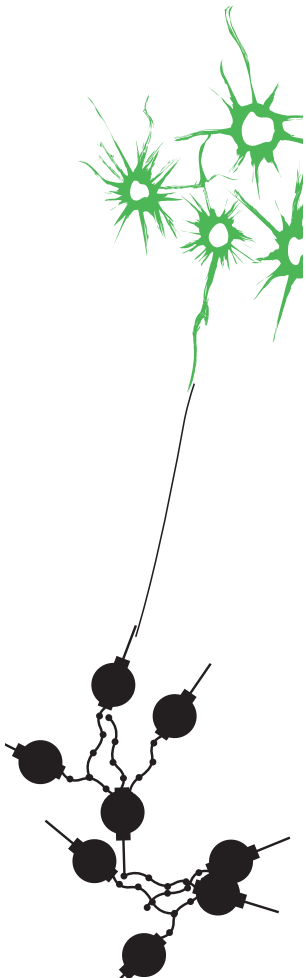


RAM ROBOTICS AND MECHATRONICS

PATH PLANNING WITH DDPMS

M.S. (Michiel) Nikken

MSC ASSIGNMENT

Committee:
prof. dr. ir. S. Stramigioli
dr. ir. F. Califano
dr. F.C. Nex

October, 2024

066RaM2024
Robotics and Mechatronics
EEMCS
University of Twente
P.O. Box 217
7500 AE Enschede
The Netherlands

UNIVERSITY OF TWENTE. | TECHMED CENTRE

UNIVERSITY OF TWENTE. | DIGITAL SOCIETY INSTITUTE

Denoising Diffusion Planner: Learning Complex Paths for Robot Control from Low-Quality Demonstrations

***Abstract*—Denoising Diffusion Probabilistic Models (DDPMs) are powerful generative deep learning models that have been very successful at image generation. In this paper, we investigate how to leverage the generalization and conditional-sampling capabilities of DDPMs to generate complex paths for a robotic end effector. We show that these paths can exhibit obstacle avoidance, even when training the DDPM on synthetically generated low-quality demonstrations. The trained DDPM is deployed in a receding-horizon control scheme to enhance its planning capabilities. The Denoising Diffusion Planner is experimentally validated through various experiments on a Franka Emika Panda robot.**

I. Introduction

Path planning is a fundamental aspect of almost any robotic task. It can be defined as finding a collision-free trajectory between a starting state and a goal state, such that the system can be driven to the desired target by tracking this trajectory using a low-level controller. Designing algorithms to find these trajectories is difficult, because robotic environments are typically high-dimensional, complex, and dynamic.

Traditionally, to address the task of path planning, combinatorial algorithms have been developed. These algorithms are complete, meaning that they will find a solution in a finite amount of time, if a solution exists [1]. However, for complex environments it is often preferable to trade completeness for efficiency. To this end, sampling-based methods [2] have been successfully developed. Examples of traditional sampling-based methods are the probabilistic roadmap method (PRM) [3] and rapidly exploring random trees (RRTs) [4].

More recently, advances in reinforcement learning (RL) [5] have introduced a different approach to planning. RL seeks to find the optimal control strategy that maximizes the expected cumulative future rewards by interacting with an unknown environment. Utilizing deep neural networks as function approximators to solve RL problems is known as deep reinforcement learning (DRL) [5], which has allowed RL to scale to high-dimensional problems [6], [7]. Iteratively executing the actions generated by the policy while updating the current state estimates using observations from the environment makes for a reactive planner [8], [9].

Despite their successful application in different path-planning problems, the previously-mentioned traditional sampling-based methods and DRL methods have several drawbacks. For example, it is not obvious how to plan a complex behavior that meets conditions other than obstacle avoidance using traditional sampling-based planners. On the other hand, DRL methods learn a richer model that may include system dynamics and arbitrary reward functions. However, applying these methods for planning requires autoregressively using their model's one-step predictions, which allows model imperfections to compound over time [10], [11].

Instead, the problem of path planning can be addressed through the paradigm of planning as inference (PAI) [12], [13]. Differently from explicit graph search or autoregressive prediction, the future is modeled as a joint probability distribution over states, actions and rewards. Sampling from this distribution yields a sequence of states and actions that represent a possible trajectory. Retrieving just any possible trajectory is not very useful by itself, but if this trajectory is known to meet certain conditions, like obstacle avoidance or achieving certain rewards, this sample can be used as a plan. In PAI, such plan can be sampled from a conditional probability distribution of possible futures. This process of conditional sampling is key for generating a plan that exhibits desirable behavior.

A particularly recent class of models that fit the PAI view is the Denoising Diffusion Probabilistic Model (DDPM) [14], [15]. DDPMs are generative probabilistic models that have been remarkably successful in image generation [16]. They are trained by iteratively adding random noise to the training data and learning the reverse process that iteratively denoises the data to retrieve the original sample. In this way, one can sample from the data distribution carrying the characteristics of the training data by first sampling noise from a Gaussian distribution and then applying the denoising process.

The generative capabilities of DDPMs can also be used to generate plans. Janner et al. [17] trained DDPMs on an offline collection of state-action sequences, with associated rewards. In [17] it was shown that DDPMs exhibit good generalization and long-horizon planning abilities, even for sparse rewards. Furthermore, they showed that DDPMs can generate novel trajectories by locally stitching together sequences from the training distribution. Moreover, Ajay et al. [18] found that DDPMs can generate plans that meet multiple conditions even though these conditions were never met simultaneously in the training data. These properties of DDPMs make them potentially powerful and versatile planners. Real-world demonstrations of using DDPMs for robot control have shown promising results [19]–[21], which inspires this work to further investigate the novel aspects of using DDPMs to solve robotic planning problems.

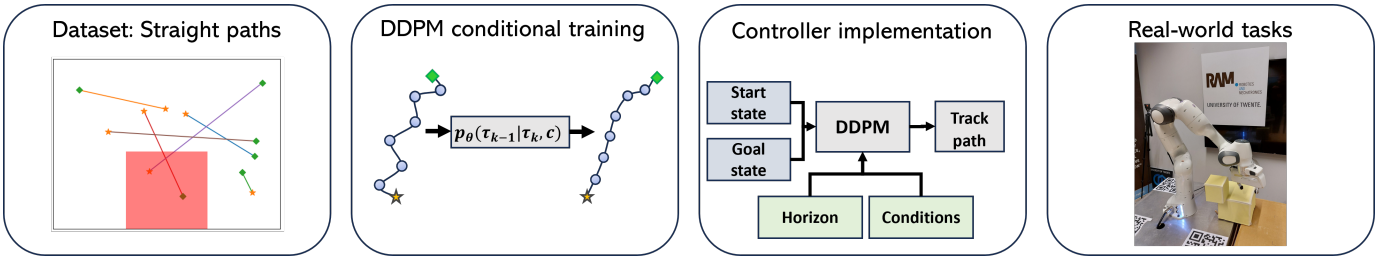


Fig. 1: Overview of the pipeline from dataset to real-world tasks.

