Motion Planning Diffusion: Learning and Adapting Robot Motion Planning with Diffusion Models

João Carvalho[1], An T. Le[1], Piotr Kicki[2,3], Dorothea Koert[1,4], and Jan Peters[1,5,6]

***Abstract*—The performance of optimization-based robot motion planning algorithms is highly dependent on the initial solutions, commonly obtained by running a sampling-based planner to obtain a collision-free path. However, these methods can be slow in high-dimensional and complex scenes and produce non-smooth solutions. Given previously solved path-planning problems, it is highly desirable to learn their distribution and use it as a prior for new similar problems. Several works propose utilizing this prior to bootstrap the motion planning problem, either by sampling initial solutions from it, or using its distribution in a maximum-a-posterior formulation for trajectory optimization. In this work, we introduce Motion Planning Diffusion (MPD), an algorithm that learns trajectory distribution priors with diffusion models. These generative models have shown increasing success in encoding multimodal data and have desirable properties for gradient-based motion planning, such as cost guidance. Given a motion planning problem, we construct a cost function and sample from the posterior distribution using the learned prior combined with the cost function gradients during the denoising process. Instead of learning the prior on all trajectory waypoints, we propose learning a lower-dimensional representation of a trajectory using linear motion primitives, particularly B-spline curves. This parametrization guarantees that the generated trajectory is smooth, can be interpolated at higher frequencies, and needs fewer parameters than a dense waypoint representation. We demonstrate the results of our method ranging from simple 2D to more complex tasks using a 7-dof robot arm manipulator. In addition to learning from simulated data, we also use human demonstrations on a real-world pick-and-place task. The experiment results show that diffusion models are strong priors for encoding multimodal trajectory distributions for optimization-based motion planning.**
https://sites.google.com/view/motionplanningdiffusion

***Index Terms*—Deep Learning, Learning to Plan, Motion Planning, Diffusion Models.**

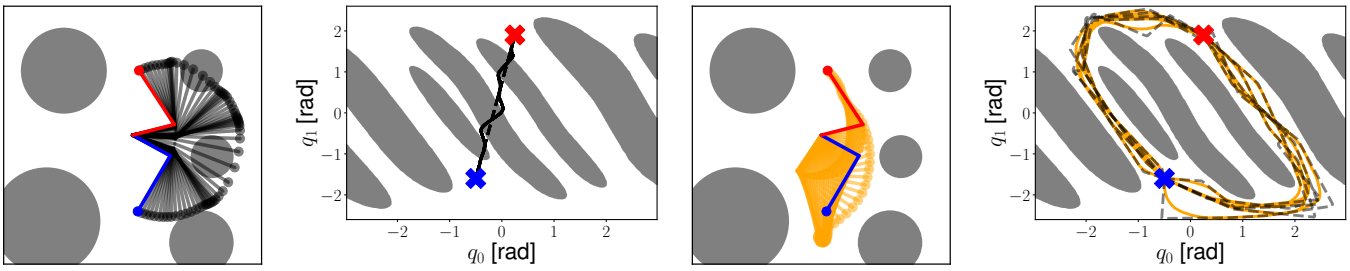
I. Introduction

Autonomous robots are becoming a ubiquitous technology, and motion planning is an important core component. Among several methods [1], optimization-based motion planning is a popular approach for solving robot motion planning problems [2]–[4]. Methods in this category formulate planning as an optimization problem, where the goal is to find a trajectory that minimizes a cost function, e.g., collision avoidance, while satisfying constraints, such as joint limits. Popular methods, such as CHOMP [2], TrajOpt [4] and GPMP/GPMP2 [3], are heavily dependent on initialization since a bad initial trajectory can lead to getting stuck in local minima and failing to find a collision-free path [4]. Without prior information, the initial trajectory is commonly assumed to be a straight line in the configuration space. However, [3] mentions that in practice, a straight-line initialization might fail for some tasks, e.g., those with narrow passages. Thus, for complex motion planning tasks, it is common to first run a sampling-based planner, such as RRT-Connect [5], followed by an optimization-based planner for trajectory smoothing [6].

