Safe Offline Reinforcement Learning using Trajectory-Level Diffusion Models

Ralf Römer, Lukas Brunke, Martin Schuck, and Angela P. Schoellig

***Abstract*— Despite its success in controlling robotic systems, reinforcement learning (RL) suffers from several issues that hinder its widespread adoption in real-world scenarios. Recently, diffusion models have emerged as a powerful tool to address some of the longstanding challenges in offline and model-based RL, improving long-horizon planning and facilitating multitask generalization. However, these algorithms are unsuitable for operating in unseen and dynamic environments where novel and time-varying constraints not represented in the training data may arise. To address this issue, we propose incorporating a projection scheme into diffusion-based trajectory generation. Our approach uses the iterative nature of diffusion models and alternates the conditional backward diffusion process with a projection of the noisy trajectory onto the constraint set. As a result, we can generate trajectories that are both safe and dynamically feasible while still achieving high reward. We evaluate our approach for goal-conditioned offline RL for two simulated robotic systems navigating in environments with static and dynamic obstacles, representing novel test-time constraints. We show that our method can satisfy these constraints in closed loop, greatly increasing the success rate of reaching the goal.**

I. Introduction

Reinforcement learning (RL) [1] has been successful in controlling robotic systems directly using sensory inputs [2], [3]. However, RL algorithms still perform poorly in terms of sample efficiency, multitask generalization, and long-horizon planning. Another obstacle to widely deploying RL algorithms on real-world robotic systems is the concern of safety [4], e.g., a self-driving car adhering to lane boundaries. Since RL algorithms rely on gathering data by applying potentially unsafe explorative actions, the community has made an effort to augment existing RL methods to account for safety [5]. Despite this effort, safe RL algorithms typically do not provide safety guarantees but only encourage safety.

A promising approach to increase sample efficiency is leveraging model-based RL [6], which explicitly models the transition probabilities of the system. While model-based RL requires fewer samples, it often converges to subpar policies compared to model-free RL and suffers from accumulating prediction errors in learned models for long-horizon tasks. Another technique that aims at the applicability of RL in the real world is offline RL [7]. Instead of directly interacting with a system during training, the RL algorithm only has access to an offline-collected dataset of interactions. This also increases the reusability of data as different policies can be trained on the same dataset. However, safely transferring offline learned policies to the real world remains challenging.

Generative models have recently gained much attention for tasks such as image [8], [9] or video generation [10]. Earlier approaches, such as variational autoencoders (VAEs) [11] or generative adversarial networks (GANs) [12], use a single step to transform random noise into a sample from the learned distribution. In contrast, diffusion models [13], [14] perform this transformation iteratively through several denoising steps. Due to their ability to capture high-dimensional multi-modal distributions, they have been the backbone for many recent state-of-the-art generative models. Due to these advances and properties, diffusion models have recently been employed in offline RL [15], [16]. Notably, the authors in [15] propose to learn a diffusion model over trajectories and show that it can outperform other offline RL algorithms. The approach merges system dynamics modeling and trajectory optimization by directly modeling the trajectory distribution, thereby avoiding accumulating single-step errors. While this approach allows generating trajectories satisfying constraints from the training data [17], it cannot handle novel or time-varying constraints, which often arise in real-world robotics applications, e.g., in the form of moving obstacles. Hence, guaranteeing the safety of the sampled trajectories remains an open challenge, hindering the application of diffusion models for robot decision-making.

Previous work [18] has aimed to incorporate safety into the diffusion process by leveraging control-barrier functions as safety filters. However, this approach is limited to state constraints expressed by continuously differentiable functions and significantly increases the computation time to generate new samples. In contrast, our work proposes a novel, computationally efficient, safe trajectory generation method for diffusion-based offline RL. Our main contributions are:

- We formulate goal-conditioned offline RL with state and action constraints as a conditional generative modeling problem and address it by combining diffusion models for learning trajectory distributions with classifier-guided sampling [19].
- We incorporate an efficient projection method into the backward diffusion process, allowing the generation of safe trajectories while being computationally cheap. Moreover, we use a receding-horizon control approach and adopt a warm-starting strategy to improve the consistency of the generated trajectories and reduce the computational demand.
- We demonstrate in simulation of two robotic systems that our method can satisfy novel constraints, increasing

The authors are with the Learning Systems and Robotics Lab (LSY), School of Computation, Information and Technology, and the Munich Institute for Robotics and Machine Intelligence (MIRMI), Technical University of Munich, Germany. Email: {ralf.roemer; lukas.brunke; martin.schuck; angela.schoellig}@tum.de.

the success rate of reaching a goal in environments with static and dynamic obstacles.

II. Related Work

Diffusion Models in Robotics: Following their success in text-to-image [8], [9] and text-to-video generation [10], diffusion models [13], [14] have recently been applied to various robotics-related tasks. Motion planning diffusion (MPD), proposed in [20], trains a diffusion model on expert state-trajectories and uses the diffusion model as prior for bootstrapping an optimization-based motion planning algorithm. In [21], cost functions are learned using diffusion models to facilitate a joint optimization of grasp poses and motions instead of the traditional modular approach. Diffusion models have also been employed for imitating expert behaviors, both as planner [18], [22] and policy [23]. DALL-E-Bot [24] performs task planning by generating desired poses with a diffusion-based text-to-image generator. Diffusion models have also been used to accelerate the robot learning process by generating synthetic training data [25].

Offline RL The broader field of offline RL has gained considerable attention in recent years due to its promise to leveraging large, existing datasets for decision-making problems without the need for online data collection [7]. To overcome the overestimation of values caused by the distributional shift between the dataset and the learned policy, [26] learn a lower-bound estimate of the Q-function and demonstrate the effectiveness of the approach in several robot manipulation environments on a limited dataset of human examples. Several works [27]–[29] have also proposed to model offline RL as a sequence modeling problem that can be learned by modified transformer architectures.

Diffusion Models for Offline RL: In the seminal work [15], an unconditional diffusion model is trained on an offline dataset of state-action trajectories. Online, new plans are generated via classifier-guided sampling [19] using a separately trained reward prediction model. Several modifications to this approach have been proposed. Notably, [17] generate state-only trajectories and learn an inverse dynamics model to compute the corresponding actions. Some works [17], [30] have adopted a classifier-free guidance scheme [31] and directly train a conditional diffusion model. The main drawback of this approach is that integrating new conditioning variables requires retraining the model from scratch. Guided sampling schemes can satisfy constraints seen in the training data or novel combinations of those constraints [17]. However, integrating explicit and previously unseen constraints into diffusion-based trajectory generation has hardly been addressed so far. As the only representative work in this direction, [18] uses control barrier functions (CBFs) to specify constraints and re-optimizes the dynamics of the backward diffusion process to generate safe trajectories with probability almost 1. However, this method can only handle continuously differentiable constraints and requires the construction of suitable CBFs, which can be challenging in practice.