Related work can be found in [19], [21], [22]. Carvalho et al. [21] propose a motion planner in the joint space using DDPMs. They use expert examples produced by an optimal motion planning algorithm to train a DDPM. During inference, they use the gradients of a cost function to bias the samples towards regions of low cost at every denoising step. Saha et al. [22] expand on this work by using an ensemble of cost functions to create a planner that can generate collision-free trajectories in a variety of environments. In order to investigate the DDPM's generalization capabilities, we train the model using only synthetically generated low-quality demonstrations and we avoid using a dynamic model for the robot. This is a key difference from [21], [22], which use expert demonstrations to train the DDPM. Chi et al. [19] use DDPMs to directly learn a policy that takes visual observations as its input and produces an action sequence as an output. In their work, they introduce closed-loop planning with DDPMs by implementing a receding-horizon control scheme. Inspired by their work, we include closed-loop planning to enhance the planning capabilities of the DDPMs.

This work investigates how to use a DDPM to create a path planner for an end effector of a robotic arm. Figure 1 shows the four stages of this process, from generating low-quality demonstrations to using the DDPM to control a robot. Furthermore, the ability of the DDPM to generalize and produce useful trajectories through conditional sampling is explored. The main contributions of this work are:

1) Propose a general approach to designing a path planner for an end effector using a DDPM, without using a dynamic robot model and using only low-quality demonstrations for training data.
2) Experimentally validate the proposed path planner using real-world experiments.

II. Background

A. Denoising Diffusion Probabilistic Models

Denoising Diffusion Probabilistic Models (DDPM) [14], [15] are a class of generative deep learning models that can learn data distributions according to a given training dataset in a way that allows for sampling from the learned distribution. DDPMs are characterized by a forward process and a reverse process. In the forward process, Gaussian noise is added to the training data iteratively. The forward process is a discrete Markov chain, i.e., a memoryless process in which the previous state of the chain $\boldsymbol{x}^{k-1}$ is sufficient to determine the current state $\boldsymbol{x}^k$. In general, given a noise-free sample $\boldsymbol{x}^0$ from dataset $\mathcal{D}$, the forward process can be expressed as follows:

$$q\left(\boldsymbol{x}^K|\boldsymbol{x}^0\right) = \prod_{k=1}^{K} q\left(\boldsymbol{x}^k|\boldsymbol{x}^{k-1}\right) , \tag{1}$$

where $k = 0, 1, ..., K$ is the index of the diffusion step and $q\left(\boldsymbol{x}^k|\boldsymbol{x}^{k-1}\right)$ the Markov diffusion kernel. This kernel is a probability distribution describing the probability of $\boldsymbol{x}^k$ for a given $\boldsymbol{x}^{k-1}$. The kernel is chosen such that its repeated application will transform the initial data distribution $q\left(\boldsymbol{x}^0\right)$ into a Gaussian distribution $\mathcal{N}\left(\boldsymbol{0}, \boldsymbol{I}\right)$:

$$q\left(\boldsymbol{x}^k|\boldsymbol{x}^{k-1}\right) = \mathcal{N}\left(\sqrt{1-\beta^k}\boldsymbol{x}^{k-1}, \beta^k \boldsymbol{I}\right) , \tag{2}$$

where $\beta^k \in (0, 1)$ is the variance in the forward process. Assuming that $q\left(\boldsymbol{\tau}^0\right)$ has been normalized to unit variance, scaling the mean of the Gaussian kernel by $\sqrt{1-\beta^k}$ will keep the variance of $q\left(\boldsymbol{\tau}^k|\boldsymbol{\tau}^0\right)$ unity. Hence, $q\left(\boldsymbol{\tau}^K\right)$ will resemble a standard normal distribution for sufficiently large number of diffusion steps K.

The reverse process is again a Markov chain, parameterized by a parameter vector $\boldsymbol{\theta}$, that can be written in as:

$$\begin{aligned} p_{\boldsymbol{\theta}}\left(\boldsymbol{x}^0\right) &= \prod_{k=1}^{K} p_{\boldsymbol{\theta}}\left(\boldsymbol{x}^{k-1}|\boldsymbol{x}^k\right) , \\ p_{\boldsymbol{\theta}}\left(\boldsymbol{x}^{k-1}|\boldsymbol{x}^k\right) &= \mathcal{N}\left(\boldsymbol{\mu}_{\boldsymbol{\theta}}\left(\boldsymbol{x}^k, k\right), \boldsymbol{\Sigma}_{\boldsymbol{\theta}}\left(\boldsymbol{x}^k, k\right)\right) , \end{aligned} \tag{3}$$

where $\boldsymbol{\mu}_{\boldsymbol{\theta}}\left(\boldsymbol{x}^k, k\right)$ is learned from data, using a deep neural network with parameters indicated by $\boldsymbol{\theta}$, and the covariance is fixed to a cosine schedule, i.e., $\boldsymbol{\Sigma}_\theta\left(\boldsymbol{x}^k, k\right) = \sigma^k \boldsymbol{I}$, as proposed by Nichol and Dhariwal [23].

Ho et al. [15] found that the Gaussian noise kernel allows for a closed-form description of the forward process:

$$q\left(\boldsymbol{x}^k|\boldsymbol{x}^0\right) = \mathcal{N}\left(\sqrt{\bar{\alpha}^k}\boldsymbol{x}^0, \left(1-\bar{\alpha}^k\right)\boldsymbol{I}\right) , \tag{4}$$

where $\bar{\alpha}^k = \prod_{s=1}^{k} 1 - \beta^s$. The reverse process can also be written in closed-form, expressing the probability of $\boldsymbol{x}^{k-1}$

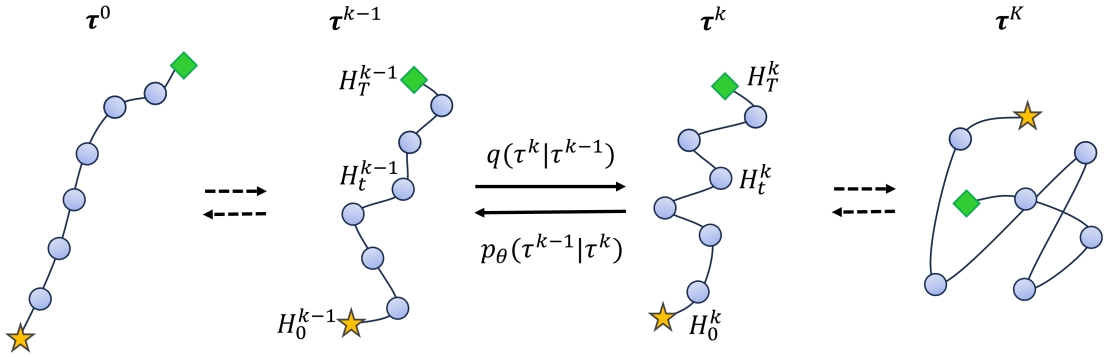


Fig. 2: Forward process and reverse process for paths. The paths are sequences of poses that start at the star (subscript 0) and end at the diamond (subscript T). Noise is added iteratively to each pose in the forward process until all structure is destroyed (superscipt K). The reverse process iteratively removes noise until a noiseless path remains (superscript 0).