To better illustrate this issue, in fig. 1 we show a motion planning task for a planar 2-link robot. The task is to obtain a smooth joint trajectory from a start to a goal configuration without colliding with the environment. Figure 1b shows how narrow passages appear in the configuration space. When using a straight-line trajectory in the configuration space, as in fig. 1b, CHOMP cannot escape local minima to find a collision-free path. As shown in fig. 1d, if first a global sampling-based planner is used, particularly RRT-Connect, which produces a collision-free path, then the initial trajectory used in CHOMP is already collision-free, albeit being non-smooth. Using CHOMP, this trajectory is further optimized for smoothness while avoiding collisions. Additionally, in fig. 1d, we can see two modes to traverse the configuration space found by multiple runs of RRT-Connect. These insights also transfer to higher dimensions. These examples depict characteristics that priors should have: being (almost) collision-free, representing complex trajectories, and encoding multimodality.

Instead of computing an initial path with a sampling-based planner, another approach is to learn from previous motion planning solutions a manifold of trajectories that are good initializations [7], [8]. Several choices must be made when encoding a prior distribution, depending on the nature of the data and the sampling process. The simplest prior is building a memory of past solutions and using them as initializations [9]. However, this approach is not scalable, and it is not straightforward to generalize to new start and goal configurations other than interpolating between the closest solutions using, e.g., the k-nearest neighbors. A parametrized model removes these issues but must consider whether the data is unimodal or multimodal and how fast we can sample from this model. If the prior trajectories are unimodal, Gaussian

Corresponding author: João Carvalho, joao@robot-learning.de
[1]Intelligent Autonomous Systems Lab, Computer Science Department, Technical University of Darmstadt, Germany; [2]Poznan University of Technology, Poland; [3]IDEAS, Warsaw, Poland; [4]Centre for Cognitive Science, Technical University of Darmstadt, Germany; [5]German Research Center for AI (DFKI), Research Department: SAIROL, Darmstadt, Germany; [6]Hessian.AI, Darmstadt, Germany



(a) CHOMP (task space) (b) CHOMP (joint space) (c) RRT-Connect + CHOMP (task space) (d) RRT-Connect + CHOMP (joint space)

Fig. 1. These figures illustrate the need for using *good* initializations for optimization-based motion planning methods when there are narrow passages in the robot's configuration space. In (a) and (c) the task space of a 2-link robot arm is depicted. The grey circles are obstacles that the robot needs to avoid while moving from the start configuration to the goal configuration. One robot trajectory corresponding to (b) and (d) is also shown. In (b) and (d), the configuration-free space is shown, where the grey areas indicate collision in the task space. In (a)/(b), we run CHOMP using a straight-line initialization in the configuration space (black dashed lines). The resulting optimized trajectories (solid lines) show that the robot gets stuck in a local minimum and cannot find a collision-free path. In (c)/(d) we first run RRT-Connect (dashed black lines) and initialize CHOMP with these trajectories. The resulting optimized trajectories (solid orange lines) are collision-free. The initial paths are already collision-free, and during optimization, they are shortened and smoothened using CHOMP. It is also possible to observe the two modes found by RRT-Connect.

distributions representing the parameters of a Probabilistic Movement Primitive (ProMP) [10] or a Gaussian Process (GP) of the waypoints of a discretized trajectory [8], [11] can be used. These, however, fail to encode multimodal data. For multimodality, GMMs [12] can be used, but they are not easy to train for high-dimensions and large datasets. Additionally, choosing the number of modes is not trivial [13]. In contrast to the simple parametric models, deep generative models, such as Variational Auto Encoders (VAEs) [14], Generative Adversarial Networks (GANs) [15], Energy-Based Models (EBMs) [16], and diffusion models [17], [18], can express multimodal distributions due to the modeling power of deep neural networks. Methods such as VAEs and GANs are easy to sample from but show problems such as mode-collapse or training instability [19]. EBMs can be difficult to train and sample from [20]. Therefore, we chose to model prior trajectory distributions using diffusion models, which present several important qualities when learning from demonstrations. First, they encode well multimodal data [21], as there might be several collision-free paths [22]. Second, diffusion models are empirically easier to train than others (e.g., GANs and EBMs) [21]. Finally, after learning, given an external likelihood function to optimize, sampling from a posterior distribution is done by smoothly biasing samples from the prior toward the high-likelihood regions during the denoising process [23].

