abhiram Maddukuri, Abhishek Gupta, Abhishek Padalkar, Abraham Lee, Acorn Pooley, Agrim Gupta, Ajay Mandlekar, et al. Open X-Embodiment: Robotic learning datasets and RT-X models. *arXiv preprint arXiv:2310.08864*, 2023.

[15] Dean A Pomerleau. Alvinn: An autonomous land vehicle in a neural network. *Advances in neural information processing systems*, 1988.

[16] M Asif Rana, Anqi Li, Dieter Fox, Sonia Chernova, Byron Boots, and Nathan Ratliff. Towards coordinated robot motions: End-to-end learning of motion policies on transform trees. In *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2021.

[17] Nathan D Ratliff, Jan Issac, Daniel Kappler, Stan Birchfield, and Dieter Fox. Riemannian motion policies. *arXiv preprint arXiv:1801.02854*, 2018.

[18] Matthew Retchin, Brandon Amos, Steven Brunton, and Shuran Song. Koopman constrained policy optimization: A koopman operator theoretic method for differentiable optimal control in robotics. In *ICML 2023 Workshop on Differentiable Almost Everything: Differentiable Relaxations, Algorithms, Operators, and Simulators*, 2023. URL https://openreview.net/forum?id=3W7vPqWCeM.

[19] John Schulman, Yan Duan, Jonathan Ho, Alex Lee, Ibrahim Awwal, Henry Bradlow, Jia Pan, Sachin Patil, Ken Goldberg, and Pieter Abbeel. Motion planning with sequential convex optimization and convex collision checking. *The International Journal of Robotics Research*, 2014.

[20] Mingyo Seo, H Andy Park, Shenli Yuan, Yuke Zhu, and Luis Sentis. Legato: Cross-embodiment imitation using a grasping tool. *arXiv preprint arXiv:2411.03682*, 2024.

[21] Nur Muhammad Mahi Shafiullah, Anant Rai, Haritheja Etukuru, Yiqian Liu, Ishan Misra, Soumith Chintala, and Lerrel Pinto. On bringing robots home. *arXiv preprint arXiv:2311.16098*, 2023.

[22] Jiaming Song, Chenlin Meng, and Stefano Ermon. Denoising diffusion implicit models. *Proceedings of International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=St1giarCHLP.

[23] Weikang Wan, Ziyu Wang, Yufei Wang, Zackory Erick-

son, and David Held. DiffTORI: Differentiable trajectory optimization for deep reinforcement and imitation learning. *Advances in Neural Information Processing Systems*, 2024. URL https://openreview.net/forum?id=Mwj57TcHWX.

[24] Chen Wang, Haochen Shi, Weizhuo Wang, Ruohan Zhang, Li Fei-Fei, and C. Karen Liu. Dexcap: Scalable and portable mocap data collection system for dexterous manipulation. In *Proceedings of Robotics: Science and Systems*, 2024.

[25] Dian Wang, Stephen Hart, David Surovik, Tarik Kelestemur, Haojie Huang, Haibo Zhao, Mark Yeatman, Jiuguang Wang, Robin Walters, and Robert Platt. Equivariant diffusion policy. In *Proceedings of Conference on Robot Learning*, 2024.

[26] Lirui Wang, Jialiang Zhao, Yilun Du, Edward H Adelson, and Russ Tedrake. Poco: Policy composition from and for heterogeneous robot learning. In *Proceedings of Robotics: Science and Systems*, 2024.

[27] Chuan Wen*, Xingyu Lin*, John So*, Kai Chen, Qi Dou, Yang Gao, and Pieter Abbeel. Any-point trajectory modeling for policy learning. In *Proceedings of Robotics: Science and Systems*, 2024.

[28] Philipp Wu, Yide Shentu, Zhongke Yi, Xingyu Lin, and Pieter Abbeel. Gello: A general, low-cost, and intuitive teleoperation framework for robot manipulators. In *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2024.

[29] Wei Xiao, Tsun-Hsuan Wang, Ramin Hasani, Makram Chahine, Alexander Amini, Xiao Li, and Daniela Rus. Barriernet: Differentiable control barrier functions for learning of safe robot control. *IEEE Transactions on Robotics*, 2023. doi: 10.1109/TRO.2023.3249564.

[30] Mengda Xu, Zhenjia Xu, Yinghao Xu, Cheng Chi, Gordon Wetzstein, Manuela Veloso, and Shuran Song. Flow as the cross-domain manipulation interface. In *Proceedings of Conference on Robot Learning*, 2024.

[31] Zhengtong Xu and Yu She. LeTac-MPC: Learning model predictive control for tactile-reactive grasping. *IEEE Transactions on Robotics*, 2024. doi: 10.1109/TRO.2024.3463470.

[32] Zhengtong Xu and Yu She. Leto: Learning constrained visuomotor policy with differentiable trajectory optimization. *IEEE Transactions on Automation Science and Engineering*, 2024. doi: 10.1109/TASE.2024.3486542.

[33] Tianhe Yu, Deirdre Quillen, Zhanpeng He, Ryan Julian, Karol Hausman, Chelsea Finn, and Sergey Levine. Meta-world: A benchmark and evaluation for multi-task and meta reinforcement learning. In *Proceedings of Conference on robot learning*, 2020.

[34] Yanjie Ze, Gu Zhang, Kangning Zhang, Chenyuan Hu, Muhan Wang, and Huazhe Xu. 3D Diffusion Policy: generalizable visuomotor policy learning via simple 3D representations. In *Proceedings of Robotics: Science and Systems*, 2024.

[35] Tony Z. Zhao, Vikash Kumar, Sergey Levine, and Chelsea Finn. Learning Fine-Grained Bimanual Manipulation with Low-Cost Hardware. In *Proceedings of Robotics: Science and Systems*, 2023. doi: 10.15607/RSS.2023.XIX.016.

[36] Yifeng Zhu, Abhishek Joshi, Peter Stone, and Yuke Zhu. Viola: Imitation learning for vision-based manipulation with object proposal priors. In *Proceedings of Conference on Robot Learning*, 2023.

APPENDIX

IV. RELATED WORK

A. Supervised Policy Learning

Supervised policy learning entails training a policy using supervised learning techniques on a dataset of pre-collected demonstrations [15]. Recent advancements in machine learning algorithms and data collection frameworks have substantially advanced the capabilities and effectiveness of supervised policy learning in manipulation applications. For example, various data collection frameworks and devices have been proposed in [6, 8, 20, 21, 24, 35], enabling efficient data collection and the training of effective manipulation policies. To better capture the multi-modal action distributions present in demonstration data, prior works have explored various techniques, such as using diffusion models [5], VQ-VAE [9], and energy-based models [7]. Furthermore, various effective action representations for supervised policy learning have been proposed, such as flow by point tracking [27, 30] and equivariant action representation [25]. Moreover, many of works have contributed to training more general and scalable policy policies on large-scale datasets [3, 11, 13, 14].

However, the aforementioned methods largely overlook the importance of incorporating constraints into the actions generated by policy outputs. Despite these advancements, ensuring the generation of high-quality and constraint-compliant trajectories for neural network-based policies remains a significant and open challenge. DiffOG addresses this issue by applying transformer-based differentiable trajectory optimization, refining its generated actions to improve smoothness, adherence to hard constraints, and fidelity to the demonstration.

B. Policies with Differentiable Optimization

The integration of differentiable optimization into policy learning has garnered significant attention due to its potential to combine model-based formulations with the representational power of neural networks. Previous studies have investigated the integration of model-based structures with policies through differentiable optimization, including differentiable model predictive control (MPC) [2] and the Koopman operator framework [18]. However, these methods are predominantly designed for low-dimensional observations, which limits their flexibility and applicability to the general formulation of visuomotor policies. The work in [31] presents a tactile-reactive grasping controller that integrates an image encoder with differentiable MPC. Although effective for

specific tactile-based grasping tasks, this approach lacks the formulation required for broader policy learning frameworks. In the domain of obstacle avoidance, an end-to-end learning framework incorporating differentiable optimization is proposed in [29]. However, this approach is specifically designed for navigation tasks and lacks applicability to robotic manipulation scenarios and visuomotor policies.

Riemannian motion policies [17] and their extensions [4, 10, 16] advance the generation of motions in high-dimensional spaces with complex, nonlinear dynamics into end-to-end robot learning by leveraging automatic differentiation and Riemannian geometry. However, in the context of policy learning, Riemannian motion policies are primarily designed for tasks involving low-dimensional observations, such as those demonstrated through kinesthetic teaching [16]. In contrast, DiffOG is tailored for a broader range of visuomotor policies, refining policies that utilize camera observations as input. DiffOG focuses on generating task-completing trajectories while enhancing smoothness and ensuring compliance with hard constraints.

DiffTORI [23] integrates differentiable trajectory optimization into the policy representation to generate actions, with a focus on capturing multi-modal action distributions in the context of imitation learning. However, DiffTORI lacks explicit and rigorous formulations for constraints and smoothness, limiting its ability to optimize trajectory smoothness and enforce compliance with hard constraints. In contrast, DiffOG incorporates a rigorously designed optimization layer that not only enables actions to satisfy hard constraints, but also enhances the interpretability of the policy.