when $\boldsymbol{x}^k$ and $\boldsymbol{x}^0$ are given:

$$
\begin{aligned}
p\left(\boldsymbol{x}^{k-1}|\boldsymbol{x}^k,\boldsymbol{x}^0\right) &= \mathcal{N}\left(\tilde{\boldsymbol{\mu}}^k\left(\boldsymbol{x}^k,\boldsymbol{x}^0\right),\tilde{\beta}^k\boldsymbol{I}\right), \\
\tilde{\boldsymbol{\mu}}^k\left(\boldsymbol{x}^k,\boldsymbol{x}^0\right) &= \frac{\sqrt{\bar{\alpha}^{k-1}}\beta^k}{1-\bar{\alpha}^k}\boldsymbol{x}^0 + \frac{\sqrt{\bar{\alpha}^k}\left(1-\bar{\alpha}^{k-1}\right)}{1-\bar{\alpha}^k}\boldsymbol{x}^k, \\
\tilde{\beta}^k &= \frac{1-\bar{\alpha}^{k-1}}{1-\bar{\alpha}^k}\beta^k,
\end{aligned}
\tag{5}
$$

Equation (4) can be rewritten as a function of the added noise $\boldsymbol{\epsilon}$ to $\boldsymbol{x}_k\left(\boldsymbol{x}_0,\boldsymbol{\epsilon}\right) = \sqrt{\bar{\alpha}_k}\boldsymbol{x}_0 + \sqrt{1-\bar{\alpha}_k}\boldsymbol{\epsilon}$, where $\boldsymbol{\epsilon} \sim \mathcal{N}\left(\boldsymbol{0},\boldsymbol{I}\right)$. Using this reparameterization, Ho et al. [15] derived a simplified training objective for a neural network $\boldsymbol{\epsilon_\theta}$ that estimates the noise $\boldsymbol{\epsilon}$:

$$
L\left(\theta\right) = \mathbb{E}_{k,\boldsymbol{\epsilon},\boldsymbol{x}^0}\left[\left\|\boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta\left(\boldsymbol{x}^k,k\right)\right\|^2\right]. \tag{6}
$$

1) Conditional sampling with DDPMs: In general, the reverse process is trained to turn samples from a standard normal distribution into samples that resemble those in the training dataset. Oftentimes, we have additional demands to the samples that are generated, i.e., the samples need to meet certain conditions. This is achieved through conditional sampling. A condition vector $\boldsymbol{c} \in \mathbb{R}^n$ encodes a property of the samples, for example, a class. Different techniques have been proposed for condition sampling, such as classifier guidance [16] and classifier-free guidance [24]. In this work, we opt to use classifier-free guidance rather than classifier guidance, since the latter involves training an additional classifier model, which increases the training cost and complexity.

Classifier-free guidance guides the reverse process to produce samples that meet the condition using an additional input to the DDPM. The condition $\boldsymbol{c}$ is added to $\boldsymbol{\epsilon}_\theta\left(\boldsymbol{x}^k,k\right)$ in Equation (6) and the noise prediction is subsequently rewritten as:

$$
\begin{aligned}
\tilde{\boldsymbol{\epsilon}}_{\boldsymbol{\theta}}\left(\boldsymbol{x}^k,k,\boldsymbol{c}\right) =& \boldsymbol{\epsilon}_\theta\left(\boldsymbol{x}^k,k,\varnothing\right) + \\
& w\left(\boldsymbol{\epsilon}_\theta\left(\boldsymbol{x}^k,k,\boldsymbol{c}\right) - \boldsymbol{\epsilon}_\theta\left(\boldsymbol{x}^k,k,\varnothing\right)\right),
\end{aligned}
\tag{7}
$$

where w is a weight factor to balance unconditional and conditional generation and $\varnothing$ is a null-token used as the unconditional embedding. To train the neural network, a random binary mask is applied to the condition embedding that is passed to the network at each training step. In this way, the network cannot use the conditional information for every prediction. Therefore, the DDPM simultaneously learns the conditional and unconditional relation.

Additionally, DDPMs are capable of inpainting. Fixing a component of $\boldsymbol{x}^k$ to a desired value by setting $x_i^k = \hat{x}_i$ at every diffusion step $k = K, K-1, ..., 0$ causes the DDPM to inpaint about the desired component.

III. Path planning with DDPMs

Path planning is defined as finding a collision-free motion from a starting configuration to a goal configuration [25]. In this work, the path is discretized into a sequence of poses assumed to be equidistant in time, although we are not limited to that. We indicate a path as: $\boldsymbol{\tau} = (H_0, H_1, ..., H_T)$, where H is an element of the Lie-group SE(3) [26] which encodes the position and orientation of a frame with respect to a reference frame.

Note: There are two indices to keep track of; the time index and the diffusion index. We use the convention of Janner et al. [17] to use subscripts to denote the time index and superscripts to denote the index of the diffusion process. For example, a pose at time step t and noise step k is hence written as H_t^k.

To enable DDPMs to generate paths, the diffusion process is defined for the state trajectory $\boldsymbol{\tau}$, giving $q\left(\boldsymbol{\tau}^K|\boldsymbol{\tau}^0\right)$ for the forward process and $p_\theta\left(\boldsymbol{\tau}^0\right)$ for the reverse process. Figure 2 depicts the diffusion process for a path. This is achieved by choosing coordinates for H_t and stacking all entries of $\boldsymbol{\tau}$ along the time dimension. The path $\boldsymbol{\tau}$ is then represented by a matrix to which all equations in Section II-A apply by setting $\boldsymbol{x}^k = \left[H_0^k, H_1^k, ..., H_T^k\right]$.

A. Path planning through conditional sampling

Sampling Gaussian noise and applying the reverse process yields entire paths that match the distribution of the data that the DDPM is trained on, without requiring autoregression along the time dimension. To ensure that the generated paths

go from the starting pose to the goal pose, inpainting is used. That is, H_0^k and H_T^k are fixed for all k, as is displayed in Figure 3. To sample paths that avoid obstacles, two conditional

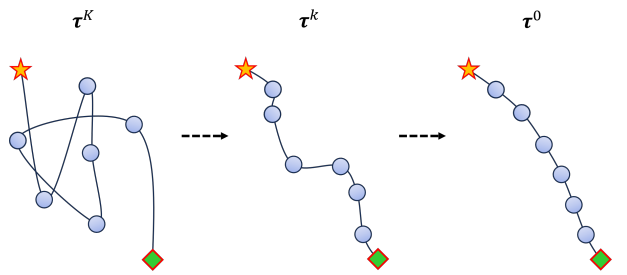


Fig. 3: Inpainting is used to set the start and goal poses of a generated path. By fixing the start and goal poses (with red rims) for all $k = K, K-1, ..., 0$, the DDPM adapts the other poses in order to match the training data distribution at $k = 0$.

sampling techniques are investigated. Firstly, we use classifier-free guidance. We use as condition the return of a path given by the sum of discounted rewards:

$$c = \sum_{t=0}^{T} \gamma^t r_t(H_t)\,, \tag{8}$$

where γ is the discount factor and $r_t(H_t)$ a reward based on the distance to the obstacles, see Equation (12) and (13). Conditioning only one these rewards results in the condition vector $\boldsymbol{c}$ in Equation (7) to be reduced to a scalar: $\boldsymbol{c} = c$. Classifier-free guidance, allows sampling paths with high returns and which should, therefore, steer clear of obstacles.