The key contributions of our work are:

- Motion Planning Diffusion (MPD) - a novel method that combines learning and adaptation of robot trajectories using diffusion models; implemented as a diffusion-based generative framework that learns trajectory priors in a compact B-spline parameterization.
- A cost-guided posterior sampling, following the planning-as-inference framework, that blends learned trajectory priors with task-specific objectives during the reverse diffusion process, enabling the generation of diverse, feasible, and collision-free trajectories.
- An empirical evaluation demonstrating MPD's effectiveness across a variety of planning problems, ranging from simple 2D setups to environments with a 7-dof robot manipulator, and a real-world pick-and-place task learned from human demonstrations via kinesthetic teaching.

The rest of this article is structured as follows. Section II presents related works on learning priors for motion planning and how diffusion models are used in robotics and motion planning. In section III we provide the necessary background and present our method. To highlight the benefits of diffusion models as priors, in section IV we evaluate our method against several baselines in simulated and real-world tasks. In section V we discuss the limitations of our work and propose further research ideas. Lastly, section VI summarizes the key points of this article.

This work is an extended version of our previous work [24], in which we added several improvements and more clarification and details. Instead of learning a model on dense waypoints, we propose using parametric trajectories with lower-dimensional representations that ensure smoothness, namely B-splines. Previously, we only considered contexts as joint positions at the start and goal. In this work, we use the desired end-effector goal pose as a conditioning variable, which is more natural for some planning tasks. We expanded the related work section, described MPD in more detail, and performed more experiments in simulated environments. To show the method's applicability to real-world tasks, we learn a prior distribution from human demonstrations via kinesthetic teaching for a pick-and-place task, and adapt it by using obstacles not present in the training setup.

II. Related Work

A large body of literature exists on learning to plan for robotics. In this section, we discuss classical works in path and motion planning (section II-A), and methods that combine learning methods with optimization-based motion planning approaches (section II-B). Additionally, we present an overview of how diffusion models are used in robotics and, more specifically, in motion planning (section II-C).

A. Path and motion planning

Path and motion planning are fundamental components of any robotics system. Path planning aims to find a collision-free path, and motion planning aims to find a trajectory (a time-dependent path) that prevents the robot from colliding with its environment and respects its joint limits. Importantly, for robot applications, the solutions should be smooth to avoid jerky movements. There are two main branches of path and motion planning: sampling-based and optimization-based.

Sampling-based planning algorithms ensure probabilistic completeness by conducting an extensive search over the whole configuration space. They sample points, evaluate whether they are in collision, and connect them to form a path. These include classical algorithms such as PRM [25], RRT [26], RRTConnect [5], or their improved versions, PRM/RRT* [27], Informed-RRT* [28], BIT* [29] and AIT* [30]. One drawback of these planners is that they simply connect configurations in such a way that the resultant paths are typically non-smooth. Hence, after finding a path, it must be post-processed by a smoother guaranteeing the motion is collision-free [31].

On the other hand, optimization-based motion planners search for a collision-free movement by locally optimizing a trajectory either via gradient descent or stochastic optimization, aiming for smoothness while satisfying other objective constraints. These methods can be seen as performing sampling-based planners' search and smoothening steps in one step. Gradient-based methods include CHOMP [2], TrajOpt [4] and GPMP/GPMP2 [3]. Stochastic optimization methods based on path integral formulations [32] include STOMP [33], Stochastic GPMP [34] and MGPTO [35]. Other works solve motion planning by optimal transport [36].

B. Learning priors for optimization-based motion planning

Optimization-based planning methods often use an uninformed initial/prior distribution, whose mean is a constant-velocity straight line between the start and goal configurations. As a deterministic method, CHOMP has no initial distribution. STOMP uses a distribution with high entropy in the middle of the trajectory and decreasing entropy towards the start and goal points. GPMP/GPMP2 uses a Gaussian Process prior and likelihoods of the exponential family to perform maximum-a-posteriori (MAP) trajectory optimization.

To find good initializations, we might have access to a database/library containing a set of motion-planning tasks and their respective solutions. When presented with a new task, some methods take the k-nearest neighbor (k-NN) solution in the database [9], [37], [38]. k-NN approaches may achieve good results in low-dimensional tasks but have two issues: they need to keep a growing database, which can hurt memory, and suffer from the curse of dimensionality when computing distances in higher dimensions. Alternatively, other methods, commonly named memory of motion, build a function approximator that maps tasks to solutions [9], [39]. Then, they query the learned function at inference time to obtain an initial solution to warm start a trajectory optimizer. Care is needed when approximating the function mapping from tasks to trajectories. If a deterministic function approximator is learned using mean squared error [40], [41], the learned model averages the solutions, resulting in a poor fit. A better approach is to learn a multimodal distribution of trajectories [9], [42]. With an expressive enough model, we can recover all the distribution modes effectively. Several of the mentioned methods first obtain a sample from the prior distribution and then optimize it, which can lead to the final solution being far away from the prior distribution. This can be problematic if the prior is constructed from human demonstrations, which we'd like to be close to when performing optimization. In turn, in our work, we use a diffusion model to build a memory of motions, which stay close to the prior while minimizing a cost function.