Leto [32], by employing differentiable optimization, enables end-to-end training and inference that embedded trajectory optimization, allowing constraints to be incorporated as part of the training objectives. However, Leto was designed as a discriminative model, which limits its ability to represent multi-modal action distributions [7]. Its use of an optimization layer as the action head makes it difficult to integrate with commonly-used diffusion-based action heads [5, 11, 13, 34], significantly reducing its flexibility and applicability. In comparison, DiffOG offers greater flexibility and enhanced representational capacity, enabling it to achieve superior policy performance and adapt more effectively to a wider range of tasks.

V. More Details of Method

A. Overview

In imitation learning, the policy learns the demonstrated task skills through supervised learning on the demonstration dataset, as shown in Fig. 2. The successful execution of a task depends on the policy generating action data that matches the distribution of the demonstration dataset. The objective of supervised learning in this context is to enable the policy to generate action that follows the same distribution as the demonstration dataset. However, traditional policy learning neglects trajectory constraints and smoothness. As a result, post-processing the policy's output to enforce these properties tends to shift the actions away from the demonstrated distribution, thereby compromising policy performance.

To overcome this challenge, DiffOG leverages the differentiability of the optimization problem to directly learn a trajectory optimizer within a supervised learning framework. This trajectory optimizer is designed to balance objectives including constraint satisfaction, smoothness, and fidelity to the dataset, such that the resulting actions remain within the demonstrated distribution. Moreover, by incorporating transformer into the differentiable optimization layer, DiffOG significantly enhances both the generalization ability and representational power of trajectory optimization, making it applicable to a wide range of tasks, including long-horizon manipulation tasks with high-dimensional action spaces.

The primary objective of DiffOG is to take an unoptimized action trajectory $\mathbf{a}_t$ as input and output an optimized trajectory that can accomplishes the demonstrated task, ensuring constraint satisfaction and improving smoothness. During training, DiffOG supports two modes:

1. DiffOG dataset training: Supervised learning is performed directly on the demonstration dataset.

2. DiffOG refine training: Supervised learning refines the actions generated by a pre-trained policy.

In the inference phase, DiffOG is applied atop the base policy to optimize the actions. The overall workflow is illustrated in Figs. 1 and 2, showing how an unoptimized trajectory is transformed into an optimized one through our transformer-based differentiable trajectory optimization framework. In the following sections, we elaborate on the process of transforming an unoptimized action sequence into an optimized one, explain why it is trainable, describe how this process is designed to be generalizable, and detail the training procedure.

B. Feasibility and Differentiability

We have introduced the overall formulation of trajectory optimization in DiffOG. To further enable learning within this optimization layer, it is essential to emphasize a key property: differentiability. A differentiable trajectory optimization module can be seamlessly integrated with neural networks for end-to-end training, enabling the optimization process to acquire stronger representational capacity through data-driven learning. In what follows, we delve into the details of differentiability, beginning with the following proposition:

Proposition 1. *If $d_{min} < d_{max}$ and $\mathbf{Q}$ is symmetric positive definite, optimization problem* (6) *is always feasible and strictly convex.*

Proof: The cost function of optimization problem (6) can be expressed as

$$\frac{1}{2}\mathbf{y}_t^{\mathrm{T}}\left(\mathbf{Q}+\alpha\mathbf{S}^{\mathrm{T}}\mathbf{A}_{\text{diff}}^{\mathrm{T}}\mathbf{A}_{\text{diff}}\mathbf{S}\right)\mathbf{y}_t+\mathbf{a}_t^{\mathrm{T}}\mathbf{y}_t.$$

Because rank(S) = D_c, rank($\mathbf{S}$) = T_pD_c. Because c is taken from a subset of a, it follows that $D_a \geq D_c$. Therefore, $\mathbf{S}$ has full rank. Since $\mathbf{Q}$ is symmetric positive definite and the

smoothing weight $\alpha \geq 0$, the matrix $\mathbf{Q} + \alpha \mathbf{S}^{\mathrm{T}} \mathbf{A}_{\text{diff}}^{\mathrm{T}} \mathbf{A}_{\text{diff}} \mathbf{S}$ is also symmetric positive definite. The condition $d_{\min} < d_{\max}$ ensures the feasibility of constraints, guaranteeing constraints are not contradictory.

Therefore, optimization problem (6) is always feasible and strictly convex. ■

Remark 1 (Trajectory Optimization)**.** *Proposition 1 guarantees that optimization problem* (6) *is feasible and strictly convex, ensuring the existence of a unique optimal solution* $\mathbf{y}_t^\star$*. However, optimization problem* (6) *constitutes a meaningful trajectory optimization only under Assumption 1. This is because, under Assumption 1, the constraints in* (7) *correspond to physically meaningful velocity or acceleration bounds, and the smoothing term* $(\frac{d\hat{\mathbf{c}}_t}{dt})^{\mathrm{T}} \frac{d\hat{\mathbf{c}}_t}{dt}$ *represents a physically interpretable quantity which reflects the trajectory's smoothness. For instance, a large* $(\frac{d\hat{\mathbf{c}}_t}{dt})^{\mathrm{T}} \frac{d\hat{\mathbf{c}}_t}{dt}$ *indicates a lack of smoothness in the trajectory. Consequently, optimization problem* (6) *can enforce trajectory smoothing by minimizing the associated cost. Moreover, Assumption 1 ensures that the condition* $d_{min} < d_{max}$ *carries explicit physical meaning. For example, if the action space is defined in terms of joint angles, this condition ensures that the minimum and maximum joint velocity constraints do not conflict. Only when both Assumption 1 and the condition* $d_{min} < d_{max}$ *hold does the constraining in optimization problem* (6) *remain valid and physically meaningful.*

In terms of differentiability, by Proposition 1 and the work in [1], we can get the following remark.

Remark 2 (Differentiability)**.** *Since* $\mathbf{Q} \succ 0$*, the optimization problem* (6) *is strictly convex, ensuring a unique optimal solution* $\mathbf{y}_t^\star$ *that is continuous and subdifferentiable with respect to all variables everywhere, and differentiable at all but a measure-zero set of points. In this case, for example, the gradient* $\nabla_{\mathbf{Q}} \mathbf{y}_t^\star$ *can be computed explicitly via the solution of the KKT conditions, leveraging the fact that the optimality conditions are differentiable [1].*

C. The Learning Objectives of DiffOG

According to Remark 2, we know that optimization problem (6) is theoretically differentiable with respect to all variables. However, in practice, we can design the optimization process such that certain variables remain differentiable while others are treated as constants. This section will introduce the rationale behind our design choice of constants and differentiable variables and provide details on the resulting learning objectives.

First, let us assume that $\mathbf{Q}$ is a symmetric positive definite matrix. According to Proposition 1, this condition with proper constraints values ensures that the optimization problem (6) is both feasible and differentiable. By Cholesky factorization, there exists a unique upper-triangular matrix $\mathbf{R}$ with positive diagonal entries such that $\mathbf{Q} = \mathbf{R}^{\mathrm{T}}\mathbf{R}$. Because $\mathbf{Q}$ is symmetric positive definite, the factor $\mathbf{R}$ is invertible (i.e., full rank).

Next, suppose we define the vector $\mathbf{g}_t$ by the relation

$$\mathbf{g}_t = -(\mathbf{R}^{\mathrm{T}})^{-1}\mathbf{a}_t.$$

Since $\mathbf{R}$ is full rank and upper triangular, its transpose $\mathbf{R}^{\mathrm{T}}$ is also invertible. Hence $(\mathbf{R}^{\mathrm{T}})^{-1}$ exists, and the vector $\mathbf{g}_t$ is uniquely determined by the above equation. In addition, $\mathbf{a}_t = -\mathbf{R}^{\mathrm{T}}\mathbf{g}_t$. Then, part of the cost function of optimization problem (6) can be rewritten as

$$\begin{aligned}
&\frac{1}{2}\mathbf{y}_t^{\mathrm{T}}\mathbf{Q}\mathbf{y}_t + \mathbf{a}_t^{\mathrm{T}}\mathbf{y}_t \\
&\qquad = \frac{1}{2}\mathbf{y}_t^{\mathrm{T}}\mathbf{R}^{\mathrm{T}}\mathbf{R}\mathbf{y}_t + \mathbf{a}_t^{\mathrm{T}}\mathbf{y}_t \\
&\qquad = \frac{1}{2}\mathbf{y}_t^{\mathrm{T}}\mathbf{R}^{\mathrm{T}}\mathbf{R}\mathbf{y}_t - \mathbf{g}_t^{\mathrm{T}}\mathbf{R}\mathbf{y}_t \\
&\qquad = \frac{1}{2}\Big(\mathbf{y}_t^{\mathrm{T}}\mathbf{R}^{\mathrm{T}}\mathbf{R}\mathbf{y}_t - 2\mathbf{g}_t^{\mathrm{T}}\mathbf{R}\mathbf{y}_t + \mathbf{g}_t^{\mathrm{T}}\mathbf{g}_t - \mathbf{g}_t^{\mathrm{T}}\mathbf{g}_t\Big) \\
&\qquad = \frac{1}{2}\Big\|\mathbf{R}\mathbf{y}_t - \mathbf{g}_t\Big\|^2 + \underbrace{\Big(-\tfrac{1}{2}\|\mathbf{g}_t\|^2\Big)}_{\text{independent of } \mathbf{y}_t}.
\end{aligned}$$