In related work, Carvalho et al. [21] introduced a different way of conditional sampling, inspired by classifier-guided sampling [16]. They define a cost function $J\left(\boldsymbol{\tau}^k\right)$, containing different terms to penalize collisions and promote smooth trajectories towards the target. During inference, they optimize this cost function by offsetting the mean of the reverse process in Equation (3) using the gradients of the cost function as follows:

$$\tilde{\boldsymbol{\mu}}_{\boldsymbol{\theta}}\left(\boldsymbol{\tau}^k, k\right) = \boldsymbol{\mu}_{\boldsymbol{\theta}}\left(\boldsymbol{\tau}^k, k\right) - \nabla_{\boldsymbol{\tau}^k} J(\boldsymbol{\tau}^k)|_{\boldsymbol{\tau}^k = \boldsymbol{\mu}_{\boldsymbol{\theta}}(\boldsymbol{\tau}^k, k)}. \tag{9}$$

This method will hereinafter be referred to as "cost-guided sampling", or simply "cost guidance". We will also investigate this method of conditional sampling to achieve obstacle avoidance, as well as the combined application of classifier-free guidance and cost guidance.

Furthermore, the probabilistic nature of DDPMs allows for a Monte Carlo approach to further optimize the paths. Instead of sampling a single path, DDPMs can be used to sample n different paths using the same conditions and selecting the path in the batch that minimizes a cost function $C\left(\boldsymbol{\tau}^k\right)$:

$$\boldsymbol{\tau}^* = \underset{\boldsymbol{\tau}}{\operatorname{argmin}}(C(\boldsymbol{\tau})). \tag{10}$$

B. Closed-loop planning

Although DDPMs do not have an inherent sense of temporal ordering, the quality of a path may still degrade along the time dimension due to discounting of the rewards, in the sense that future poses carry less weight in the reward structure. This issue can be addressed with a closed-loop approach to path planning. Rather than sampling a path and tracking it entirely, one can sample a path, track it for m steps, and then resample to update the rest of the path. Upon resampling, the planning horizon is shifted to achieve receding-horizon control, like Chi et al. [19] did. A block diagram of the process is shown in Figure 4. This approach resembles Model Predictive Control in the sense that we have a model that predicts future states and optimizes some cost function over a finite horizon.

C. Training data

In this work, we aim to create a path planner that is not robot-specific, nor requires expert examples, while still being able to produce useful paths. To showcase the generalization capabilities of the DDPM, the training dataset is synthetic and consists only out of straight lines with associated returns. To create a path in the training dataset, two random poses are sampled uniformly from a three-dimensional workspace. The other poses are obtained through separately interpolating the positions and orientations between the two initial poses, such that all positions are on a straight line. Note that these paths can be anywhere in the workspace, even in collision with obstacles. The dataset contains 60,000 paths.

In order to train the network to be able to avoid obstacles, the paths are associated with a return value (Equation (8)). We consider two options for $r_t\left(H_t\right)$. The first one will be referred to as dense rewards. It penalizes not only the collisions themselves, but also the distance towards the nearest obstacle. A pose H_t can be described by a matrix that can separated into a rotation matrix $\boldsymbol{R}_t$ that encodes its orientation and a vector $\boldsymbol{p}_t$ that encodes its position [26];

$$\boldsymbol{H}_t = \begin{bmatrix} \boldsymbol{R}_t & \boldsymbol{p}_t \\ \boldsymbol{0}^\top & 1 \end{bmatrix}, \tag{11}$$

where $\top$ refers to the transpose operator and $\boldsymbol{0}$ is a column vector of three zeros. Using the position of the nearest point of the nearest obstacle $\boldsymbol{p}_{\text{obst}}$, the reward is then:

$$r_{t^{\text{dense}}}\left(H_t\right) = \begin{cases} -1 & \text{when in collision} \\ -e^{-a \cdot \|\boldsymbol{p}_{\text{obst}} - \boldsymbol{p}_t\|_2} & \text{otherwise,} \end{cases} \tag{12}$$

where $a \in \mathbb{R}$ is a scaling parameter and $\|\boldsymbol{p}_{\text{obst}} - \boldsymbol{p}_t\|_2$ is the Euclidean norm of $\boldsymbol{p}_{\text{obst}} - \boldsymbol{p}_t$. In the following experiments, we consider static obstacles, a discount factor $\gamma = 0.99$ and $a = 46$.

The second option will be referred to as sparse rewards and it penalizes only the collisions themselves:

$$r_{t^{\text{sparse}}}\left(H_t\right) = \begin{cases} -1 & \text{when in collision,} \\ 0 & \text{otherwise.} \end{cases} \tag{13}$$

IV. Experiments and Results

The following section describes various simulations and experiments performed with the DDPM. Training and inference settings that are fixed throughout this work are in Table I.

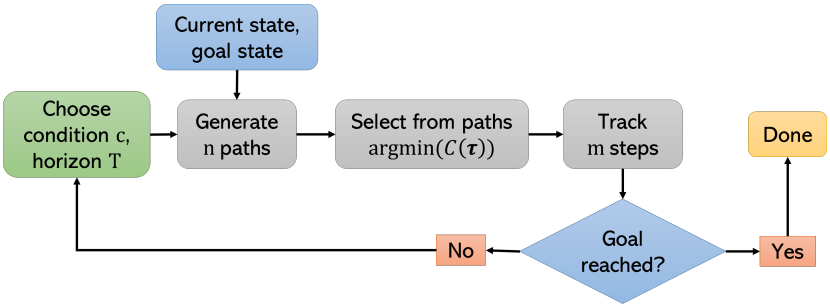


Fig. 4: Block diagram of a closed-loop control configuration for a DDPM planner

TABLE I: Model settings kept fixed throughout the experiments.

Setting	Value
Number of diffusion steps K	200
Path length in training T	32
Classifier-free guidance scale w	1.2
End effector loss weight w_{ee}	10^{-3}
Collision loss weight w_c	10^{-3}
Neural network architecture	Convolutional U-Net
Number of model parameters	$\sim$60M
Optimizer	Adam
Learning rate	$2 \cdot 10^{-4}$
Batch size	32
Number of training steps	200k

A. Impact of horizon and returns on classifier-free sampling

Since the training dataset contains paths in the entire workspace, even inside obstacles, the avoidance of obstacles is entirely achieved through conditional sampling. When using classifier-free guidance, these conditions are learned from the training data, which only consists out of straight lines. The extent to which the planner can generate paths that avoid an obstacle in between the start pose and goal pose depends on the DDPM's generalization capabilities, but also on the hyperparameters used in inference. Two important hyperparameters are the planning horizon and the returns on which the sample is conditioned. To investigate these, a simple scenario is considered in which the DDPM is used in open-loop configuration to inpaint a path between a starting pose and a goal pose, while an obstacle is positioned in between. Figure 5 shows simulations of the paths for various combinations of these parameters. For the model trained on sparse rewards, only a returns value of zero is considered, since this is the only value that represents a collision-free path.