Other methods propose using learning from human demonstrations [43] to build trajectory distribution priors. Then, given a likelihood function, they use sampling or optimization methods to obtain the maximum-a-posterior solution, a sample from the posterior distribution, or even recover the posterior distribution in closed form. [10] introduced DEBATO, which learns a ProMP of trajectories containing the position of a human hand during demonstrations. The ProMP weights are Gaussian distributed, so they only encode unimodal distributions. During inference, obstacles are included in the scene, and a return function that includes a negative collision cost and a KL divergence penalty between the current trajectory distribution and the demonstrations is used. The Gaussian parametrization allows for computing the posterior distribution in closed form by solving the optimization problem using Relative Entropy Policy Search (REPS) [44]. This work only considers collisions between the end-effector and the environment. In CLAMP [11], the authors encoded a trajectory prior distribution of the robot's end-effector position and velocity, which was enough for their tasks, using a Gaussian Process prior. At inference, they use GPMP to solve the motion planning problem. The likelihood factors/costs used are to start close to a given start joint state and to avoid collisions of the robot with the environment. With maximum-a-posterior inference, one solution is found to optimize the likelihood function and penalize deviations from the prior GP distribution. Like [10], this method works for unimodal distributions and produces only one solution and not a posterior distribution. Contrary to these works, we use diffusion models to capture the multimodality of human demonstrations and learn trajectory distributions directly on the robot's joint space. To handle trajectory multimodality, [34] learns an Energy-Based Model (EBM), whose inputs are the current state and a phase variable, and the output represents the density of the state distribution. To generate a trajectory from this model, a cost function is built to minimize the energy of the single states across increasing phase values while minimizing the norm between adjacent states. At inference, new costs are formulated and added as energy functions. The posterior distribution is obtained in the form of particles by sampling from the EBM using stochastic optimization. In contrast to incorporating the prior as a cost, we model the full trajectory using diffusion, and optimization is done with gradient-based approaches. With our formulation, we can directly sample trajectories from the

posterior by following the reverse diffusion sampling process, contrasting to sampling from the optimized proposal distribution, which commonly fails because sampling trajectories is hard for EBMs [20].

C. Diffusion models in robotics and motion planning

In recent years, several works have explored score-based and diffusion models in robotics. They are commonly used as expressive multimodal generative model priors for downstream tasks, as similarly done in image generation [45]. At task planning levels, [46] proposed DALL-E-BOT, which, given a set of objects, generates a text description from a scene image and then prompts the text-to-image generator DALL-E [47] to generate a "proper goal scene". [48] proposes *StructDiffusion* for arranging objects based on language commands by predicting the object arrangements using a language-conditioned diffusion model. For grasping, [49] used denoising score matching to learn a generative grasp pose model for a parallel gripper in SE(3) to optimize motion and grasping poses jointly. Several works have also used diffusion models for generating grasps using dexterous grippers [50], [51]. In trajectory planning, [52] introduced Diffuser, a trajectory generative model used for planning in Offline RL and long-horizon tasks. This work introduced a U-Net deep learning architecture to encode trajectories, which is the basis for many follow-up works that model trajectory distributions with diffusion models. MPD builds on the ideas of Diffuser but with a focus on robot motion planning. Namely, we use a B-spline trajectory representation instead of waypoints, leading to faster inference and smooth trajectories; and at inference time, we add new obstacles not seen during training and generate collision-free trajectories using the environment's signed distance function as a cost function. In behavior cloning and visuomotor imitation learning from human demonstrations, [53] introduced diffusion policy, a method that learns policies that take as input a history of visual observations and outputs a sequence of actions. The action distribution is modeled with a diffusion model. Concurrently, [54] introduced BESO, which has a similar structure to diffusion policy, except they use a different formulation of score-based models. These works focused on closed-loop reactive policies that generate short-horizon trajectories, while we focus on generating open-loop point-to-point collision-free trajectories. Recent methods have been built on top of these two works to include better perception, language commands, collision avoidance (for dynamic environments), equivariance and define subgoals for an MPC controller [55]–[62]. In these works, trajectories are typically encoded using waypoints, but in [63] the authors introduced Movement Primitive Diffusion, which instead generates the parameters of a Probabilistic Dynamic Movement Primitive (ProDMP) [64], guaranteeing smoothness in the predict trajectories. Similarly, we predict the control points of a B-spline.