III. Background on Diffusion Models

Diffusion models are a class of generative models that aim to learn an unknown target distribution $q(\boldsymbol{x})$, where $\boldsymbol{x} \in \mathcal{X}$, from samples $\boldsymbol{x} \sim q(\cdot)$. This is done by constructing a forward diffusion process that gradually transforms the data into noise and learning a reverse denoising process to reconstruct the data. There are two popular formulations of diffusion models: Denoising diffusion probabilistic models (DDPM) [14] and score-based generative models [32]. We focus on DDPMs due to their simplicity and since we do not use the score-based formulation's continuous-time perspective on the diffusion process. DDPMs are based on introducing latent variables $\boldsymbol{x}^1, \dots, \boldsymbol{x}^k \in \mathcal{X}$. They construct a forward diffusion Markov process

$$q\big(\boldsymbol{x}^k|\boldsymbol{x}^{k-1}\big) = \mathcal{N}\left(\sqrt{1-\beta_k}\boldsymbol{x}^k, \beta_k \boldsymbol{I}\right), \tag{1}$$

where $\boldsymbol{x}^0 = \boldsymbol{x}$, $k = 1, \dots, K$, is the diffusion time step and $\beta_k \in (0, 1)$ is a noise schedule. Since the transition dynamics (1) are Gaussian, we can directly sample $\boldsymbol{x}^k$ from $\boldsymbol{x}^0$ via $q\big(\boldsymbol{x}^k|\boldsymbol{x}^0\big) = \mathcal{N}\left(\sqrt{\alpha_k}\boldsymbol{x}^0, (1-\alpha_k)\boldsymbol{I}\right)$, where $\alpha_k = \prod_{i=1}^{k}(1-\beta_i)$, bypassing the iterative sampling from (1). The noise schedule and the number of diffusion steps $K \in \mathbb{N}$ are chosen such that $q\big(\boldsymbol{x}^K|\boldsymbol{x}^0\big) \approx \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$, i.e., the forward process gradually transforms the data into Gaussian noise. This process is reversed by the learnable backward diffusion process

$$p_{\boldsymbol{\theta}}\left(\boldsymbol{x}^{k-1}|\boldsymbol{x}^k, k\right) = \mathcal{N}\left(\boldsymbol{\mu}_{\boldsymbol{\theta}}\left(\boldsymbol{x}^k, k\right), \boldsymbol{\Sigma}_{\boldsymbol{\theta}}\left(\boldsymbol{x}^k, k\right)\right), \tag{2}$$

where $\boldsymbol{\mu}_{\boldsymbol{\theta}}$ and $\boldsymbol{\Sigma}_{\boldsymbol{\theta}}$ can be parameterized by neural networks. The objective for learning the parameters $\boldsymbol{\theta}$ is to match the joint distributions in the forward and backward process, i.e., $q\big(\boldsymbol{x}^0, \dots, \boldsymbol{x}^k\big)$ and $p_{\boldsymbol{\theta}}\big(\boldsymbol{x}^0, \dots, \boldsymbol{x}^k\big)$, by minimizing their KL divergence. As shown in [14], for the fixed variance schedule $\boldsymbol{\Sigma}_{\boldsymbol{\theta}}\left(\boldsymbol{x}^k, k\right) = \boldsymbol{\Sigma}_k = \beta_k \boldsymbol{I}$, this can be achieved by minimizing the surrogate loss

$$\min_{\boldsymbol{\theta}} \mathbb{E}_{k, \boldsymbol{x}^0, \boldsymbol{\epsilon}}\left[\left\|\boldsymbol{\epsilon} - \boldsymbol{\epsilon}_{\boldsymbol{\theta}}\big(\sqrt{\alpha_k}\boldsymbol{x}^0 + \sqrt{1-\alpha_k}\boldsymbol{\epsilon}, k\big)\right\|_2\right], \tag{3}$$

where $k \sim \mathcal{U}([1, K])$, $\boldsymbol{x}^0 \sim q(\cdot)$ and $\boldsymbol{\epsilon} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$. Here, $\boldsymbol{\epsilon}_{\boldsymbol{\theta}}$ can be expressed as a function of $\boldsymbol{\mu}_{\boldsymbol{\theta}}$; see [14]. New samples from $q(\boldsymbol{x})$ can be generated by drawing a noisy sample $\boldsymbol{x}^k \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$ and then iteratively sampling from (2) with the learned mean

$$\boldsymbol{\mu}_{\boldsymbol{\theta}}\left(\boldsymbol{x}^k, k\right) = \frac{1}{\sqrt{1-\beta_k}}\left(\boldsymbol{x}^k - \frac{\beta_k}{\sqrt{1-\alpha_k}}\boldsymbol{\epsilon}_{\boldsymbol{\theta}}(\boldsymbol{x}^k, k)\right). \tag{4}$$

IV. Problem Setup

We consider a robotic system with state $\boldsymbol{s}_t \in \mathcal{S} \subset \mathbb{R}^n$ and action $\boldsymbol{a}_t \in \mathcal{A} \subset \mathbb{R}^m$ at timestep $t \in \mathbb{N}_0$. The state space $\mathcal{S}$ and action space $\mathcal{A}$ are continuous. The system is governed by the unknown discrete-time dynamics

$$\boldsymbol{s}_{t+1} = \boldsymbol{f}(\boldsymbol{s}_t, \boldsymbol{a}_t). \tag{5}$$

We aim to achieve a task-specific goal $\boldsymbol{g} \in \mathcal{G}$, which could, for example, correspond to reaching a certain region of the state space. A binary function $\varphi : \mathcal{S} \times \mathcal{G} \to \{0, 1\}$ indicates whether the goal is achieved at a particular state,

in which case $\varphi(\boldsymbol{s}_t, \boldsymbol{g}) = 1$. The actions are chosen by a stochastic policy $\boldsymbol{\pi}$, i.e., $\boldsymbol{a}_t \sim \boldsymbol{\pi}(\cdot|\boldsymbol{s}_t, \boldsymbol{g})$. The quality of a state-action pair with respect to the goal is measured by a reward function $r : \mathcal{S} \times \mathcal{A} \times \mathcal{G} \to \mathbb{R}$, and we aim to maximize the expected value of the discounted sum of rewards $\sum_{t=0} \gamma^t r(\boldsymbol{s}_t, \boldsymbol{a}_t, \boldsymbol{g})$, where $\gamma \in (0, 1)$. In contrast to the standard RL setting, we additionally aim to satisfy time-dependent state and input constraints

$$\boldsymbol{s}_t \in \mathcal{S}_t \subseteq \mathcal{S}, \qquad \boldsymbol{a}_t \in \mathcal{A}_t \subseteq \mathcal{A}, \tag{6}$$

$\forall t = 0, 1, \ldots$, that are only available at test-time, i.e., *not during training*. Such constraints are very common in robotics, for example, in the form of collision constraints when operating in dynamic environments. Our objective can be formulated as solving the constrained stochastic optimal control problem

$$\begin{aligned} \boldsymbol{\pi}^* = \arg\max_{\boldsymbol{\pi}} \mathbb{E}_{\boldsymbol{a}_t \sim \boldsymbol{\pi}(\cdot|\boldsymbol{s}_t, \boldsymbol{g})} \left[\sum_{t=0} \gamma^t r(\boldsymbol{s}_t, \boldsymbol{a}_t, \boldsymbol{g}) \right] \\ \text{s.t.} \quad (6), \qquad \forall t, \end{aligned} \tag{7}$$

to obtain a constraint-satisfying return-maximizing policy.

V. Methodology

In this section, we propose a tractable way to approximately solve (7) using trajectory-level diffusion models. First, we simplify (7) by considering a fixed finite horizon $T \in \mathbb{N}$. We define a trajectory of system (5) as $\boldsymbol{Z} = (\boldsymbol{s}_0, \boldsymbol{a}_0, \ldots, \boldsymbol{s}_T, \boldsymbol{a}_T)$ and the corresponding cumulative reward as $R_{\boldsymbol{g}}(\boldsymbol{Z}) = \sum_{t=0}^{T} r(\boldsymbol{s}_t, \boldsymbol{a}_t, \boldsymbol{g})$. Then, the finite-horizon version of (7) can be written as

$$\begin{aligned} \boldsymbol{\pi}^* = \arg\max_{\boldsymbol{\pi}} \mathbb{E}_{\boldsymbol{a}_t \sim \boldsymbol{\pi}(\cdot|\boldsymbol{s}_t, \boldsymbol{g})} \left[R_{\boldsymbol{g}}(\boldsymbol{Z}) \right] \\ \text{s.t.} \quad (6), \qquad \forall t \in \{0, \ldots, T\}. \end{aligned} \tag{8}$$

Common approaches to solving (8) such as learning-based model predictive control (MPC) [33] or safe model-based RL [34] first learn an approximation of the one-step dynamics model (5) and then optimize by rolling out the model. A drawback of this approach is that it suffers from the compounding error problem, which is exacerbated by considering a longer horizon.