B. Inpainting with repeated goal states

In Figure 5 it is clear that the generated paths generally do not connect well to the goal pose. We hypothesize that the long horizon makes the single goal state have relatively little weight in the reverse process, causing the gap at the end of the path. To assess this, we used a DDPM trained with dense rewards to inpaint a path with a horizon length of $T = 256$ and conditioned on a return value of $c = -0.001$, which are the same parameters as used in the bottom-right of Figure 5a. However, instead of fixing only the final state to be the goal state, we fixed the final i states to be the goal state. Simulations of the resulting paths are shown in Figure 6.

C. Real-world experiments

To validate the path planner, path planning tasks are performed with a Franka Emika Panda robot. Two different environments were created for validating the planner. For both environments, we define a cuboidal region in which some objects are placed. All training data is generated within this region. The first environment is the same as used in the previously presented simulations and contains one cuboid obstacle. The second environment contains two cuboid obstacles; a big cuboid and a small cube placed on top of it. The obstacles are visible in Figure 7 and 8. The starting and goal poses are chosen such that no collision-free straight path exists between the start and the goal.

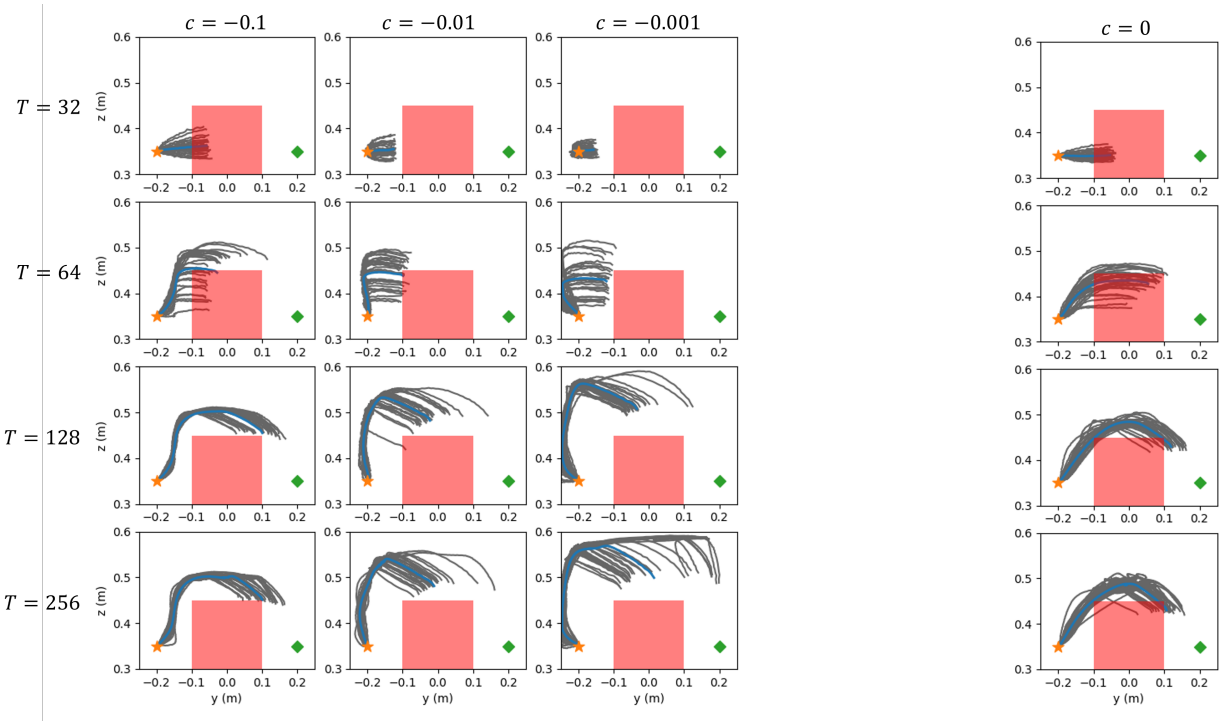
Different methods of conditional sampling are tested. Firstly, we perform experiments using classifier-free guidance, using planners that were trained with dense rewards and planners trained with sparse rewards. Secondly, in order to make comparisons, we use the cost-guided conditional sampling approach put forward in [21]. The cost function used in the experiments is the sum of two relevant cost terms used in [21]; the end-effector cost and the collision cost. The end effector cost is given by:

$$g_{ee}\left(H_t\right) = d_{SE(3)}\left(H_t, H_g\right) , \tag{14}$$

where $d_{SE(3)}\left(H_t, H_g\right) = \|\boldsymbol{p}_t - \boldsymbol{p}_g\|_2^2 + \|LogMap\left(\boldsymbol{R}_t^T \boldsymbol{R}_g\right)\|_2^2$. The $LogMap()$ is a function from Lie Theory [26] that can be used to quantify the difference between two orientations. The collision cost is given by:

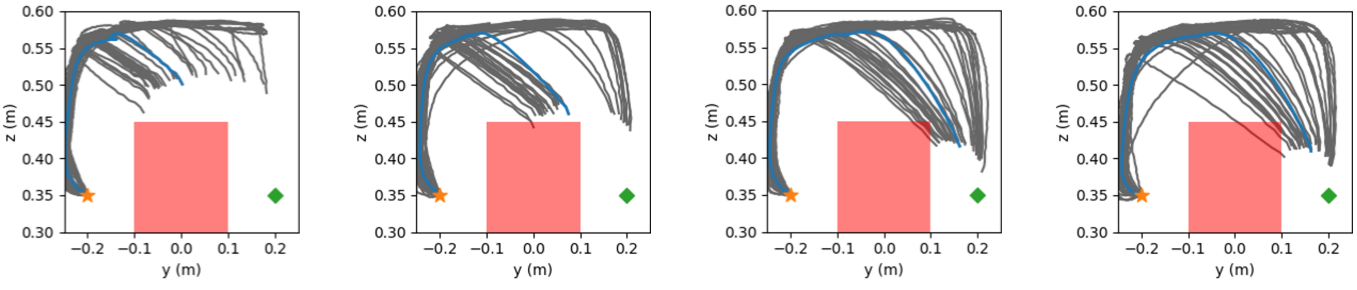
$$g_c\left(H_t\right) = \begin{cases} -d(p_t) & \text{if } d(p_t) \leq 0 \\ 0 & \text{if } d(p_t) > 0 \end{cases} \tag{15}$$

where $d\left(\boldsymbol{p}_t\right)$ is a differentiable signed distance function from the position $\boldsymbol{p}_t$ to the surface of the nearest obstacle, defined to be positive when p_t is outside the obstacle. The total cost function is $J\left(\boldsymbol{\tau}\right) = \sum_{t=1}^{T-1} w_{ee} g_{ee}\left(H_t\right) + w_c g_c\left(H_t\right)$, with the weighting factors w_{ee} and w_c. Thirdly, we experiment with a novel way of sampling through combining classifier-free sampling with cost-guided sampling. In total, this results



(a) (b)

Fig. 5: Open-loop path generation with classifier-free guidance from the star to the diamond for various horizons T and conditioned on various returns c. In (a), dense rewards are used and the returns are varied. In (b) sparse rewards are used and the returns are set to $c = 0$. The mean position of thirty generated paths is indicated by the blue line.