Since the term $\big(-\frac{1}{2}\|\mathbf{g}_t\|^2\big)$ is independent of $\mathbf{y}_t$, it does not affect the optimization process and can be omitted from the cost function of optimization problem (6). Thus, optimization problem (6) can be rewritten as:

$$\begin{aligned}
\mathbf{y}_t^\star =& \operatorname*{argmin}_{\mathbf{y}_t} \frac{1}{2}\Big\|\mathbf{R}\mathbf{y}_t - \mathbf{g}_t\Big\|^2 + \frac{\alpha}{2}(\frac{d\hat{\mathbf{c}}_t}{dt})^{\mathrm{T}}\frac{d\hat{\mathbf{c}}_t}{dt}, \quad (8)\\
\text{subject to} \quad & d_{\min}\Delta t \leq \hat{c}_{t+k+1} - \hat{c}_{t+k} \leq d_{\max}\Delta t, \\
& k = 0, 1, \ldots, T_p - 2.
\end{aligned}$$

We can observe that if we dropout the constraints and smoothing term, the solution of optimization problem (8) will be

$$\mathbf{y}_t = -\mathbf{R}^{-1}(\mathbf{R}^{\mathrm{T}})^{-1}\mathbf{a}_t = -(\mathbf{R}^{\mathrm{T}}\mathbf{R})^{-1}\mathbf{a}_t = -\mathbf{Q}^{-1}\mathbf{a}_t, \quad (9)$$

which represent a linear transformation of the original action $\mathbf{a}_t$. In the presence of constraints and smoothing terms, the optimization problem (8) is no longer solely about a linear transformation; instead, it simultaneously accounts for three aspects: transformation, constraints, and smoothness. Next, we demonstrate how differentiability and learning can be leveraged to integrate these three aspects into three interpretable learning objectives.

As previously mentioned, optimization problem (8) is derived by rewriting optimization problem (6) using Cholesky factorization $\mathbf{Q} = \mathbf{R}^{\mathrm{T}}\mathbf{R}$. Therefore, in optimization problem (8), we can make $\mathbf{Q}$ learnable/differentiable and train differentiable trajectory optimization (8) with the loss function

$$\mathcal{L} = \mathbb{E}_{\mathbf{a}_t \in \mathcal{D}}\|\mathbf{y}_t^\star - \mathbf{a}_t\|^2. \quad (10)$$

In this case, 1) the forward pass of DiffOG involves solving the optimization problem (8) to generate $\mathbf{y}_t^\star$, and 2) the backpropagation process updates $\mathbf{Q}$ using the loss function (10). The interpretable learning objectives can then be summarized in the following remark.

Remark 3 (Interpretability)**.** *Training differentiable trajectory optimization* (8) *with loss function* (10) *results in three interpretable learning objectives:*

1) The first objective is to make the output $\mathbf{y}_t^\star$ as close as possible to $\mathbf{a}_t$. From the perspective of imitation learning, this makes that $\mathbf{y}_t^\star$ remain fidelity to demonstration dataset, thereby enabling the actions generated by DiffOG to successfully perform the demonstrated task.

2) The second objective is to make the output satisfy the constraints in optimization problem (8)*. According to Proposition 1, optimization problem* (8) *is guaranteed to have a unique optimal solution, and this solution will always satisfy the constraints.*

3) Due to the presence of the smooth term, $\mathbf{y}_t^\star$ tends to be smoother. Specifically, the sum of its derivatives across time steps tends to be smaller during optimization.

Thus, the training of DiffOG balances the three interpretable learning objectives outlined in Remark 3. In essence, this process can be summarized as learning $\mathbf{Q}$ to formulate a quadratic programming problem that produces a solution balancing smoothness, enforcing constraint satisfaction, and ensuring that $\mathbf{y}_t^\star$ closely aligns with $\mathbf{a}_t$.

Moreover, the reason for setting only $\mathbf{Q}$ as learnable is that the remaining variables, such as the upper and lower bounds of the constraints d_{max} and d_{min}, and the smoothing weight α, have more direct model-based meanings and can be manually specified based on requirements without the need for learning. Different values for the smoothing weight α, and the bounds of the constraints d_{max} and d_{min} can directly and controllably influence the properties of the output $\mathbf{y}_t^\star$.

D. Inference

The training of DiffOG is on discrete trajectory chunks. Therefore, to fully constrain the generated trajectory during inference, the forward pass of optimization problem (6) need to add a new constraint.

$$\begin{aligned}\mathbf{y}_t^\star =& \underset{\mathbf{y}_t}{\operatorname{argmin}} \frac{1}{2}\mathbf{y}_t^{\mathrm{T}}\mathbf{Q}\mathbf{y}_t + \mathbf{a}_t^{\mathrm{T}}\mathbf{y}_t + \frac{\alpha}{2}(\frac{d\hat{\mathbf{c}}_t}{dt})^{\mathrm{T}}\frac{d\hat{\mathbf{c}}_t}{dt},\\ \text{subject to}\ & d_{\text{min}}\Delta t \le \hat{c}_t - \hat{c}_{t-1} \le d_{\text{max}}\Delta t,\\ & d_{\text{min}}\Delta t \le \hat{c}_{t+k+1} - \hat{c}_{t+k} \le d_{\text{max}}\Delta t,\\ & k = 0, 1. \ldots, T_p - 2.\end{aligned}$$

$\hat{c}_{t-1}$ is the action at the final time step from the previous iteration.

VI. Baselines

In this section, we introduce two trajectory processing baseline methods: constraint clipping and penalty-based trajectory optimization. These baselines apply post-hoc processing to action trajectories generated by policies. Since visuomotor policies lack mechanisms for enforcing constraints and adjusting actions, model-based post-hoc trajectory processing methods are widely adopted in real-world systems.

The goal of both constraint clipping and penalty-based trajectory optimization is to regulate the action outputs of base policies, ensuring adherence to predefined constraints. These methods are formulated within model-based optimization frameworks. Constraint clipping can be viewed as a greedy optimization approach, whereas penalty-based optimization performs global optimization across all time steps by leveraging penalties.

According to equation (2), for the sake of simplicity, we only include the action dimensions to be considered for trajectory optimization in the formulas presented in this section. For action dimensions that do not require trajectory optimization, such as grasping actions, the corresponding action dimensions from the base policy output can be retained. These can then be concatenated with the optimized trajectory actions after the trajectory optimization process. In this case, the input of the optimization problem becomes

$$\mathbf{c}_t = \mathbf{S}\mathbf{a}_t = \left[c_t^{\mathrm{T}}, c_{t+1}^{\mathrm{T}}, \ldots, c_{t+T_p-1}^{\mathrm{T}}\right]^{\mathrm{T}} \in \mathbb{R}^{T_p D_c},$$

where $\mathbf{S}$ is for selecting the the action dimensions to be considered for trajectory optimization. Similarly, we define an output variable

$$\hat{\mathbf{c}}_t = \left[\hat{c}_t^{\mathrm{T}}, \hat{c}_{t+1}^{\mathrm{T}}, \ldots, \hat{c}_{t+T_p-1}^{\mathrm{T}}\right]^{\mathrm{T}} = \mathbf{S}\mathbf{y}_t \in \mathbb{R}^{T_p D_c},$$

and we denote $\hat{\mathbf{c}}_t^\star$ as the optimized output variable. For continuous trajectory optimization, like what we do in Section V-D, a previous action $\hat{c}_{t-1}$ from last prediction is also needed as input.

Based on equation (7), the optimization is subject to constraints defined by two vectors, $d_{\text{min}}\Delta t$ and $d_{\text{max}}\Delta t$, which specify the minimum and maximum allowable bounds for the change in each dimension of the action vector.

A. Greedy Constraint Clipping

The goal of constraint clipping is to compute an optimized sequence of actions $\hat{\mathbf{c}}_t$ that satisfies the given constraints by greedy strategy.

For the initial time step, the optimized action is initialized as $\hat{c}_{t-1}^\star = \hat{c}_{t-1}$. Then, for each subsequent time step $t+k, k = 0 \ldots T_p - 1$, the difference between the input action c_{t+k} and the optimized action from the previous time step $\hat{c}_{t+k-1}^\star$ is calculated as $\Delta c_{t+k} = c_{t+k} - \hat{c}_{t+k-1}^\star$. This difference Δc_{t+k} is then clipped element-wise to ensure it lies within the bounds $d_{\text{min}}\Delta t \le \Delta c_{t+k} \le d_{\text{max}}\Delta t$, resulting in

$$\Delta c_{t+k}^{\text{clip}} = \text{clip}(\Delta c_{t+k}, d_{\text{min}}\Delta t, d_{\text{max}}\Delta t).$$

The optimized action at time $t+k$ is then updated as

$$\hat{c}_{t+k}^\star = \hat{c}_{t+k-1}^\star + \Delta c_{t+k}^{\text{clip}}.$$

Through this process, the optimized trajectory $\hat{\mathbf{c}}_t^\star$ satisfies the constraints at each time step while remaining as close as possible to the input trajectory $\mathbf{c}_t$ by greedy optimization.