In this paper, we take a conceptually different approach and address (8) via conditional generative modeling. The key idea is to learn a distribution over system trajectories from collected data and perform decision-making via conditional sampling from this distribution. For this, a trajectory $\boldsymbol{Z}$ is represented by a random variable

$$\boldsymbol{\tau}(\boldsymbol{Z}) = (\boldsymbol{z}_0, \ldots, \boldsymbol{z}_T), \tag{9}$$

where we consider two different formulations: $\boldsymbol{z} = (\boldsymbol{s}_t, \boldsymbol{a}_t)$ for learning state-action [15] and $\boldsymbol{z}_t = \boldsymbol{s}_t$ for learning state-only trajectories [17], [21]. We label these as w/o ID and w/ ID, respectively, throughout this paper, since the second formulation requires learning an additional inverse dynamics (ID) control law $\boldsymbol{a}_t = \boldsymbol{h}_{\boldsymbol{\phi}}(\boldsymbol{s}_t, \boldsymbol{s}_{t+1})$. The ID controller should ideally satisfy the condition

$$\boldsymbol{f}(\boldsymbol{s}_t, \boldsymbol{h}_{\boldsymbol{\phi}}(\boldsymbol{s}_t, \boldsymbol{s}_{t+1})) = \boldsymbol{s}_{t+1}, \qquad \forall \boldsymbol{s}_t, \boldsymbol{s}_{t+1} \in \mathcal{S}, \tag{10}$$

and the parameters $\boldsymbol{\phi}$ can be learned via standard supervised learning techniques [35].

In the following, we mostly write $\boldsymbol{\tau}$ instead of $\boldsymbol{\tau}(\boldsymbol{Z})$ for brevity. We introduce two binary variables corresponding to our main objectives: $\mathcal{O}$ indicates whether $\boldsymbol{\tau}$ is optimal with respect to $R_{\boldsymbol{g}}(\boldsymbol{\tau})$ ($\mathcal{O} = 1$) and $\mathcal{H}$ indicates whether $\boldsymbol{\tau}$ satisfies the constraints (6) ($\mathcal{H} = 1$). Then, our objective can be formulated as learning the conditional trajectory distribution

$$\begin{aligned} p(\boldsymbol{\tau}|\mathcal{O}, \mathcal{H}) &= \frac{p(\boldsymbol{\tau}|\mathcal{O}) p(\mathcal{H}|\boldsymbol{\tau})}{p(\mathcal{H}|\mathcal{O})} \\ &\propto p(\boldsymbol{\tau}|\mathcal{O}) p(\mathcal{H}|\boldsymbol{\tau}). \end{aligned} \tag{11}$$

Here, we have used that $p(\mathcal{H}|\boldsymbol{\tau}, \mathcal{O}) = p(\mathcal{H}|\boldsymbol{\tau})$ as the safety of $\boldsymbol{\tau}$ with respect to (6) is independent of whether $\boldsymbol{\tau}$ is optimal. If (11) is known, then safe goal-conditioned decision-making can simply be achieved by sampling from $p(\boldsymbol{\tau}|\mathcal{O} = 1, \mathcal{H} = 1)$. In the following, we present a computationally tractable way to approximate (11).

A. Sampling Feasible Trajectories

Trajectories (9) of the system (5) follow the distribution

$$q(\boldsymbol{\tau}) = \begin{cases} q(\boldsymbol{A}) \prod_{t=1}^{T} \delta\left(\boldsymbol{s}_t - \hat{\boldsymbol{s}}_t\right), & \text{if } \boldsymbol{z}_t = (\boldsymbol{s}_t, \boldsymbol{a}_t), \\ \prod_{t=1}^{T} \delta\left(\boldsymbol{s}_t - \hat{\boldsymbol{s}}_t\right), & \text{if } \boldsymbol{z}_t = \boldsymbol{s}_t, \end{cases} \tag{12}$$

where $\boldsymbol{A} = (\boldsymbol{a}_0, \ldots, \boldsymbol{a}_T) \sim q(\cdot)$ is an action sequence, $\delta(\cdot)$ is the dirac delta function and $\hat{\boldsymbol{s}}_t = \boldsymbol{f}(\boldsymbol{s}_{t-1}, \boldsymbol{a}_{t-1})$. This unconditional (prior) distribution depends on a prior action distribution $q(\boldsymbol{a})$, i.e., on the behavior policy used for data collection.

To approximate (12), we train an unconditional diffusion model $\boldsymbol{\epsilon}_{\boldsymbol{\theta}}$, as explained in Section III. For this, we collect a dataset $\mathcal{D}$ of dynamically feasible trajectories by applying a sequence of actions $\boldsymbol{A} = (\boldsymbol{a}_0, \ldots, \boldsymbol{a}_T) \sim q(\cdot)$ to (5). In contrast to imitation learning [36], we do not aim to mimic a behavior or expert policy but aim to learn the *best* policy. Therefore, we do not bias the trajectory prior towards a certain control behavior by choosing $q(\boldsymbol{a})$ to be the uniform distribution over $\mathcal{A}$, i.e., trajectories are generated by applying a random sequence of actions. We can sample from the learned trajectory distribution $p_{\boldsymbol{\theta}}(\boldsymbol{\tau})$ via the backward diffusion process

$$p_{\boldsymbol{\theta}}\left(\boldsymbol{\tau}^{k-1}|\boldsymbol{\tau}^k, k\right) = \mathcal{N}\left(\boldsymbol{\mu}_{\boldsymbol{\theta}}\left(\boldsymbol{\tau}^k, k\right), \boldsymbol{\Sigma}_k\right), \tag{13}$$

$k = K, \ldots, 1$, where $\boldsymbol{\tau}^K \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$. The trajectories generated in this way obey the system dynamics (5), but in general, they are neither optimal with respect to $R_{\boldsymbol{g}}(\boldsymbol{\tau})$ nor do they satisfy the constraints (6). We discuss optimality next and later address constraint satisfaction.

B. Incorporating Optimality

We not only wish to obtain dynamically feasible trajectories but also to maximize the reward $R_{\boldsymbol{g}}(\boldsymbol{\tau})$ to achieve the goal. In the generative modeling framework, this can be

formulated as learning the conditional distribution $p(\boldsymbol{\tau}|\mathcal{O})$, where $\mathcal{O}$ denotes the optimality of $\boldsymbol{\tau}$ with respect to $R_{\boldsymbol{g}}(\boldsymbol{\tau})$. From Bayes rule, we have

$$p(\boldsymbol{\tau}|\mathcal{O}) \propto p(\boldsymbol{\tau})p(\mathcal{O}|\boldsymbol{\tau}). \tag{14}$$

Consequently, to sample from $p(\boldsymbol{\tau}|\mathcal{O})$ using a diffusion model, we can iteratively sample from the conditional backward diffusion process

$$p_{\boldsymbol{\theta}}\left(\boldsymbol{\tau}^{k-1}|\boldsymbol{\tau}^k, k, \mathcal{O}\right) \propto p_{\boldsymbol{\theta}}\left(\boldsymbol{\tau}^{k-1}|\boldsymbol{\tau}^k, k\right) p\left(\mathcal{O}|\boldsymbol{\tau}^{k-1}\right), \tag{15}$$

where $p_{\boldsymbol{\theta}}\left(\boldsymbol{\tau}^{k-1}|\boldsymbol{\tau}^k, k\right)$ is given by (13). Sampling from this distribution exactly is intractable. However, if $p\left(\mathcal{O}|\boldsymbol{\tau}\right)$ is sufficiently smooth, the log-likelihood at $\boldsymbol{\tau} = \boldsymbol{\tau}^{k-1}$ can be represented with a first-order Taylor expansion around $\boldsymbol{\mu}_{\boldsymbol{\theta},k} = \boldsymbol{\mu}_{\boldsymbol{\theta}}\left(\boldsymbol{\tau}^k, k\right)$ as