(a) (b) (c) (d)

Fig. 6: Open-loop path generation with classifier-free guidance with a varying number of repetitions of goal state inpainting. The number of repetitions are $1, 2, 5$, and 10, for (a), (b), (c), and (d), respectively. Conditioning on dense rewards of $c = -0.001$ with a horizon of $T = 256$ were used. The mean position of thirty generated paths is indicated by the blue line.

in five combinations for conditional sampling; *i)* classifier-free guidance with dense rewards; *ii)* classifier-free guidance with sparse rewards; *iii)* cost guidance; *iv)* classifier-free guidance with dense rewards combined with cost guidance; *v)* classifier-free guidance with sparse rewards combined with cost guidance.

The path planner was deployed in a receding-horizon configuration as explained in Section III-B. The following hyperparameters were chosen; the planning horizon $h = 128$, number of steps tracked per execution $m = 64$, number of paths produced per batch $n = 5$, returns condition with dense rewards $c_{\text{dense}} = -0.01$, returns condition with sparse rewards $c_{\text{sparse}} = 0$. In inpainting, the goal state was repeated five times at the end of the path. Of the five paths generated per batch, the path with the lowest returns calculated with dense rewards was selected for tracking, regardless of the conditional sampling method that was used. The generated paths were tracked using cartesian impedance control. A limitation of this experiment is that the orientation of the end effector is kept fixed to prevent the robot arm from twisting itself into a configuration in which

it cannot track the planned path anymore due to joint limits.

Figures 7 and 8 show video frames from recordings of the experiment of the scene with one obstacle and two obstacles, respectively. The only conditional sampling methods that did not result in any collision-free paths for a particular scenario are cost guidance (combination *iii)*) and using classifier-free guidance with sparse rewards combined with cost guidance (combination *v)*), both in the environment with a single obstacle. All other conditional sampling methods could repeatedly generate collision-free paths in both scenarios. Sparse rewards resulted in paths much closer to the obstacles compared to dense rewards, regardless of the inclusion of cost guidance. In the scenario with two obstacles, using only classifier-free guidance with dense rewards (combination *i)*) often resulted in lingering before crossing from one side of the smaller obstacle to the other.

V. Discussion

The parameter sweep in Figure 5 shows the impact of the horizon and returns on the generated paths with classifier-free guidance. Firstly, it is evident that in all configurations, coherent paths are generated that start at the intended starting pose and tend towards the goal pose, even for horizons that are much longer than seen in training. The paths that were generated using sparse rewards are close to the obstacles, whereas for dense rewards, the distance to the obstacle is strongly influenced by the rewards. For the model trained with dense rewards, collisions are avoided when conditioning on high rewards, but choosing them too high can lead to overly conservative paths. However, with sparse rewards, there is no control over the trade-off between generating a path that closely approaches the goal state and steering clear of the obstacle, which leads to many paths slightly violating the condition of obstacle avoidance. For both types of rewards, the model is able to generate paths that are not in collision with the obstacle, when using a sufficiently long horizon. However, the consistent gap between the final planned state and the goal state prevents them from being directly useful for tracking. This is not merely an issue of the horizon being too short, as even doubling the horizon from 128 to 256 hardly improves this connection.

When it comes to closing the gap between the final planned state and the goal state, Figure 6 shows that repeating the goal state in inpainting may be a useful trick. Even adding a single additional goal state to the reverse process already made the paths approach the goal more closely. However, there appears to be a limit, as inpainting with ten copies of the goal state yields no further improvement compared to using five states.

The real-world experiments validate that a capable path planner can be made using various configurations of conditional sampling. However, using only cost guidance or using classifier-free guidance with sparse rewards in combination with cost guidance did not result in collision-free paths for the scenario with one obstacle. This shows that the optimization procedure carried out in cost guidance benefits from a good prior that can be provided by a DDPM that uses classifier-free guidance with dense rewards. With the cost function used here, cost-guided sampling on its own or in combination with classifier-free guidance with sparse rewards may fail because there is no term that promotes to keep a distance to the obstacles beyond the edge of the obstacle, which opens the possibility of the end-effector cost bringing the path into collision. On the other hand, we observed that path planning with classifier-free guidance with dense rewards could be improved by using the cost function guidance. In particular in the scenario with two obstacles, the path planner without cost function guidance would often linger in place to avoid moving closer to the obstacles, even though that was required to reach the goal pose. Adding cost function guidance pushes poses to be close to the goal pose, which resolved this shortcoming.

A. Future work

Future research could further investigate the trade-off that exists between returns conditioning, goal state inpainting and the planning horizon for planning with DDPMs, since selecting the right hyperparameters for inference is crucial for generating high-quality paths. Furthermore, the line of research concerned with training DDPMs with low-quality demonstrations could be continued by exploiting the structure that the context of robotic path planning provides. For example, kinematic path constraints, like large distances between consecutive poses may also be enforceable through methods similar to those used in physics-informed diffusion models [27], [28]. The inclusion of the robotic context can be further aided by using models that diffuse directly on SE(3), like [29], [30], in order to obtain a geometric description of the diffusion process.

VI. Conclusion

In this work, we proposed an approach to designing a path planner for a robotic end effector using DDPMs that does not use expert demonstrations. Instead, using only straight lines as training data, the path planner can produce complex paths that avoid obstacles by utilizing the generalization capabilities of the DDPM and various options of conditional sampling. Deploying the DDPM in a closed-loop receding-horizon control scheme in order to continuously update the plan allowed the path planner to reach targets that it could not effectively reach in an open-loop configuration.

References

[1] J. T. Schwartz and M. Sharir, "A survey of motion planning and related geometric algorithms," *Artificial Intelligence*, vol. 37, no. 1-3, pp. 157–169, Dec. 1988, ISSN: 0004-3702. DOI: 10.1016/0004-3702(88)90053-7.

[2] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *International Journal of Robotics Research*, vol. 30, no. 7, pp. 846–894, Jun. 2011, ISSN: 02783649. DOI: 10 . 1177 / 0278364911406761.