B. Penalty-Based Trajectory Optimization

The goal of penalty-based optimization is to compute an optimized sequence of actions $\hat{\mathbf{c}}_t$ that satisfies the given constraints by leveraging penalties to perform global optimization across all time steps. Unlike constraint clipping,

which operates greedily on each time step, penalty-based optimization considers all time steps jointly.

Denote $\mathbf{c}_{t-1} = \left[\hat{c}_{t-1}^{\mathrm{T}}, c_t^{\mathrm{T}}, c_{t+1}^{\mathrm{T}}, \ldots, c_{t+T_p-2}^{\mathrm{T}}\right]^{\mathrm{T}} \in \mathbb{R}^{T_p D_c}$. Then define the difference sequence $\Delta\mathbf{c}_t = \mathbf{c}_t - \mathbf{c}_{t-1}$. At each iteration, the violations of the constraints are computed as:

$$\mathbf{v}_{\text{max}} = \max(0, \Delta\mathbf{c}_t - \mathbf{d}_{\text{max}}\Delta t),$$

$$\mathbf{v}_{\text{min}} = \max(0, \mathbf{d}_{\text{min}}\Delta t - \Delta\mathbf{c}_t),$$

where $\mathbf{d}_{\text{max}} = \left[d_{\text{max}}^{\mathrm{T}}, d_{\text{max}}^{\mathrm{T}}, \ldots, d_{\text{max}}^{\mathrm{T}}\right]^{\mathrm{T}} \in \mathbb{R}^{T_p D_c}$ is a sequence of upper constraint bounds, and we can also define the sequence of lower constraint bounds $\mathbf{d}_{\text{min}}$ following this format. Here, the max and min operations are applied element-wise to the vector, meaning that each element of the vector is individually processed through the max or min operation. The penalty for the global optimization is then expressed as:

$$\nabla = \mathbf{v}_{\text{max}} - \mathbf{v}_{\text{min}}.$$

We can initialize $\hat{\mathbf{c}}_t = \mathbf{c}_t$ for warm start. The optimized trajectory $\hat{\mathbf{c}}_t$ is updated using penalty:

$$\hat{\mathbf{c}}_t \leftarrow \hat{\mathbf{c}}_t - \eta\nabla,$$

where η is the step size. This process is repeated for a fixed number of iterations or until the maximum violation across all time steps falls below a given tolerance $\epsilon^{\mathbf{v}}$, denoted as $\|\mathbf{v}_{\text{max}}\| + \|\mathbf{v}_{\text{min}}\| < \epsilon^{\mathbf{v}}$. Then we can get the optimized trajectory $\hat{\mathbf{c}}_t^{\star}$. Through this approach, $\hat{\mathbf{c}}_t^{\star}$ satisfies the constraints while remaining as close as possible to the original input trajectory $\mathbf{c}_t$ by penalty-based optimization.

VII. Experimental Details

A. Benchmark with Baselines Across Tasks

In this section, we demonstrate how DiffOG optimizes the trajectories generated by base policies across 13 tasks. The 13 tasks span three types, corresponding to the three action spaces described in Assumption 1. For the end-effector 6-DOF velocity action space, we include five tasks: lift, can, square, tool hang, and transport, all sourced from Robomimic benchmark [12]. For the end-effector linear position action space, we include six tasks: push-T from [5], along with pick place wall, shelf place, disassemble, stick push, and stick pull from Meta-World [33]. Notably, we selected all the "very hard" tasks identified in Meta-World as marked in [34]. Lastly, for the joint angle action space, we include two real-world tasks, arrange desk and move the stack, with data collection and policy experiments conducted using ALOHA [35].

The goal of the arrange desk task is to organize two cups and a bowl placed on the table. This is a dual-arm, long-horizon manipulation task. As shown in Fig. 5, the two arms first individually grasp a bowl and a cup, stacking them together. Next, one arm moves the stack to the location of the coaster. Finally, the other arm picks up the remaining cup and stacks it on top of the stack already placed on the coaster.

The goal of the move the stack task is to pick up a glass cup with a spoon placed on top of it and move it to the target coaster, as shown in Fig. 6. Since the spoon is balanced on the glass cup, the task requires smooth robotic motions; otherwise, the spoon may fall off. Therefore, this task cannot tolerate excessively jerky motions.

For real-world tasks, we randomize the initial positions and orientations of each object while maintaining a consistent placement layout to test the policy's generalization ability; details are provided in the supplementary video. For the move the stack task, the primary goal of our testing is to evaluate the impact of trajectory smoothness on task performance. Therefore, instead of randomizing the placement of cups and coasters across the entire table for data collection and policy rollout, we placed them on opposite sides of the table. This reduces the number of demonstrations required for generalization.

The purpose of the experiments are to investigate the impact of applying trajectory processing, which enforces specific constraints on action trajectories, on policy performance across a wide range of tasks. Additionally, we aim to evaluate whether this approach effectively makes action trajectories smoother and more constrained. The experimental results are shown in Tables I, II, III and Figs. 8 and 9. By analyzing the experimental results, we can draw the following conclusions.

1. In the simulation benchmarks, we evaluate trajectory smoothness by analyzing the first-order derivatives of actions. Specifically, we compare the maximum values and standard deviations of these derivatives throughout the policy rollouts. In Robomimic, the first-order derivative corresponds to angular acceleration in Euler angles, while in Meta-World and push-T, it represents linear velocity. For the real-world tasks, in addition to computing the maximum and standard deviation of the first-order derivatives (i.e., joint angular velocities), we also assess the second-order derivatives (joint angular accelerations) using the same metrics.

These values serve as indicators of trajectory smoothness: the jerkier a trajectory is, the more abrupt and random the variations in the derivatives of the quantities represented in the action space, which is captured by higher standard deviations. Furthermore, jerky trajectories tend to exhibit sharp peaks in velocity or acceleration when sudden changes occur. Therefore, comparing the maximum values of velocity and acceleration also reflects the degree of smoothness.

Trajectory metrics are evaluated under a unified standard, with rollouts terminating upon task completion. To ensure fair and statistically meaningful comparisons, we perform extensive rollouts for each method to obtain quantitative results. For trajectory metrics of simulation benchmarks, we report the average over 1500 rollouts per method (across 50 environments, 3 seeds, and the last 10 checkpoints per seed).

According to Figs. 8 and 9, both DiffOG, constraint clipping, and penalty-based trajectory optimization effectively constrain action trajectories. For Meta-World tasks, DiffOG and trajectory processing baselines achieve lower maximum velocity and reduced velocity standard deviation compared to the base policy trajectories. Similarly, for velocity control

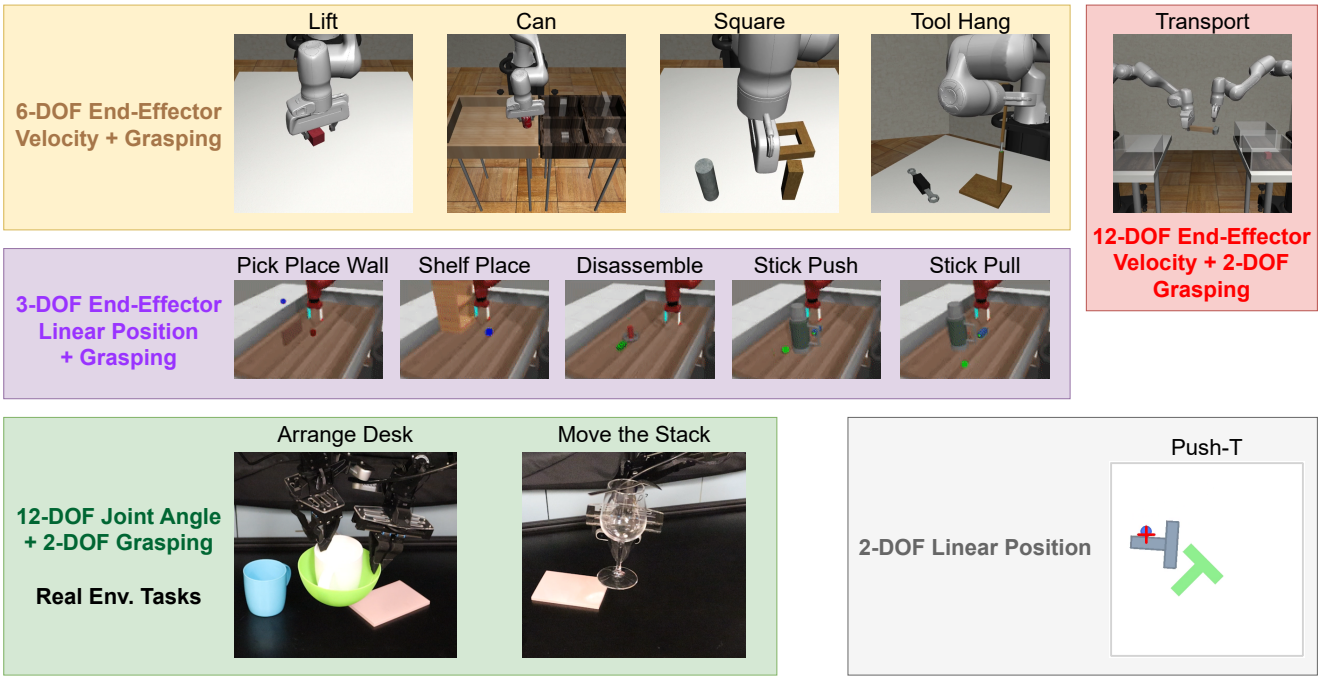


Fig. 4: We validated DiffOG on 13 different tasks. The 13 tasks span three types, corresponding to the three action spaces described in Assumption 1.