$$\log p\left(\mathcal{O}|\boldsymbol{\tau}^{k-1}\right) \approx \log p\big(\mathcal{O}|\boldsymbol{\mu}_{\boldsymbol{\theta},k}\big) + \left(\boldsymbol{\tau}^{k-1} - \boldsymbol{\mu}_{\boldsymbol{\theta},k}\right)\boldsymbol{v},$$

where $\boldsymbol{v} = \nabla_{\boldsymbol{\tau}} \log p\left(\mathcal{O}|\boldsymbol{\tau}\right)|_{\boldsymbol{\tau}=\boldsymbol{\mu}_{\boldsymbol{\theta},k}}$. We can then approximate the conditional probability (15) by

$$p\left(\boldsymbol{\tau}^{k-1}|\boldsymbol{\tau}^k, k, \mathcal{O}\right) \approx \mathcal{N}\big(\boldsymbol{\mu}_{\boldsymbol{\theta},k} + \boldsymbol{\Sigma}_k\boldsymbol{v}, \boldsymbol{\Sigma}_k\big), \tag{16}$$

as shown in [19]. To make the gradient computation tractable, we can define the optimality indicator by $p(\mathcal{O}|\boldsymbol{\tau}) = \exp(R_{\boldsymbol{g}}(\boldsymbol{\tau}))$, which implies $\boldsymbol{v} = \nabla_{\boldsymbol{\tau}} R_{\boldsymbol{g}}(\boldsymbol{\tau})|_{\boldsymbol{\tau}=\boldsymbol{\mu}_{\boldsymbol{\theta},k}}$. If the cumulative $R_{\boldsymbol{g}}(\boldsymbol{\tau})$ is differentiable, the analytical gradient can be directly plugged into (16), as in [20]. We do not make this assumption and instead train a value diffusion model $V_{\boldsymbol{\psi}}(\boldsymbol{\tau}, \boldsymbol{g})$ parameterized by $\boldsymbol{\psi}$ to predict the cumulative reward $R_{\boldsymbol{g}}(\boldsymbol{\tau})$. This is done by minimizing the loss

$$\mathcal{L}(\boldsymbol{\psi}) = \mathbb{E}_{\boldsymbol{\tau}\sim\mathcal{D}}\left[|V_{\boldsymbol{\psi}}(\boldsymbol{\tau}, \boldsymbol{g}) - R_{\boldsymbol{g}}(\boldsymbol{\tau})|_2\right]. \tag{17}$$

We replace the log-likelihood gradient by $\boldsymbol{v} = \nabla_{\boldsymbol{\tau}} V_{\boldsymbol{\psi}}(\boldsymbol{\tau}, \boldsymbol{g})|_{\boldsymbol{\tau}=\boldsymbol{\mu}_{\boldsymbol{\theta},k}}$ and scale the gradient by a scalar $s > 0$ to adjust the influence of the optimality conditioning. This results in the modified conditional backward diffusion process

$$p_{\boldsymbol{\theta}}\left(\boldsymbol{\tau}^{k-1}|\boldsymbol{\tau}^k, k, \mathcal{O}\right) \approx \mathcal{N}\left(\boldsymbol{\mu}_{\boldsymbol{\theta}}\left(\boldsymbol{\tau}^k, k\right) + s\boldsymbol{\Sigma}_k\boldsymbol{v}, \boldsymbol{\Sigma}_k\right), \tag{18}$$

from which we can iteratively sample to generate optimal trajectories $\boldsymbol{\tau}^0 \sim p(\cdot|\mathcal{O})$.

C. Incorporating Safety

We now address the satisfaction of the test-time constraints (6). Analogously to (14), generating safe trajectories can be formulated as sampling from

$$p(\boldsymbol{\tau}|\mathcal{H}) \propto p(\boldsymbol{\tau})p(\mathcal{H}|\boldsymbol{\tau}). \tag{19}$$

In principle, we could follow a similar approach to Section V-B, i.e., learn a safety classifier and add its log-likelihood gradient as a second guidance term into the backward diffusion process (18). This method, however, is unsuitable for several reasons. First, it requires to trade-off safety and optimality via different scaling variables [37]. Second, it cannot generate trajectories satisfying constraints that are unknown a priori and potentially time-varying. We propose to achieve this flexible type of constraint satisfaction by modifying the sampling from $p_{\boldsymbol{\theta}}(\boldsymbol{\tau}|\mathcal{O})$ such that we approximately sample from $p_{\boldsymbol{\theta}}(\boldsymbol{\tau}|\mathcal{O}, \mathcal{H})$ instead.

We assume that at each timestep $t \in \{0, \dots, T\}$ along the horizon, there are $M_t \in \mathbb{N}_0$ unsafe regions $\bar{\mathcal{S}}_{t,1}, \dots, \bar{\mathcal{S}}_{t,M_t}$ in the state space and $N_t \in \mathbb{N}_0$ unsafe regions $\bar{\mathcal{A}}_{t,1}, \dots, \bar{\mathcal{A}}_{t,N_t}$ in the action space. Note that M_t and N_t can also be zero, in which case the safe sets correspond to the whole state and action space, respectively. We assume the unsafe regions to be disjoint, i.e., $\bar{\mathcal{S}}_{t,i} \cap \bar{\mathcal{S}}_{t,j} = \emptyset$ and $\bar{\mathcal{A}}_{t,i} \cap \bar{\mathcal{A}}_{t,j} = \emptyset$, $\forall t, i \neq j$. The constraint sets in (6) are given by

$$\mathcal{S}_t = \mathcal{S} \setminus \bigcup_{i=0}^{M_t} \bar{\mathcal{S}}_{t,i}, \qquad \mathcal{A}_t = \mathcal{A} \setminus \bigcup_{j=0}^{N_t} \bar{\mathcal{A}}_{t,j}, \tag{20}$$

but we do not need to represent them explicitly with our method. It follows from (20) and the definition of the safety indicator $\mathcal{H}$ that

$$p(\mathcal{H}|\boldsymbol{\tau}) = \begin{cases} 1, & \text{if } \boldsymbol{s}_t \notin \bar{\mathcal{S}}_{t,i} \text{ and } \boldsymbol{a}_t \notin \bar{\mathcal{A}}_{t,j},\ \forall t, i, j \\ 0, & \text{else.} \end{cases} \tag{21}$$

Consequently, we can enforce $\mathcal{H} = 1$ by ensuring that at each planning timestep t, the state $\boldsymbol{s}_t$ (and action $\boldsymbol{a}_t$ for the w/o ID case) are not contained in any of the unsafe regions.

To integrate this into our generative modeling framework, we use the unique iterative nature of diffusion models. After each backward diffusion step (18), we project the denoised trajectory $\boldsymbol{\tau}^{k-1}$ into the safe set as follows: For each unsafe region $\bar{\mathcal{S}}_{t,i}$ in the state space at timestep t, we check whether the t-th state $\boldsymbol{s}_t$ in the trajectory is contained in $\bar{\mathcal{S}}_{t,i}$. If $\boldsymbol{s}_t \in \bar{\mathcal{S}}_{t,i}$, we modify $\boldsymbol{s}_t$ by projecting it out of $\bar{\mathcal{S}}_{t,i}$. The same is done for the t-th action if $\boldsymbol{z}_t = (\boldsymbol{s}_t, \boldsymbol{a}_t)$. This procedure is described in detail in Algorithm 1 for the case $\boldsymbol{z}_t = (\boldsymbol{s}_t, \boldsymbol{a}_t)$ (w/o ID). Due to the assumption that the unsafe regions are disjoint, Algorithm 1 is guaranteed to yield a trajectory satisfying the constraints (6).

Algorithm 1: Algorithm for projecting a state-action trajectory into the safe set (20).