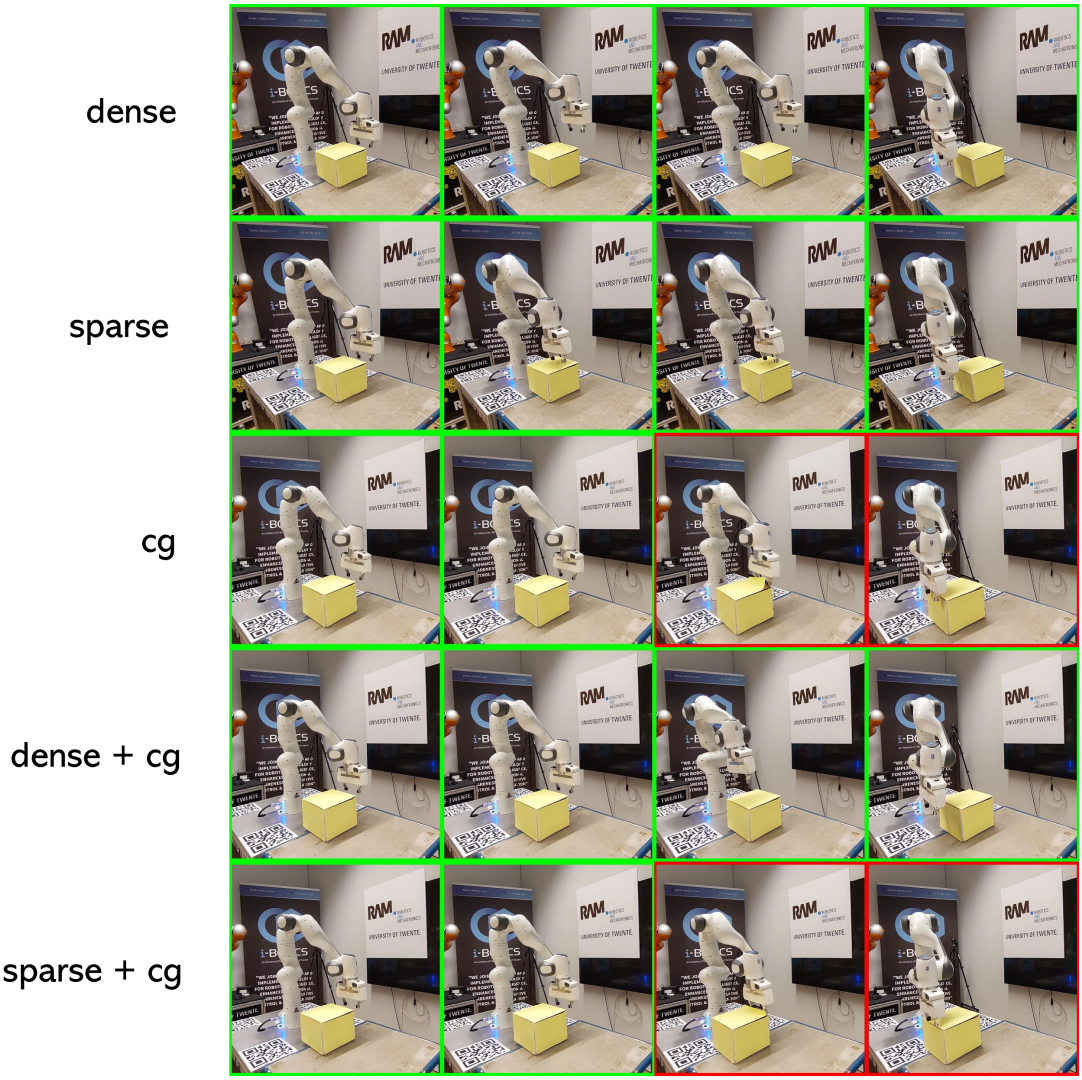


Fig. 7: Frames from recordings of using various conditional sampling techniques to avoid a single obstacle. From top to bottom, the sampling techniques are: *i)* classifier-free guidance with dense rewards; *ii)* classifier-free guidance with sparse rewards; *iii)* cost guidance; *iv)* classifier-free guidance with dense rewards combined with cost guidance; *v)* classifier-free guidance with sparse rewards combined with cost guidance. Green edges indicate that the robot is not in collision, whereas red frames show that the robot is in collision.

[3] L. E. Kavraki, P. Švestka, J. C. Latombe, and M. H. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE Transactions on Robotics and Automation*, vol. 12, no. 4, pp. 566–580, 1996, ISSN: 1042296X. DOI: 10.1109/70.508439.

[4] S. M. Lavalle, "Rapidly-exploring random trees: A new tool for path planning," Iowa State University, Tech. Rep., Oct. 1998.

[5] R. S. Sutton and A. G. Barto, *Reinforcement learning : an introduction*. Cambridge, Massachusetts : The MIT Press, 2018, ISBN: 9780262352703.

[6] V. Mnih, K. Kavukcuoglu, D. Silver, *et al.*, "Playing Atari with Deep Reinforcement Learning," Dec. 2013.

[7] V. Mnih, K. Kavukcuoglu, D. Silver, *et al.*, "Human-level control through deep reinforcement learning," *Nature 2015 518:7540*, vol. 518, no. 7540, pp. 529–533, Feb. 2015, ISSN: 1476-4687. DOI: 10.1038/NATURE14236.

[8] A. Konar, I. G. Chakraborty, S. J. Singh, L. C. Jain, and A. K. Nagar, "A deterministic improved q-learning for path planning of a mobile robot," *IEEE Transactions on Systems, Man, and Cybernetics Part A:Systems and Humans*, vol. 43, no. 5, pp. 1141–1153, 2013, ISSN: 10834427. DOI: 10.1109/TSMCA.2012.2227719.

[9] E. S. Low, P. Ong, and K. C. Cheah, "Solving the optimal path planning of a mobile robot using improved Q-learning," *Robotics and Autonomous Systems*, vol. 115,