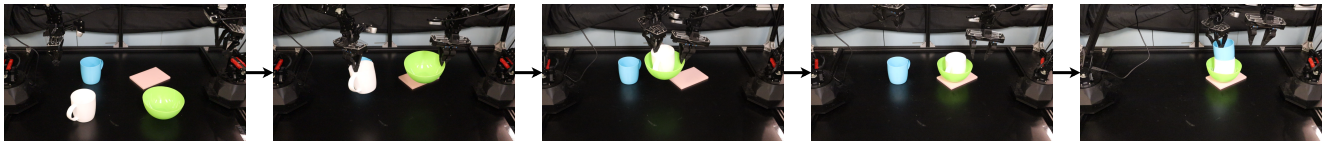


Fig. 5: The process of the arrange desk task.

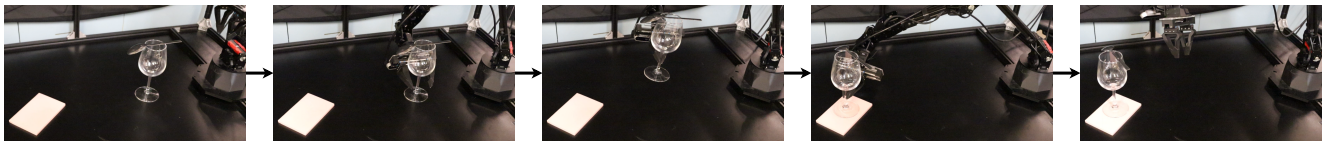


Fig. 6: The process of the move the stack task.

tasks from the Robomimic benchmark, both the maximum acceleration and the standard deviation of acceleration are lower when using DiffOG and trajectory processing baselines compared to the base policy trajectories. Moreover, results of real-world tasks consistently indicate that applying DiffOG and trajectory processing baselines reduces these metrics.

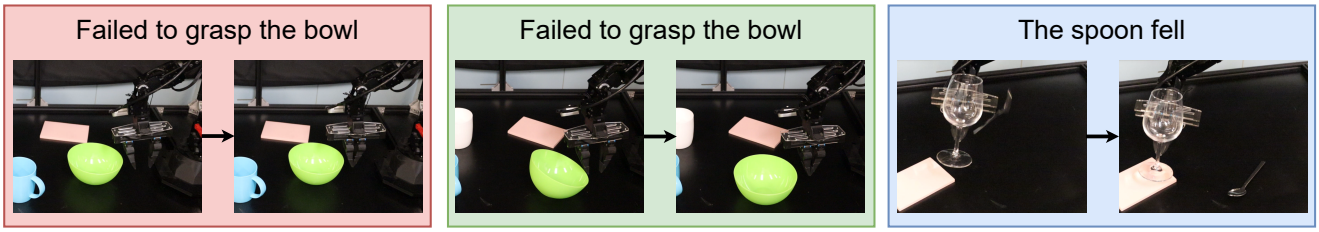
The comparison of these metrics demonstrates that DiffOG, constraint clipping, and penalty-based trajectory optimization can all optimize the action trajectories generated by the policy, making them smoother and more constrained.

2. Based on Tables I, II, and III, we conclude that compared to constraint clipping and penalty-based trajectory optimization, DiffOG generates action trajectories that not only satisfy the constraints but also maintain fidelity to the dataset. As a result, DiffOG outperforms constraint clipping and penalty-based trajectory optimization across all tasks. This demonstrates that DiffOG effectively minimizes the impact on the base policy while further optimizing action trajectories to make them smoother and more constraint-compliant.

3. Although constraint clipping and penalty-based optimization achieve acceptable performance on some tasks, they lead to a significant decline in base policy performance for more challenging tasks, such as square, tool hang, transport, and arrange desk. This decline occurs because the base policy was not trained with constraints as part of its learning objectives. Applying constraint clipping or penalty-based trajectory optimization introduces information loss, resulting in action trajectories that deviate from the demonstration dataset.

As shown in Fig. 7, a common failure in the arrange desk task is the inability to grasp the bowl. This issue arises because trajectory processing baselines cause the actions to deviate from the fidelity of the demonstration, leading to task failures.

In contrast, DiffOG, through a data-driven approach, effectively balances the objectives of trajectory optimization and fidelity to the dataset. Consequently, for these challenging tasks, DiffOG significantly outperforms both constraint clipping and penalty-based trajectory optimization. On the



Penalty-Based Optimization | Constraint Clipping | Base Policy: Diffusion Policy

Fig. 7: Typical failure cases of baselines.

	Lift	Can	Square	Tool Hang	Transport	Push-T
Base Policy: Diffusion Policy	1.00 ± 0.00	0.98 ± 0.01	0.91 ± 0.01	0.83 ± 0.03	0.91 ± 0.01	0.84 ± 0.03
DiffOG Dataset (Ours)	$\mathbf{1.00 \pm 0.00}$	$\mathbf{0.98 \pm 0.01}$	0.87 ± 0.02	$\mathbf{0.82 \pm 0.02}$	0.89 ± 0.02	$\mathbf{0.83 \pm 0.02}$
DiffOG Refine (Ours)	$\mathbf{1.00 \pm 0.00}$	$\mathbf{0.98 \pm 0.01}$	$\mathbf{0.90 \pm 0.01}$	0.81 ± 0.02	$\mathbf{0.91 \pm 0.03}$	0.80 ± 0.02
Constraint Clipping	$\mathbf{1.00 \pm 0.00}$	0.93 ± 0.04	0.78 ± 0.03	0.59 ± 0.03	0.82 ± 0.02	0.82 ± 0.02
Penalty-Based Optimization	$\mathbf{1.00 \pm 0.00}$	0.96 ± 0.02	0.76 ± 0.03	0.79 ± 0.01	0.81 ± 0.02	0.81 ± 0.04

TABLE I: Success rate of simulated tasks using Diffusion Policy [5] as the base policy. We report the mean success rate, calculated as the average over the final 10 checkpoints, with each checkpoint evaluated across 50 different environment initializations. The results are further averaged over 3 training seeds, totaling 150 environment evaluations. We use proficient-human demonstrations of Robomimic benchmark [12]. We highlight in **bold** the method that achieves the highest success rate after applying trajectory processing. DiffOG, penalty-based optimization, and constraint clipping use same constraints and same base policy. On challenging tasks such including square, tool hang, and transport, DiffOG outperforms penalty-based optimization with an average success rate improvement of 9% and surpasses constraint clipping with an average success rate improvement of 14.7%.

	Pick Place Wall	Shelf Place	Disassemble	Stick Push	Stick Pull
Base Policy: DP3	0.98 ± 0.01	0.77 ± 0.04	0.87 ± 0.02	1.00 ± 0.00	0.70 ± 0.02
DiffOG Dataset (Ours)	$\mathbf{0.98 \pm 0.01}$	$\mathbf{0.73 \pm 0.01}$	0.89 ± 0.05	$\mathbf{1.00 \pm 0.00}$	$\mathbf{0.70 \pm 0.03}$
DiffOG Refine (Ours)	$\mathbf{0.98 \pm 0.01}$	0.72 ± 0.02	$\mathbf{0.90 \pm 0.03}$	$\mathbf{1.00 \pm 0.00}$	0.68 ± 0.02
Constraint Clipping	$\mathbf{0.98 \pm 0.01}$	0.70 ± 0.01	0.86 ± 0.05	$\mathbf{1.00 \pm 0.00}$	0.69 ± 0.02
Penalty-Based Optimization	$\mathbf{0.98 \pm 0.01}$	0.67 ± 0.03	0.86 ± 0.06	$\mathbf{1.00 \pm 0.00}$	$\mathbf{0.70 \pm 0.03}$

TABLE II: Success rate of simulated tasks using DP3 [34] as the base policy. We report the mean success rate, calculated as the average over the final 10 checkpoints, with each checkpoint evaluated across 50 different environment initializations. The results are further averaged over 3 training seeds, totaling 150 environment evaluations. We highlight in **bold** the method that achieves the highest success rate after applying trajectory processing. All tasks are trained using 30 demonstrations. DiffOG, penalty-based optimization, and constraint clipping use same constraints and same base policy.