Input: Trajectory $\boldsymbol{\tau}^{k-1} = (\boldsymbol{s}_0, \boldsymbol{a}_0, \dots, \boldsymbol{s}_T, \boldsymbol{a}_T)$, unsafe regions $\bar{\mathcal{S}}_{t,i}, \bar{\mathcal{A}}_{t,j}$.
for $t = 0, \dots, T$ **do**
  $\hat{\boldsymbol{s}}_t = \boldsymbol{s}_t$, $\hat{\boldsymbol{a}}_t = \boldsymbol{a}_t$
  for $i = 0, \dots, M_t$ **do**
    if $\hat{\boldsymbol{s}}_t \in \bar{\mathcal{S}}_{t,i}$ **then**
      $\hat{\boldsymbol{s}}_t \leftarrow \mathrm{proj}_{\mathcal{S}\setminus\bar{\mathcal{S}}_{t,i}}(\hat{\boldsymbol{s}}_t)$
    end
  end
  for $j = 0, \dots, N_t$ **do**
    if $\hat{\boldsymbol{a}}_t \in \bar{\mathcal{A}}_{t,j}$ **then**
      $\hat{\boldsymbol{a}}_t \leftarrow \mathrm{proj}_{\mathcal{A}\setminus\bar{\mathcal{A}}_{t,j}}(\hat{\boldsymbol{a}}_t)$
    end
  end
end
Output: Constraint-satisfying safe trajectory $\boldsymbol{\tau}^{k-1} \leftarrow \left(\hat{\boldsymbol{s}}_0, \hat{\boldsymbol{a}}_0, \dots, \hat{\boldsymbol{s}}_T, \hat{\boldsymbol{a}}_T\right)$.

In principle, Algorithm 1 can be applied regardless of the shape of the unsafe regions as long as the projections in lines 6 and 11 can be computed. While this may be computationally demanding for complex set representations, projections out of simple geometries, such as spheres or hyperrectangles, can be calculated in a computationally efficient way. In practice, we can project out of the unsafe regions by slightly enlarging them and projecting onto the boundary of the resulting set.

We have modified the sampling process to alternate between guiding toward optimal trajectories and enforcing safety, allowing us to approximately sample from $p(\boldsymbol{\tau}|\mathcal{O},\mathcal{H})$. Note that this approach is not directly applicable to other types of generative models, such as VAEs or GANs, that generate samples in one step instead of iteratively. For those methods, projecting a trajectory sample out of the unsafe set can easily destroy the dynamic feasibility of the trajectory.

Algorithm 2: Our proposed algorithm for safe goal-conditioned diffusion-based receding horizon control with warm-starting.

1 **Input:** Trajectory diffusion model $\boldsymbol{\epsilon_\theta}$, value model $V_{\boldsymbol{\psi}}$, goal $\boldsymbol{g}$, inverse dynamics $\boldsymbol{h_\phi}$ (for w/ ID).
2 Set $t=0$.
3 **while** *goal* $\boldsymbol{g}$ *not reached* **do**
4 **if** $t=0$ **then**
5 Sample from noise $\boldsymbol{\tau}_{1:B}^{K} \sim \mathcal{N}(\mathbf{0},\boldsymbol{I})$.
6 **else**
7 $\boldsymbol{\tau}_t^{0,1:B} \leftarrow \text{Shift\_and\_Pad}\left(\boldsymbol{\tau}_{t-1}^{0,1:B}\right)$.
8 Add noise: $\boldsymbol{\tau}_t^{\tilde{K},1:B} \sim \mathcal{N}\left(\sqrt{\alpha_k}\boldsymbol{\tau}_t^{0,1:B}, (1-\alpha_k)\boldsymbol{I}\right)$.
9 **end**
10 **for** $k=\tilde{K},\dots,1$ **do**
11 Compute gradient $\boldsymbol{v} = \nabla_{\boldsymbol{\tau}} V_{\boldsymbol{\psi}}(\boldsymbol{\tau},\boldsymbol{g})|_{\boldsymbol{\tau}=\boldsymbol{\tau}_t^{k,1:B}}$.
12 Denoising step: Sample $\boldsymbol{\tau}_t^{k-1,1:B} \sim \mathcal{N}\left(\boldsymbol{\mu_\theta}\left(\boldsymbol{\tau}_t^{k,1:B},k\right) + s\boldsymbol{\Sigma}_{k-1}\boldsymbol{v}, \boldsymbol{\Sigma}_{k-1}\right)$
13 Set first state in $\boldsymbol{\tau}_t^{k-1,1:B}$ to the current state.
14 Project $\boldsymbol{\tau}_t^{k-1,1:B}$ into the constraint set via Algorithm 1.
15 **end**
16 Select best trajectory $\boldsymbol{\tau}^* = \arg\max_{\boldsymbol{\tau}_t^{0,i}} V_{\boldsymbol{\psi}}(\boldsymbol{\tau}_t^{0,i})$.
17 **if** *w/ ID* **then**
18 Apply action $\boldsymbol{a}_0^* = \boldsymbol{h_\phi}(\boldsymbol{s}_0^*, \boldsymbol{s}_1^*)$.
19 **else**
20 Apply first action $\boldsymbol{a}_0^*$ in $\boldsymbol{\tau}^*$.
21 **end**
22 $t \leftarrow t+1$.
23 **end**

D. Receding-Horizon Solution and Warm-Starting

To close the control loop, we adopt a receding horizon control approach [38]. If $(\boldsymbol{z}_t = (\boldsymbol{s}_t, \boldsymbol{a}_t))$, the first action $\boldsymbol{a}_0$ in $\boldsymbol{\tau}$ is applied to the system, and if $(\boldsymbol{z}_t = \boldsymbol{s}_t)$, then $\boldsymbol{a}_0$ is calculated from $\boldsymbol{s}_0$ and $\boldsymbol{s}_1$ via the inverse dynamics controller (10). For consistency, the first state $\boldsymbol{s}_0$ in the sampled trajectory should correspond to the current state $\boldsymbol{s}$. We enforce this condition by modifying the first state accordingly after each backward diffusion step.

Our online algorithm is summarized in Algorithm 2. The generative modeling framework allows sampling a whole batch of B trajectories without significantly increasing computation time. We choose the trajectory with the largest predicted reward $V_{\boldsymbol{\psi}}(\boldsymbol{\tau},\boldsymbol{g})$; see line 16. Alternative selection criteria, such as the distance to the unsafe regions, can also be adopted.

So far, we have considered the standard formulation of diffusion models that starts from Gaussian noise $\boldsymbol{\tau}^K \sim \mathcal{N}(\mathbf{0},\boldsymbol{I})$ to generate a new sample. The receding horizon control execution enables us to deviate from this formulation. We adopt a warm-starting strategy and apply $\tilde{K} < K$ forward and backward diffusion steps to the previous batch of trajectories shifted by one timestep. In this way, some information about the previous plan is kept, leading to more consistency between trajectories sampled at consecutive timesteps. Moreover, the warm-starting approach reduces the online computation time.

VI. Experiments

Our evaluation primarily aims to answer the following questions:

- **Q1**: How do the hyperparameters of Algorithm 2 affect its performance?
- **Q2**: Does the proposed projection method improve constraint satisfaction, and how does it affect performance?
- **Q3**: How does sampling state-action trajectories (w/o ID) compare to sampling state trajectories and using an inverse dynamics model for control (w/ ID)?

We subsequently describe our simulation environment, model architecture and training process before presenting and discussing our results.

A. Simulation Environment

We consider two robotic systems: A mobile robot with acceleration commands and a quadrotor with thrust commands. Both simulation environments are implemented using OpenAI Gym [39][1].

Mobile robot: The state is defined as $\boldsymbol{s} = \left(x, \dot{x}, y, \dot{y}\right)$, where (x,y) is the robot's position in the horizontal plane, and we command the accelerations, i.e., $\boldsymbol{a} = \left(\ddot{x}, \ddot{y}\right)$. We use a simulation timestep of $0.1\,\text{s}$.