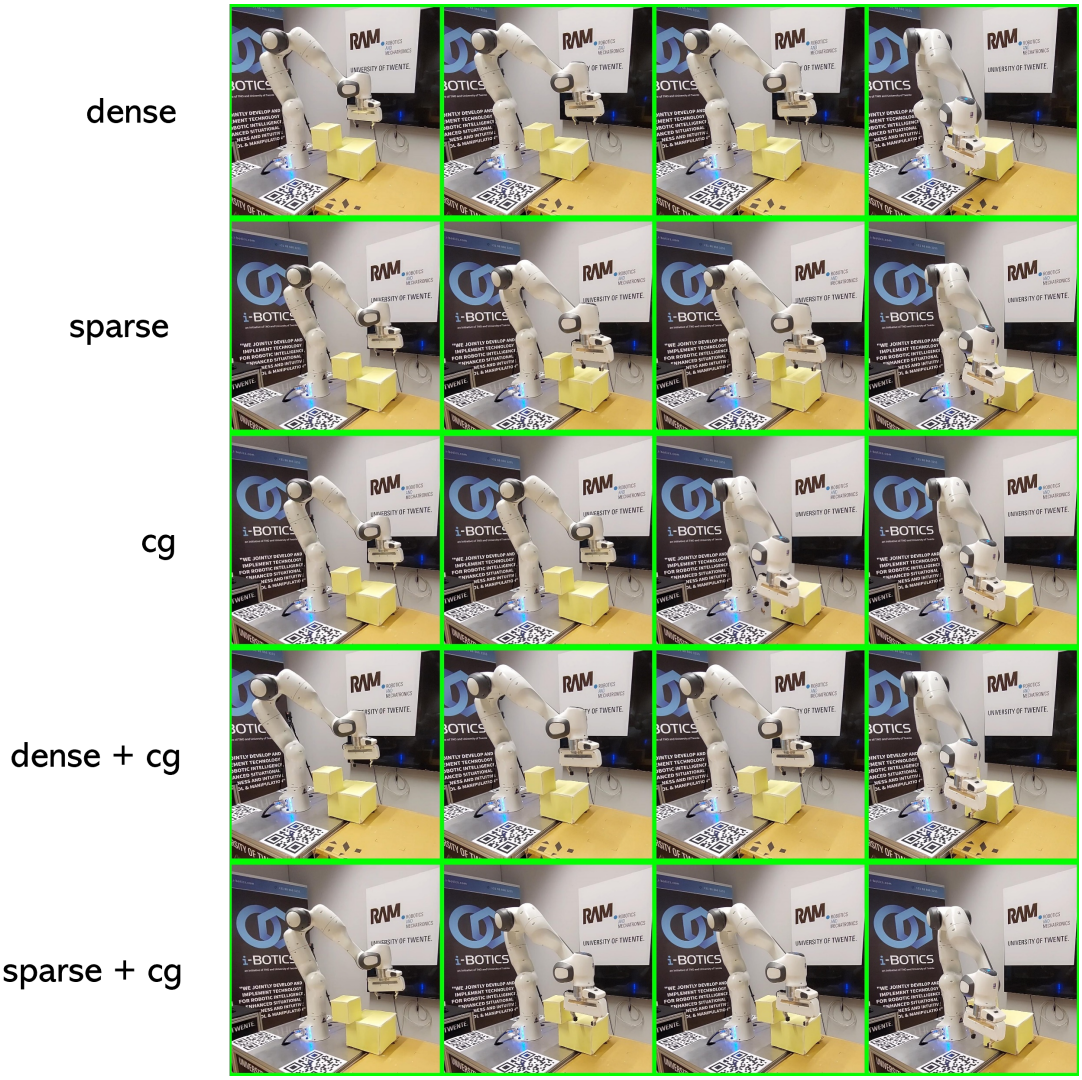


Fig. 8: Frames from recordings of using various conditional sampling techniques to avoid two obstacles. From top to bottom, the sampling techniques are: *i)* classifier-free guidance with dense rewards; *ii)* classifier-free guidance with sparse rewards; *iii)* cost guidance; *iv)* classifier-free guidance with dense rewards combined with cost guidance; *v)* classifier-free guidance with sparse rewards combined with cost guidance. Green edges indicate that the robot is not in collision, whereas red frames show that the robot is in collision.

pp. 143–161, May 2019, ISSN: 0921-8890. DOI: 10.1016/J.ROBOT.2019.02.013.

[10] K. Asadi, D. Misra, S. Kim, and M. L. Littman, "Combating the Compounding-Error Problem with a Multi-step Model," May 2019.

[11] C. Xiao, Y. Wu, C. Ma, D. Schuurmans, and M. Müller, "Learning to Combat Compounding-Error in Model-Based Reinforcement Learning," Dec. 2019. [Online]. Available: https://arxiv.org/abs/1912.11206v1.

[12] H. Attias, "Planning by Probabilistic Inference," in *Proceedings of the Ninth International Workshop on Artificial Intelligence and Statistics*, B. J. Bishop Christopher M.Frey, Ed., PMLR, Jan. 2003, pp. 9–16.

[13] M. Botvinick and M. Toussaint, "Planning as inference," *Trends in Cognitive Sciences*, vol. 16, no. 10, pp. 485–488, Oct. 2012, ISSN: 1364-6613. DOI: 10.1016/J.TICS.2012.08.006.

[14] J. Sohl-Dickstein, E. Weiss, N. Maheswaranathan, and S. Ganguli, "Deep Unsupervised Learning using Nonequilibrium Thermodynamics," in *Proceedings of the 32nd International Conference on Machine Learning*, F. Bach and D. Blei, Eds., PMLR, Jul. 2015, pp. 2256–2265.

[15] J. Ho, A. Jain, and P. Abbeel, "Denoising Diffusion Probabilistic Models," *Advances in Neural Information Processing Systems*, vol. 33, pp. 6840–6851, 2020.

[16] P. Dhariwal and A. Nichol, "Diffusion Models Beat GANs on Image Synthesis," 2021.

[17] M. Janner, Y. Du, J. B. Tenenbaum, and S. Levine, "Planning with Diffusion for Flexible Behavior Synthesis," 2022.

[18] A. Ajay, Y. Du, A. Gupta, J. Tenenbaum, and T. Jaakkola, "Is Conditional Generative Modeling all you need for Decision-Making?," 2023.

[19] C. Chi, S. Feng, Y. Du, *et al.*, "Diffusion Policy: Visuomotor Policy Learning via Action Diffusion," *Robotics: Science and Systems*, 2023.

[20] J. Liu, M. Stamatopoulou, and D. Kanoulas, "DiPPeR: Diffusion-based 2D Path Planner applied on Legged Robots," 2024.

[21] J. Carvalho, A. T. Le, M. Baierl, D. Koert, and J. Peters, "Motion Planning Diffusion: Learning and Planning of Robot Motions with Diffusion Models," *IEEE International Conference on Intelligent Robots and Systems*, pp. 1916–1923, Aug. 2023, ISSN: 21530866. DOI: 10.1109/IROS55552.2023.10342382.

[22] K. Saha, V. Mandadi, J. Reddy, *et al.*, "EDMP: Ensemble-of-costs-guided Diffusion for Motion Planning," *Proceedings - IEEE International Conference on Robotics and Automation*, pp. 10 351–10 358, 2024. DOI: 10.1109/ICRA57147.2024.10610519.

[23] A. Q. Nichol and P. Dhariwal, "Improved Denoising Diffusion Probabilistic Models," in *Proceedings of the 38th International Conference on Machine Learning*, M. Meila and T. Zhang, Eds., PMLR, Jul. 2021, pp. 8162–8171.

[24] J. Ho and T. Salimans, "Classifier-Free Diffusion Guidance," 2022.

[25] M. W. Spong, S. Hutchinson, and M. Vidyasagar, "Path and Trajectory Planning," in *Robot Modeling and Control*, John Wiley & Sons, 2020, ch. 7, ISBN: 978-1-119-52399-4.

[26] J. Solà, J. Deray, and D. Atchuthan, "A micro Lie theory for state estimation in robotics," 2018.

[27] D. Shu, Z. Li, and A. Barati Farimani, "A physics-informed diffusion model for high-fidelity flow field reconstruction," *Journal of Computational Physics*, vol. 478, p. 111 972, Apr. 2023, ISSN: 0021-9991. DOI: 10.1016/J.JCP.2023.111972.

[28] J.-H. Bastek, W. Sun, and D. M. Kochmann, "Physics-Informed Diffusion Models," Mar. 2024.

[29] J. Yim, B. L. Trippe, V. De Bortoli, *et al.*, "SE(3) diffusion model with application to protein backbone generation," *Proceedings of Machine Learning Research*, vol. 202, pp. 40 001–40 039, Feb. 2023, ISSN: 26403498.

[30] H. Jiang, M. Salzmann, Z. Dang, J. Xie, and J. Yang, "SE (3) Diffusion Model-based Point Cloud Registration for Robust 6D Object Pose Estimation," in *Thirty-seventh Conference on Neural Information Processing Systems*, 2023.