	Base Policy: Diffusion Policy	DiffOG Dataset	Constraint Clipping	Penalty-Based Optimizaiton
Arrange Desk	**9/15**	**9/15**	2/15	4/15
Move the Stack	3/15	**10/15**	6/15	7/15

TABLE III: Success rate of real tasks using Diffusion Policy [34] as the base policy and using ALOHA [35] as experimental platform. We highlight in **bold** the method that achieves the highest success rate.

	Arrange Desk	Move the Stack
Base Policy: Diffusion Policy	0.153 s	0.070 s
DiffOG Layer	0.064 s	0.047 s

TABLE IV: We report the average inference time over 50 inferences for the base policy and DiffOG on 3080. For diffusion policy, we use 16 DDIM [22] steps for arrange desk and 8 DDIM steps for move the stack, with arrange desk requiring more due to its longer horizon and higher DoF. Results demonstrate DiffOG's real-time feasibility.

four challenging tasks: square, tool hang, transport, and arrange desk, DiffOG achieves an average success rate that is 22.7% higher than constraint clipping and 15.1% higher than penalty-based optimization. Compared to the baselines, DiffOG consistently delivers good performance across both simple and challenging tasks.

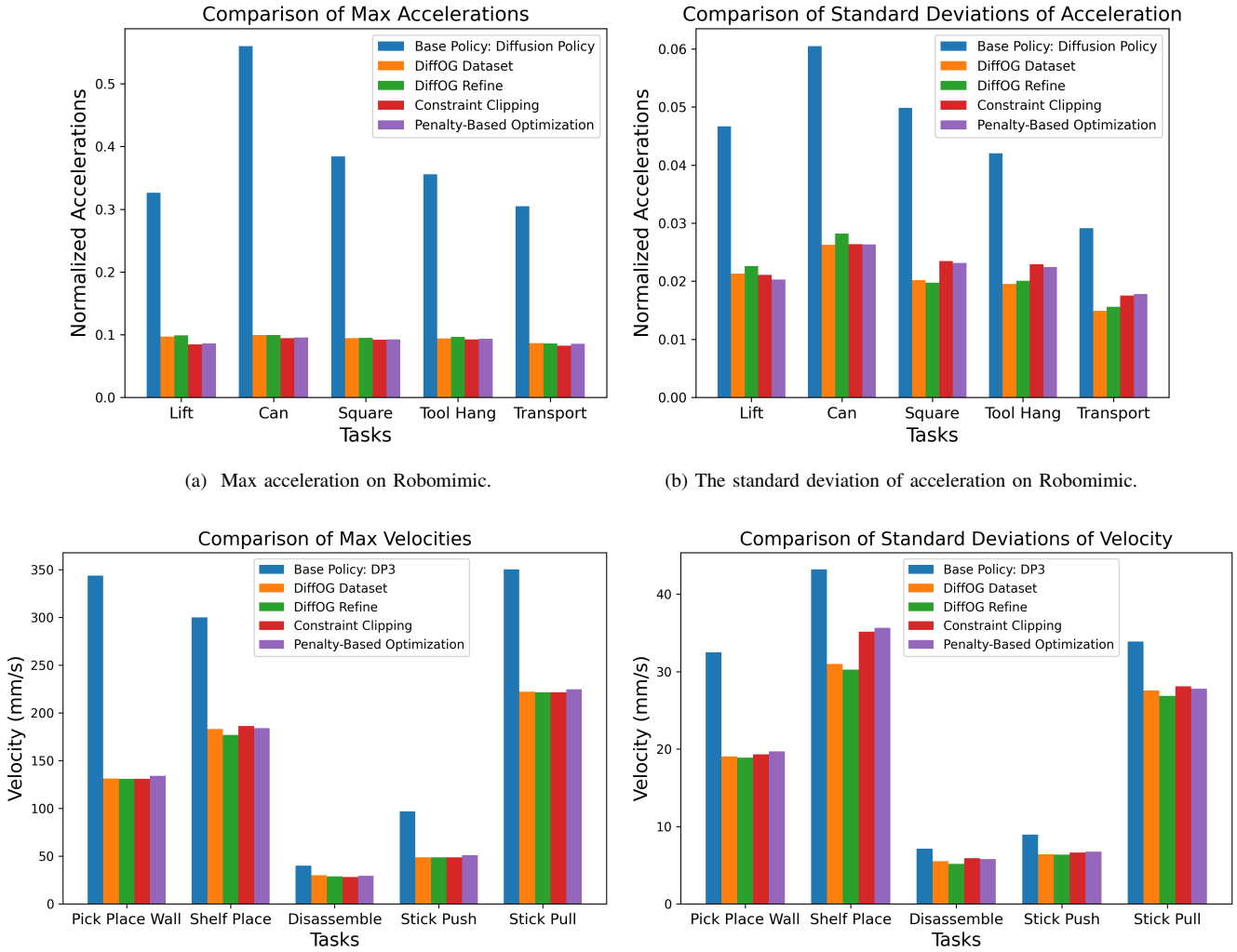
4. For tasks where trajectory constraints and smoothness are crucial, such as move the stack in Table III, the base policy often fails due to the jerkiness of the action trajectories, resulting in a high failure rate. In contrast, DiffOG significantly improves trajectory quality, leading to better policy performance.

5. We report the inference time of DiffOG in real-world experiments, as shown in Fig. IV, demonstrating DiffOG's real-time feasibility.

B. Ablation Study on Adjustability of Trajectory

DiffOG combines model-based and data-driven designs through differentiable trajectory optimization, offering strong interpretability. In this section, we present an ablation study to demonstrate how changes in the smoothing weight α and the constraint bounds $d_{\max}\Delta t$ and $d_{\min}\Delta t$ affect the action trajectories generated by the differentiable trajectory optimization learned by DiffOG. The results are shown in Fig. 10.

Fig. 10(a) corresponds to the scenario where the smoothing weight varies while the constraint bound remains fixed at 0.1. From Fig. 10(a), we observe that an excessively large smoothing weight slightly reduces the policy's performance. This is reasonable because when the smoothing weight is very large, the objective during DiffOG's learning process prioritizes fitting the demonstration with extremely smooth



(a) Max acceleration on Robomimic.

(b) The standard deviation of acceleration on Robomimic.

(c) Max velocity on Meta-World.

(d) The standard deviation of velocity on Meta-World.

Fig. 8: Comparison of trajectory metrics across methods on Robomimic and Meta-World benchmarks. These results are derived from 1500 rollouts (across 50 environments, 3 seeds, and the last 10 checkpoints per seed), providing strong statistical significance. For Robomimic tasks, the larger maximum acceleration and greater standard deviation of acceleration over time indicate less smooth trajectories. Similarly, for Meta-World tasks, this observation applies to the velocity metrics. Since the actions provided by the Robomimic dataset are already normalized [12], the acceleration metrics shown here for Robomimic are also normalized. For details on the calculation of acceleration and velocity metrics, as well as the details to unnormalize the Robomimic metrics, please refer to Appendix.

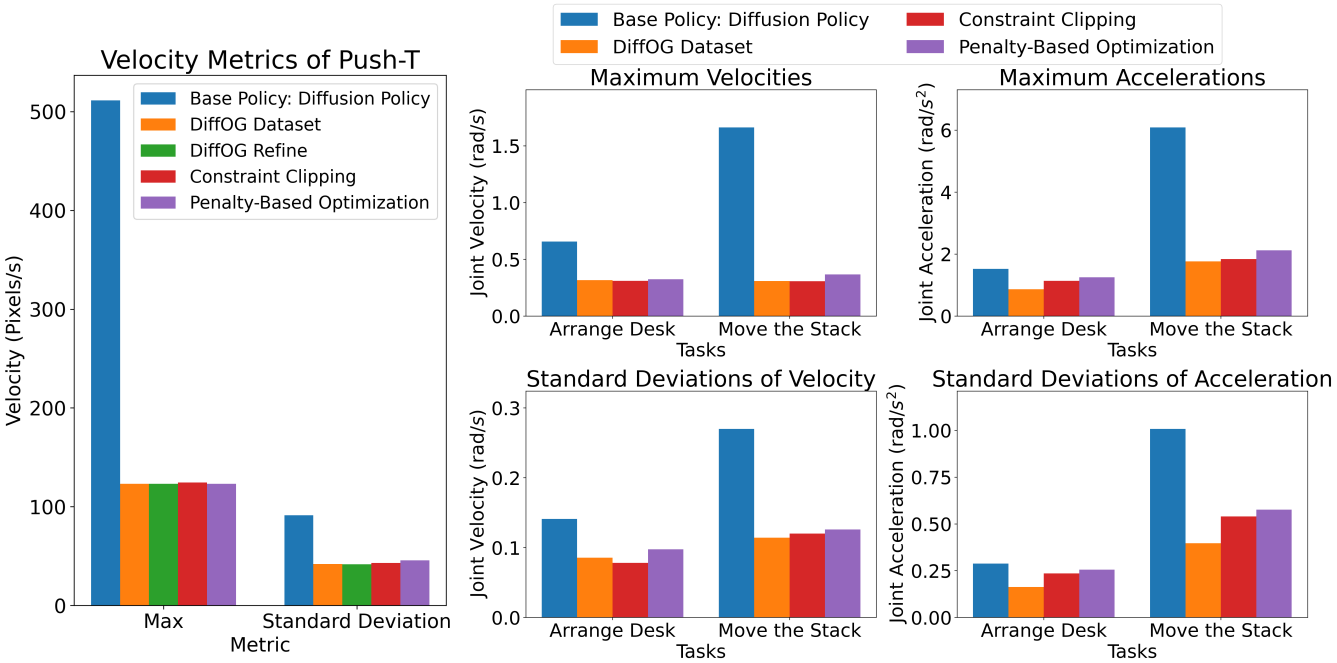
trajectories. In such cases, the fidelity of the learned trajectories to the demonstration decreases.