2D quadrotor: As a more challenging system, we consider a quadrotor flying in the vertical plane with position (x,z) and pitch angle θ. The system is simulated by discretizing the continuous-time dynamics [40]

$$\begin{aligned} m\ddot{x} &= -(T_1+T_2)\sin(\theta) \\ m\ddot{z} &= (T_1+T_2)\cos(\theta) - mg \\ I_{yy}\ddot{\theta} &= (T_1-T_2)d, \end{aligned} \tag{22}$$

[1]Code is available at github.com/ralfroemer99.

State/action component	Bounds
Position	$\pm 5\,\mathrm{m}$
Linear velocity	$\pm 5\,\frac{\mathrm{m}}{\mathrm{s}}$
Angular velocity (quadrotor)	$\pm 5\,\frac{\mathrm{rad}}{\mathrm{s}}$
Acceleration (mobile robot)	$\pm 5\,\frac{\mathrm{m}}{\mathrm{s}^2}$
Total thrust (quadrotor)	$[0.75, 1.25]\,mg$

TABLE I: State and action space for the two simulated systems.

Hyperparameter	Value
Optimizer	Adam [41]
Batch size	32
Learning rate	2×10^{-5}
Training steps	10^6
Epochs	100
Diffusion steps K	20

TABLE II: Hyperparameters for training the diffusion models.

where $m = 0.1\,\mathrm{kg}$ is the mass, (T_1, T_2) are the motor thrusts, $g = 9.81\,\frac{\mathrm{m}}{\mathrm{s}^2}$, $d = 0.1\,\mathrm{m}$ is the length of the effective moment arm of the propellers, and $I_{yy} = \frac{1}{12}md^2$ is the inertia about the y-axis. We define the state as $\boldsymbol{s} = \big(x,\, \dot{x},\, z,\, \dot{z},\, \sin\theta,\, \cos\theta,\, \dot{\theta}\big)$ and the action as $\boldsymbol{a} = \big(T_1,\, T_2\big)$. As the quadrotor is challenging to stabilize, we use a smaller simulation timestep of $0.05\,\mathrm{s}$.

Both systems' state and action spaces are provided in Table I. In addition, we constrain the absolute thrust difference in (22) to $|T_1 - T_2| \leq 0.01\,mg$, which implies $\big|\ddot{\theta}\big| \leq 11.8\,\frac{\mathrm{rad}}{\mathrm{s}^2}$. The state and action values are normalized to $[-1, 1]$. For both systems, the objective is to reach a circle with radius $0.2\,\mathrm{m}$ centered at a randomly selected goal position $\boldsymbol{g}$. We use a simple quadratic reward $r(\boldsymbol{s}_t, \boldsymbol{a}_t, \boldsymbol{g}) = -d(\boldsymbol{s}_t, \boldsymbol{g})^2$, where $d(\cdot, \cdot)$ calculates the Euclidean distance between the current position and goal.

We consider rectangular and circular obstacles as the test-time state constraints in (6). The obstacles are either static or moving with a constant velocity, representing time-invariant or time-varying constraints.

B. Data Generation and Training

The offline dataset $\mathcal{D}$ is generated by applying randomly uniformly selected actions from $\mathcal{A}$. Whenever the distance to the goal position is less than $0.2\,\mathrm{m}$, or the state leaves the set $\mathcal{S}$, the simulation is reset with a random initial state and goal, and the trajectory is added to the dataset if its length is at least T. In this way, we generate 10^5 trajectories, of which we use 10% for testing and the rest for training the diffusion models $\boldsymbol{\epsilon_\theta}$ and V_ψ and the inverse dynamics controller. Note that our training data does not contain any constraints aside from $\boldsymbol{s}_t \in \mathcal{S}$ and $\boldsymbol{a}_t \in \mathcal{A}$. We set the horizon length to $T = 20$ for the mobile robot and $T = 40$ for the quadrotor, corresponding to a prediction horizon of $2\,\mathrm{s}$.

The trajectory diffusion model $\boldsymbol{\epsilon_\theta}$ is implemented as a temporal U-Net [15], which consists of six repeated residual blocks. Each block contains two temporal convolutions with group normalization and Mish nonlinearity, along with separate 2-layered MLPs producing timestep and condition embeddings that are concatenated and added to the activations of the first temporal convolution in each block. The same architecture is chosen for the value model V_ψ but with an additional fully connected output layer. The training hyperparameters for both diffusion models are listed in Table II. The inverse dynamics function $\boldsymbol{h_\phi}$ is implemented as a multilayer perception (MLP) with two hidden layers, 512 hidden units per layer and ReLU activations, and we train it with the same data as the diffusion models. In all experiments, we evaluate using the models performing best on the test set.

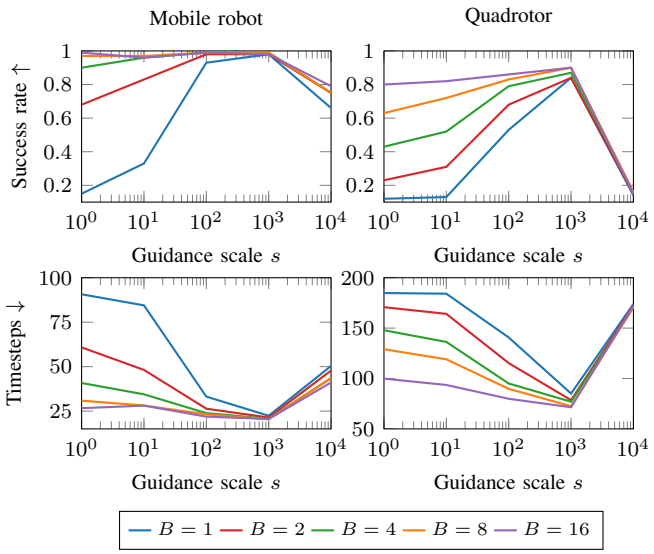

Fig. 1: Impact of the guidance scale s and the batch size B in Algorithm 2 on the success rate and the number of timesteps to reach the goal for sampling state trajectories and using an inverse dynamics controller (w/ ID).

C. Results

We use two evaluation metrics: The success rate and the average number of timesteps until reaching the goal, both calculated from 100 different environment initializations.

Q1: We evaluate the impact of two key hyperparameters of Algorithm 2: The guidance scale s in (18) and the batch size B. For each pair (s, B), we simulate the systems 100 times with random goal positions and without obstacles. Unlike for data generation, we initialize both systems with zero velocity and the quadrotor in a hovering state for testing. The results are shown in Fig. 1 for the case of sampling state trajectories and using an inverse dynamics controller (w/ ID). Sampling a larger batch consistently improves the performance since generating more trajectories increases the likelihood of obtaining one with high predicted reward $V_\psi(\boldsymbol{\tau}, \boldsymbol{g})$. The batch size has a stronger impact on controlling the quadrotor system, likely due to its higher state dimension and inherent instability. We also observe that for small batch sizes, the guidance scale has a large impact on performance, but this impact strongly decreases for larger batches. The ablation study for sampling state-action trajectories yields qualitatively similar results and is provided in the Appendix. In the following, we use $B = 16$ and $s = 100$, which we found to perform better than $s = 1000$ in scenarios with test-time constraints where we alternate guidance and projection.

Q2: We evaluate the impact of our proposed projec-

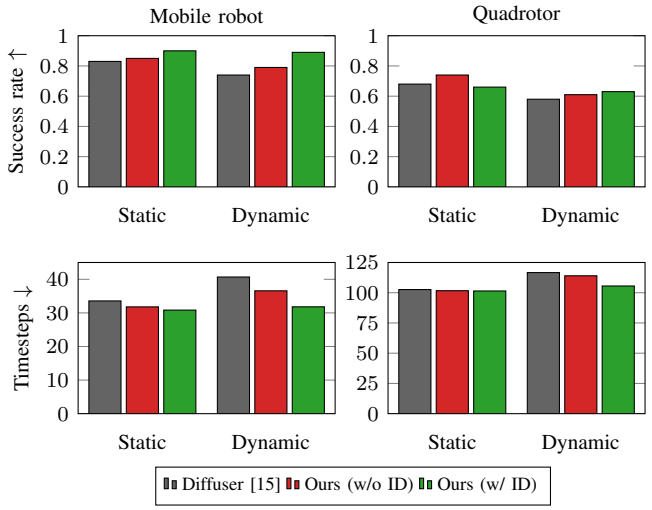

Fig. 2: Our proposed projection method reduces the number of constraint violations, resulting in a higher success rate of reaching the goal.

	Mobile Robot		Quadrotor	
	Success rate ↑	Timesteps ↓	Success rate ↑	Timesteps ↓
w/ ID	**0.99 / 0.89**	20.4 / **31.8**	**0.90 / 0.63**	**71.5 / 105.6**
w/o ID	**0.99** / 0.79	**18.4** / 36.6	0.87 / 0.61	**71.0** / 114.0

TABLE III: Comparison of calculating the actions from a state trajectory with a separate inverse dynamics controller (w ID) against directly sampling state-action trajectories. We report the values for the obstacle-free environment and the dynamic environment.

tion scheme on avoiding constraint violations in closed-loop operation. To this end, we consider two scenarios: A static environment with ten randomly placed obstacles and a dynamic environment with an additional four dynamic obstacles. We use the popular Diffuser [15] as a baseline. The results are provided in Fig. 2. Our method can avoid most of the collisions with obstacles, demonstrating its ability to improve constraint satisfaction. Also, the performance is not negatively affected by our projection method, demonstrated by a decrease in the number of timesteps across most of our experiments. Fig. 3 shows a simulation of the mobile robot with (top) and without (bottom) our projection method. In each timestep, the agent executes the first action corresponding to the red trajectory. With the projection method, trajectories going either way around the obstacle are sampled initially. When approaching the obstacle, all sampled trajectories are eventually projected to the same side of the obstacle. With the projection method, the agent satisfies the collision constraint and reaches the goal.

Q3: Finally, we compare directly sampling state-action trajectories (w/o ID) to sampling state trajectories and using an ID controller (w/ ID) in Table III. The second approach performs better in most cases. This is likely due to the fact that CNNs, such as the employed temporal U-Net, tend to struggle at capturing rapidly changing action sequences like the acceleration and thrust commands considered in this work. Using a transformer backbone can be beneficial for directly generating action sequences, however at the cost of additional tuning [22].

VII. Conclusion

This paper addresses offline RL with constraints by learning conditional trajectory distributions using diffusion models. In particular, we consider the satisfaction of unseen test-time constraints while ensuring dynamic feasibility and optimality of the generated trajectories. To achieve this, we propose to alternate the backward diffusion process with a projection into the constraint set, exploiting the iterative nature of diffusion models. We demonstrate in simulation that our proposed scheme can significantly reduce the number of constraint violations, thereby improving the goal-reaching performance. Possible avenues for future work include employing a different model architecture than U-Net and conducting hardware experiments, potentially using recently proposed fast sampling methods for diffusion models.

References

[1] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[2] N. Rudin, D. Hoeller, P. Reist, and M. Hutter, "Learning to walk in minutes using massively parallel deep reinforcement learning," in *Conference on Robot Learning*. PMLR, 2022, pp. 91–100.

[3] E. Kaufmann, L. Bauersfeld, A. Loquercio, M. Müller, V. Koltun, and D. Scaramuzza, "Champion-level drone racing using deep reinforcement learning," *Nature*, vol. 620, no. 7976, pp. 982–987, 2023.

[4] L. Brunke, M. Greeff, A. W. Hall, Z. Yuan, S. Zhou, J. Panerati, and A. P. Schoellig, "Safe learning in robotics: From learning-based control to safe reinforcement learning," *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 5, pp. 411–444, 2022.

[5] W. Zhao, T. He, R. Chen, T. Wei, and C. Liu, "State-wise safe reinforcement learning: a survey," in *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence*, 2023, pp. 6814–6822.

[6] A. Nagabandi, G. Kahn, R. S. Fearing, and S. Levine, "Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2018, pp. 7559–7566.

[7] S. Levine, A. Kumar, G. Tucker, and J. Fu, "Offline reinforcement learning: Tutorial, review, and perspectives on open problems," *arXiv preprint arXiv:2005.01643*, 2020.

[8] R. Rombach, A. Blattmann, D. Lorenz, P. Esser, and B. Ommer, "High-resolution image synthesis with latent diffusion models," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2022, pp. 10 684–10 695.

[9] A. Ramesh, P. Dhariwal, A. Nichol, C. Chu, and M. Chen, "Hierarchical text-conditional image generation with clip latents," *arXiv preprint arXiv:2204.06125*, 2022.

[10] J. Ho, W. Chan, C. Saharia, J. Whang, R. Gao, A. Gritsenko, D. P. Kingma, B. Poole, M. Norouzi, D. J. Fleet *et al.*, "Imagen video: High definition video generation with diffusion models," *arXiv preprint arXiv:2210.02303*, 2022.

[11] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," *arXiv preprint arXiv:1312.6114*, 2013.

[12] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," *Advances in Neural Information Processing Systems*, vol. 27, 2014.

[13] J. Sohl-Dickstein, E. Weiss, N. Maheswaranathan, and S. Ganguli, "Deep unsupervised learning using nonequilibrium thermodynamics," in *International Conference on Machine Learning*. PMLR, 2015, pp. 2256–2265.

[14] J. Ho, A. Jain, and P. Abbeel, "Denoising diffusion probabilistic models," *Advances in Neural Information Processing Systems*, vol. 33, pp. 6840–6851, 2020.

[15] M. Janner, Y. Du, J. Tenenbaum, and S. Levine, "Planning with diffusion for flexible behavior synthesis," in *Proceedings of the International Conference on Machine Learning*, ser. PMLR, vol. 162, 2022, pp. 9902–9915.

[16] Z. Wang, J. J. Hunt, and M. Zhou, "Diffusion policies as an expressive policy class for offline reinforcement learning," in *The Eleventh International Conference on Learning Representations*, 2023.

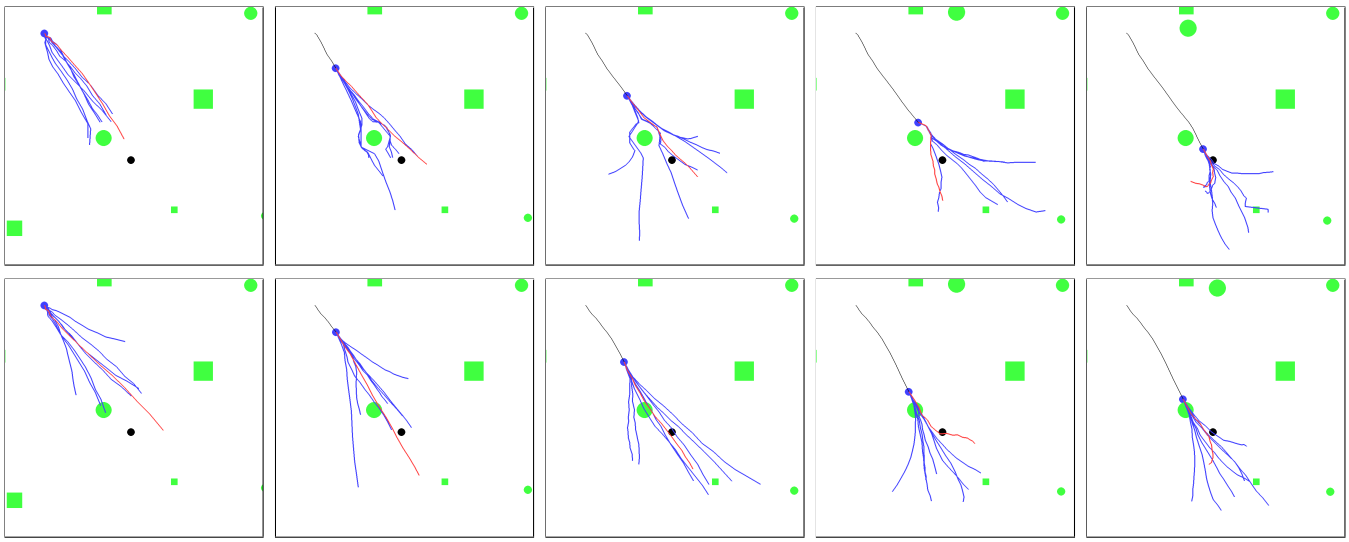

Fig. 3: Illustration of our diffusion-based algorithm for controlling a mobile robot (blue dot) to reach a goal (black dot) in dynamic environments with static and dynamic obstacles (green). The blue and red lines represent the batch of trajectories sampled from the diffusion model at that particular timestep, and the red line indicates the best trajectory $\boldsymbol{\tau}^*$ selected in line 16 of Algorithm 2. With our proposed projection method (top), the denoised trajectories are guaranteed to satisfy the constraints after each backward diffusion step, and thus, the final trajectories are safe in contrast to Diffuser [15] (bottom).