On the other hand, we observe that when the smoothing weight changes but the constraint bounds remain unchanged, the generated trajectories consistently adhere to the constraints. However, as the smoothing weight increases, the trajectories become progressively smoother, as evidenced by a decreasing standard deviation of acceleration. Thus, changes in the smoothing weight directly affect the characteristics of the generated trajectories.

In practical applications of DiffOG, it is essential to select an appropriate smoothing weight that balances achieving superior smoothness with maintaining policy performance. Based on Fig. 10(a), we selected a smoothing weight of $\alpha = 4$ for the Robomimic benchmark.

Fig. 10(b) corresponds to the scenario where the constraint bounds vary while the smoothing weight remains fixed at 4. From Fig. 10(b), we observe that overly small constraint bounds lead to a decline in policy performance. This is because when the constraint bounds are very tight, the objective during DiffOG's learning process becomes fitting the demonstration with highly constrained trajectories. Under such conditions, the fidelity of the learned trajectories to the demonstration decreases.

When we adjust the constraint bounds, the maximum acceleration of the generated trajectory changes accordingly, and the smoothness of the trajectory is also affected. This indicates that in practical applications of DiffOG, it is crucial to select appropriate constraint bounds that effectively constrain the trajectory while maintaining policy performance. Based



(a) Comparison of trajectory metrics on push-T.

(b) Comparison of trajectory metrics on real-world tasks.

Fig. 9: Comparison of trajectory metrics across methods on push-T and real-world tasks. The larger maximum velocity and acceleration and greater standard deviation of velocity and acceleration over time indicate less smooth trajectories. For details on the calculation of acceleration and velocity metrics, please refer to Appendix.

on Fig. 10(b), we selected a constraint bound of 0.1 for the Robomimic benchmark.

C. Ablation Study on Static $\mathbf{Q}$*, Matrix Learning, and Transformer*

From equation (9), we can observe that if we set $\mathbf{Q} = -\mathbf{I}$, the term $\left\|\mathbf{R}\mathbf{y}_t - \mathbf{g}_t\right\|^2$ in the optimization problem (8) simplifies to $\left\|\mathbf{y}_t - \mathbf{a}_t\right\|^2$, because $\mathbf{Q} = \mathbf{R}^{\mathrm{T}}\mathbf{R}$ and $\mathbf{g}_t = -(\mathbf{R}^{\mathrm{T}})^{-1}\mathbf{a}_t$. Under this setting, (8) becomes a trajectory optimization problem with both a smoothing cost and a distance cost. On the other hand, instead of using a transformer as in DiffOG, we can also directly learn the matrix $\mathbf{Q}$. In this section, we conduct experiments on the following three different cases:

1. When $\mathbf{Q} = -\mathbf{I}$ and is kept static. In this case, the trajectory optimization is non-learnable.

2. When $\mathbf{Q} = \mathbf{L}\mathbf{L}^{\mathrm{T}} + \epsilon\mathbf{I}$ and the matrix $\mathbf{L}$ is directly learned. In this case, the trajectory optimization is differentiable, but its composition consists solely of matrices without neural networks. Both dataset training and refinement training can be performed in this setup.

3. DiffOG, where $\mathbf{Q}$ is generated by a transformer. In this case, the trajectory optimization is differentiable. Both dataset training and refinement training can also be performed.

We selected the transport task as the test task. This is a dual-arm, long-horizon manipulation task, which poses challenges for trajectory optimization in terms of generalizing to trajectories of varying forms within long horizon manipulation and high-dimensional action space. The experimental results are shown in Table V. From the table, we observe that DiffOG refine training achieves the best performance.

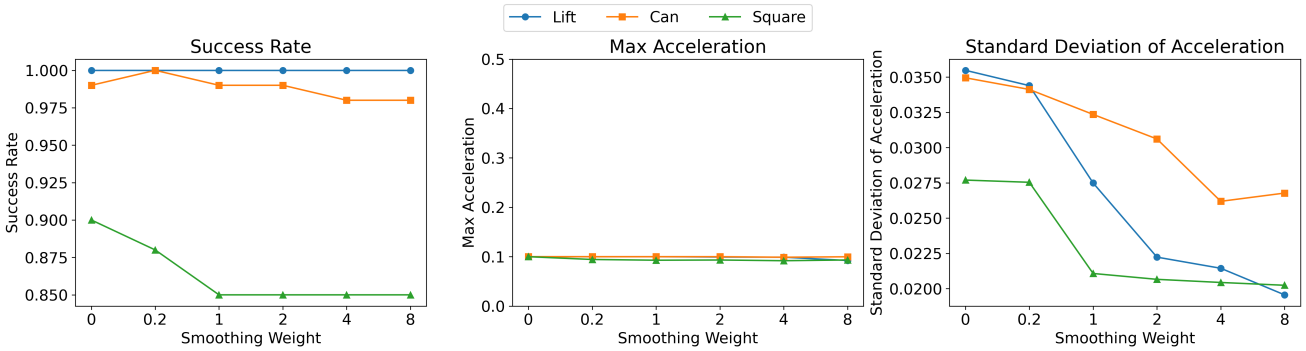
The explanation behind this is that for static $\mathbf{Q}$ and matrix-learning-based $\mathbf{Q}$, trajectory optimization cannot effectively adjust to different trajectory inputs. This highlights the flexibility and generalization capability of DiffOG, thanks to the transformer-based embedding design.

Additionally, during refine training, due to the presence of a multi-modal action distribution in demonstration dataset [5, 7], the actions generated by the pre-trained policy may differ from the targets in the dataset. This conflict between the targets and inputs does not exist in dataset training paradigm of DiffOG and matrix learning. From the results of matrix-learning refine and DiffOG refine, we can observe that trajectory optimization based on matrix learning struggles to handle such conflict. However, DiffOG can handle this conflict with transformer encoder.

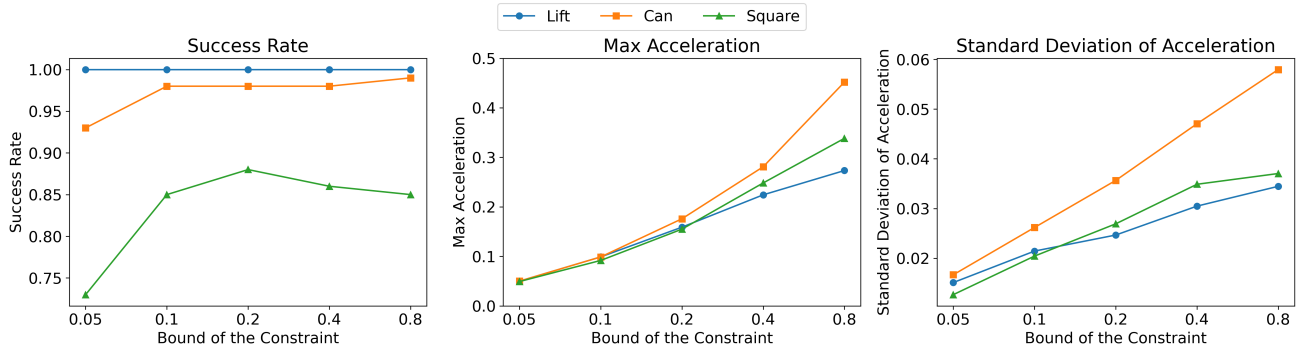
D. Comparison with a Constrained Visuomotor Policy

We conducted comparison experiments between DiffOG and constrained visuomotor policy Leto [32] on 7 tasks. The results are shown in Table VI. The results in the table compare DiffOG and Leto under the same constraint bounds.

Based on the results in Table VI, we can see that the performance of policies enhanced by DiffOG is entirely superior to Leto policies. This demonstrates the generalization capability of DiffOG across diverse tasks.



(a) The smoothing weight varies while the constraint bound remains fixed at 0.1.



(b) The constraint bounds vary while the smoothing weight α remains fixed at 4.

Fig. 10: Ablation study on adjustability of trajectory. We use DiffOG dataset training pipeline as shown in Fig. 2. For success rate, we report the mean success rate calculated as the average over the final 10 checkpoints, with each checkpoint evaluated across 50 different environment initializations. Since our primary goal is to observe the trend in metric variations as the smoothing weight and constraint bound change, we conducted experiments using a single seed, leading to slight variations in the mean success rate compared to the three-seed results reported in Table I.