[17] A. Ajay, Y. Du, A. Gupta, J. B. Tenenbaum, T. S. Jaakkola, and P. Agrawal, "Is conditional generative modeling all you need for decision making?" in *International Conference on Learning Representations*, 2023.

[18] W. Xiao, T.-H. Wang, C. Gan, and D. Rus, "Safediffuser: Safe planning with diffusion probabilistic models," *arXiv preprint arXiv:2306.00148*, 2023.

[19] P. Dhariwal and A. Nichol, "Diffusion models beat gans on image synthesis," *Advances in neural information processing systems*, vol. 34, pp. 8780–8794, 2021.

[20] J. Carvalho, A. T. Le, M. Baierl, D. Koert, and J. Peters, "Motion planning diffusion: Learning and planning of robot motions with diffusion models," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2023, pp. 1916–1923.

[21] J. Urain, N. Funk, J. Peters, and G. Chalvatzaki, "Se (3)-diffusionfields: Learning smooth cost functions for joint grasp and motion optimization through diffusion," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2023, pp. 5923–5930.

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

[23] L. Chen, S. Bahl, and D. Pathak, "Playfusion: Skill acquisition via diffusion from language-annotated play," in *Conference on Robot Learning*, 2023, pp. 2012–2029.

[24] I. Kapelyukh, V. Vosylius, and E. Johns, "Dall-e-bot: Introducing web-scale diffusion models to robotics," *IEEE Robotics and Automation Letters*, 2023.

[25] T. Yu, T. Xiao, A. Stone, J. Tompson, A. Brohan, S. Wang, J. Singh, C. Tan, J. Peralta, B. Ichter *et al.*, "Scaling robot learning with semantically imagined experience," *arXiv preprint arXiv:2302.11550*, 2023.

[26] A. Kumar, A. Zhou, G. Tucker, and S. Levine, "Conservative q-learning for offline reinforcement learning," *Advances in Neural Information Processing Systems*, vol. 33, pp. 1179–1191, 2020.

[27] L. Chen, K. Lu, A. Rajeswaran, K. Lee, A. Grover, M. Laskin, P. Abbeel, A. Srinivas, and I. Mordatch, "Decision transformer: Reinforcement learning via sequence modeling," *Advances in Neural Information Processing Systems*, vol. 34, pp. 15 084–15 097, 2021.

[28] M. Janner, Q. Li, and S. Levine, "Offline reinforcement learning as one big sequence modeling problem," *Advances in Neural Information Processing Systems*, vol. 34, pp. 1273–1286, 2021.

[29] K.-H. Lee, O. Nachum, S. Yang, L. Lee, C. D. Freeman, S. Guadarrama, I. Fischer, W. Xu, E. Jang, H. Michalewski, and I. Mordatch, "Multi-game decision transformers," *Advances in Neural Information Processing Systems*, vol. 35, pp. 27 921–27 936, 2022.

[30] J. Hu, Y. Sun, S. Huang, S. Guo, H. Chen, L. Shen, L. Sun, Y. Chang, and D. Tao, "Instructed diffuser with temporal condition guidance for offline reinforcement learning," *arXiv preprint arXiv:2306.04875*, 2023.

[31] J. Ho and T. Salimans, "Classifier-free diffusion guidance," *arXiv preprint arXiv:2207.12598*, 2022.

[32] Y. Song, J. Sohl-Dickstein, D. P. Kingma, A. Kumar, S. Ermon, and B. Poole, "Score-based generative modeling through stochastic differential equations," *arXiv preprint arXiv:2011.13456*, 2020.

[33] C. J. Ostafew, A. P. Schoellig, and T. D. Barfoot, "Robust constrained learning-based nmpc enabling reliable mobile robot path tracking," *The International Journal of Robotics Research*, vol. 35, no. 13, pp. 1547–1563, 2016.

[34] F. Berkenkamp, M. Turchetta, A. Schoellig, and A. Krause, "Safe model-based reinforcement learning with stability guarantees," *Advances in Neural Information Processing Systems*, 2017.

[35] P. Agrawal, A. V. Nair, P. Abbeel, J. Malik, and S. Levine, "Learning to poke by poking: Experiential learning of intuitive physics," *Advances in Neural Information Processing Systems*, vol. 29, 2016.

[36] D. A. Pomerleau, "Efficient training of artificial neural networks for autonomous navigation," *Neural Computation*, vol. 3, no. 1, pp. 88–97, 1991.

[37] N. Botteghi, F. Califano, M. Poel, and C. Brune, "Trajectory generation, control, and safety with denoising diffusion probabilistic models," in *ICML Workshop on New Frontiers in Learning, Control, and Dynamical Systems*, 2023.

[38] J. B. Rawlings, D. Q. Mayne, and M. Diehl, *Model predictive control: theory, computation, and design*. Nob Hill Publishing Madison, 2017, vol. 2.

[39] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," *arXiv preprint arXiv:1606.01540*, 2016.

[40] Z. Yuan *et al.*, "safe-control-gym: A unified benchmark suite for safe learning-based control and reinforcement learning in robotics," *IEEE Robotics and Automation Let.*, vol. 7, no. 4, pp. 11 142–11 149, 2022.

[41] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

APPENDIX

We provide the ablation study for directly sampling actions from the diffusion model (w/o ID) in Fig. 4. The results are very similar to the w/ID formulation; see Fig. 1. The only notable difference is the very strong performance drop when

increasing the scale from 10^3 to 10^4, indicating a higher sensitivity of the algorithm to the hyperparameter choice when directly generating actions.

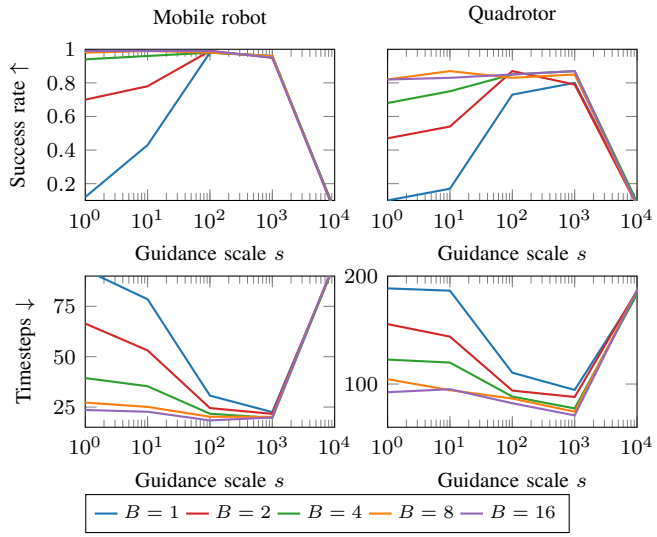

Fig. 4: Impact of the guidance scale s and the batch size B in Algorithm 2 on the success rate and the number of timesteps to reach the goal for the case of sampling state-action trajectories (w/o ID).