	DiffOG Dataset	DiffOG Refine	Static $\mathbf{Q}$	Matrix Learning Dataset	Matrix Learning Refine
Transport	0.89	**0.91**	0.84	0.85	0.14

TABLE V: Success rate of different $\mathbf{Q}$ designs on transport task.

	DiffOG Dataset	DiffOG Refine	Leto
Square	0.87	**0.90**	0.75
Tool Hang	**0.82**	0.81	0.28
Push-T	**0.83**	0.80	0.64
Pick Place Wall	**0.98**	**0.98**	0.63
Shelf Place	**0.73**	0.72	0.15
Disassemble	0.89	**0.90**	0.55
Stick Pull	**0.70**	0.68	0.20

TABLE VI: Success rate of benchmarking DiffOG with Leto [32].

VIII. Details of Training, Evaluation, and Dataset

A. Hyperparameters and Dataset

The summary of hyperparameters for trajectory optimization and dataset as shown in Table VII. In the experiments of this paper, we consistently adopt symmetric constraints which is $d_{\max} = -d_{\min}$. Since the actions require normalization to [-1,1] during training [5], all constraints are applied on the normalized action scale for DiffOG. For simplicity, when applying constraints on the normalized action scale, the same value is used for each action dimension. Therefore, the "Bound" column contains only a single value for each task. Since the distribution of action values differs across task datasets, the bounds for each action dimension may vary after unnormalization. For Robomimic tasks [12], the action values in the dataset are already normalized to the range of $[-1, 1]$. Therefore, the bound and the unnormalized bound are identical.

We can observe that, except for Robomimic tasks, the physical values of the constraint bounds can be computed by dividing the "Unnormalized Bounds" column by the Δt column. For instance, in real tasks such as move the stack and arrange desk, this calculation yields the upper and lower limits of joint angular velocities (measured in rad/s).

The bound of 0.1 for Robomimic tasks represents normalized acceleration. To convert the it back to the raw form, use a scalar of 20 $\mathrm{m/s^2}$ for linear acceleration and 200 $\mathrm{rad/s^2}$ for rotational acceleration of Euler angles [12]. For example, the maximum linear acceleration is $0.1 \times 20\ \mathrm{m/s^2} = 2\ \mathrm{m/s^2}$.

Since constraint clipping and penalty-based trajectory optimization do not require training, constraints can be directly applied to unnormalized actions. These constraints are equivalent to those imposed by DiffOG but are applied directly as unnormalized constraints. The parameters for penalty-

Task	Bound	Unnormalized Bound	Demos	Action Space	Δt	α	High Dim. Obs.
Lift	0.1	0.1	200	6-DOF Vel. + 1-DOF Grasping	0.05 s	4	$2 \times 3 \times 84 \times 84$
Can	0.1	0.1	200	6-DOF Vel. + 1-DOF Grasping	0.05 s	4	$2 \times 3 \times 84 \times 84$
Square	0.1	0.1	200	6-DOF Vel. + 1-DOF Grasping	0.05 s	4	$2 \times 3 \times 84 \times 84$
Tool Hang	0.1	0.1	200	6-DOF Vel. + 1-DOF Grasping	0.05 s	4	$2 \times 3 \times 240 \times 240$
Transport	0.1	0.1	200	12-DOF Vel. + 2-DOF Grasping	0.05 s	4	$4 \times 3 \times 84 \times 84$
Push-T	0.05	[12.5,12.2] Pixels	90	2-DOF Position	0.1 s	1	$1 \times 3 \times 96 \times 96$
Pick Place Wall	0.6	[0.53, 0.92, 3.75] mm	30	3-DOF Pos. + 1-DOF Grasping	0.0125 s	4	3×512
Shelf Place	0.6	[2.32, 1.47, 4.13] mm	30	3-DOF Pos. + 1-DOF Grasping	0.0125 s	4	3×512
Disassemble	0.6	[0.65, 1.10, 0.61] mm	30	3-DOF Pos. + 1-DOF Grasping	0.0125 s	4	3×512
Stick Push	0.6	[1.03, 0.15, 0.68] mm	30	3-DOF Pos. + 1-DOF Grasping	0.0125 s	4	3×512
Stick Pull	0.6	[4.46, 1.44, 2.49] mm	30	3-DOF Pos. + 1-DOF Grasping	0.0125 s	4	3×512
Arrange Desk	0.1	[0.070, 0.071, 0.102, 0.030, 0.056, 0.148, 0.078, 0.089, 0.105, 0.039, 0.069, 0.120] rad	100	12-DOF Angle + 2-DOF Grasping	0.25 s	4	$3 \times 3 \times 240 \times 320$
Move the Stack	0.05	[0.049, 0.051, 0.057, 0.045, 0.038, 0.069] rad	100	6-DOF Angle + 1-DOF Grasping	$\frac{1}{6}$ s	4	$2 \times 3 \times 240 \times 320$

TABLE VII: Summary of hyperparameters for trajectory optimization and dataset. α is the smooth weight, $\frac{1}{\Delta t}$ represents the frequency of policy inference, and "Demos" indicates the number of demonstration episodes. "High Dim. Obs" refers to the dimensionality of high-dimensional observations. For instance, $2 \times 3 \times 84 \times 84$ corresponds to RGB observations from two cameras, each with a shape of $3 \times 84 \times 84$, while 3×512 represents point cloud data with a shape of 3×512. "Bound" refers to $d_{\max}\Delta t$ in equation (7). "Unnormalized Bound" refers to constraint bounds unnormalized to dataset scale.

Task	Step Size η	Tolerance ϵ^{v}	Max Iterations
Robomimic Tasks	0.03	0.004	10000
Push-T	0.1	0.1	1000
Meta-World Tasks	0.1	0.05	1000
Real ALOHA tasks	0.05	0.01	1000

TABLE VIII: Parameters of penalty-based trajectory optimization.

Dropout	Heads	Layers	Embed. Dim.	Feed. Dim.
0.1	4	2	256	256

TABLE IX: Hyperparameters of transformer encoder.

based trajectory optimization are listed in Table VIII. As mentioned earlier, the actions provided by the Robomimic dataset are already normalized to the range of $[-1, 1]$. The constraint bound we impose in this case is 0.1. Based on this, the violation tolerance for penalty-based optimization is set to a relatively small value of 0.004. For other tasks, the constraints are applied to raw action values, and the violation tolerances are determined based on the scale of unnormalized bounds provided in Table VII.

Finally, we provide the hyperparameters of the transformer used in DiffOG, as shown in Table IX.

B. Computation of Trajectory Metrics

In this section, we introduce details of computing trajectory metrics shown in Figs. 8 and 9.

The trajectory metrics shown in Figs. 8 and 9 are calculated as the average over the final 10 checkpoints, with each checkpoint evaluated across 50 different environment initializations. The results are further averaged over 3 training seeds, totaling 150 environment evaluations.

For position control, we present the velocity metric; for velocity control, we present the acceleration metric. For joint angle control, we provide not only the velocity metric (first derivative) but also the acceleration metric (second derivative). For a single evaluation in a specific environment, the calculation of "max" involves taking the maximum value of each action dimension across all time steps and then averaging these maximum values across all action dimensions. The calculation of "standard deviation" involves computing the standard deviation of the metric for each action dimension over time and then averaging these standard deviations across all action dimensions. This calculation of the standard deviation reflects the degree of jerkiness in the trajectory over time steps.

The acceleration metrics for Robomimic tasks are normalized values, and are averaged on linear accelerations and rotational acceleration of Euler angles. Please refer to Section VIII-A to see more information of unnormalizing them to their raw form, which are measured in $\mathrm{m/s^2}$ and $\mathrm{rad/s^2}$.

IX. Limitations and Future Work

Since this paper focuses on supervised policy learning, both training paradigms of DiffOG, dataset training and refine training, depend on pre-collected demonstrations. This reliance limits DiffOG's ability to generalize to diverse trajectories beyond those in the dataset. Moreover, if the pre-collected dataset contains extremely jerky trajectories, standard supervised learning methods may struggle to substantially optimize these trajectories while preserving fidelity to the original dataset. A potential future direction involves leveraging reinforcement learning, which could enable fine-tuning of DiffOG on top of the existing dataset. This strategy would allow the model to incorporate multiple objectives, such as trajectory constraints and smoothness, without requiring extensive reward engineering. By simultaneously fine-tuning both the pre-trained policy and DiffOG itself, reinforcement learning could also facilitate broader exploration of trajectory diversity, ultimately enhancing DiffOG's overall effectiveness.

X. Conclusion

We introduced DiffOG, a learning-based trajectory optimization framework that combines the strengths of neural networks with an interpretable differentiable optimization layer. By integrating trajectory smoothness, adherence to hard constraints, and fidelity to demonstrations into the training objectives, DiffOG enables visuomotor policies to produce smoother and safer action trajectories while maintaining performance. Through extensive evaluations on 11 simulated tasks and 2 real-world tasks, we demonstrated that DiffOG consistently outperforms existing trajectory processing baselines, such as greedy constraint clipping and penalty-based trajectory optimization, as well as constrained visuomotor policy Leto [32]. Our results demonstrate the generalization capability of DiffOG across diverse tasks.