In motion planning for robot manipulators, [24] introduced the first version of Motion Planning Diffusion. Given motion planning trajectories obtained from an expert planner, this method learns a diffusion model as a prior distribution over trajectories. Then, at inference, when new obstacles are added to the scene, MPD samples trajectories from the posterior distribution that is formed with the diffusion prior and a likelihood function that includes collision costs and trajectory smoothening, among others, using classifier guided diffusion [23]. Since the trajectories are modeled using dense waypoints, an additional cost function (a Gaussian Process trajectory prior) is needed to ensure smooth solutions. In this work, we encode the trajectory using B-spline coefficients, which have a smaller dimension than the dense waypoints and ensure smoothness by construction. EDMP [65] follows up on the work of [24] by using an ensemble of collision cost functions instead of a single one. [66] used diffusion trajectory models as priors to sample good initializations for solving downstream constrained trajectory optimization problems. In our work, we show that first sampling from the prior and then only optimizing the cost is less performant than the proposed approach of sampling from the posterior distribution. SafeDiffuser [67] uses control barrier functions to extend Diffuser to safety-critical applications. In APEX [68], the idea of integrating sampling and collision avoidance cost guidance using diffusion models is used for bimanual tasks. Some works have also used environment conditioning to build conditional diffusion models that generate multimodal trajectories given an encoding of the environment in the form of object positions and dimensions, point clouds or RGB images [69]–[72]. For proper generalization, these approaches require obtaining multiple scenes and several motion plans, which might take too much effort to generate [73]. Several planning-with-diffusion works can also be found in autonomous driving, quadruped path planning, and UAVs, whose techniques are closely related to robotic manipulators [74]–[79]. For a recent review of generative models for robotics consult [80].

III. ROBOT MOTION PLANNING WITH DIFFUSION MODELS

This section details our method and the necessary background. Figure 2 shows an overview of the inference process.

A. Optimization-based Motion Planning

Starting from an initial joint position $\boldsymbol{q}_{\text{start}} \in \mathbb{R}^d$ with zero velocity and acceleration, we consider the task of generating a smooth joint trajectory that either reaches a goal joint position $\boldsymbol{q}_{\text{goal}} \in \mathbb{R}^d$ or a goal end-effector pose ${}^W\boldsymbol{H}^{EE}_{\text{goal}} \in \text{SE}(3)$, with zero velocity and acceleration. The obtained trajectory must avoid collisions, be smooth, short, and satisfy joint limits. Let $\boldsymbol{q}(0)$ and $\boldsymbol{q}(T) \in \mathbb{R}^d$ be the start and final joint positions of a robot with d degrees-of-freedom, respectively, T a fixed trajectory duration, $\dot{\boldsymbol{q}}$ the velocities, $\ddot{\boldsymbol{q}}$ the accelerations and $\boldsymbol{\tau}(t) = (\boldsymbol{q}(t), \dot{\boldsymbol{q}}(t), \ddot{\boldsymbol{q}}(t))$ a trajectory. This problem is formulated as trajectory optimization:

$$
\begin{aligned}
\min_{\boldsymbol{\tau}} \quad & C_{\text{vel}}(\boldsymbol{\tau}) + C_{\text{acc}}(\boldsymbol{\tau}) \qquad (1)\\
\text{s.t.} \quad & \boldsymbol{q}(0) = \boldsymbol{q}_{\text{start}} \\
& \text{FK}(\boldsymbol{q}(T)) = {}^W\boldsymbol{H}^{EE}_{\text{goal}} \quad (\text{or } \boldsymbol{q}(T) = \boldsymbol{q}_{\text{goal}}) \\
& \dot{\boldsymbol{q}}(0) = \ddot{\boldsymbol{q}}(0) = \dot{\boldsymbol{q}}(T) = \ddot{\boldsymbol{q}}(T) = \boldsymbol{0} \\
& \text{within joint limits} \\
& \text{no collisions},
\end{aligned}
$$

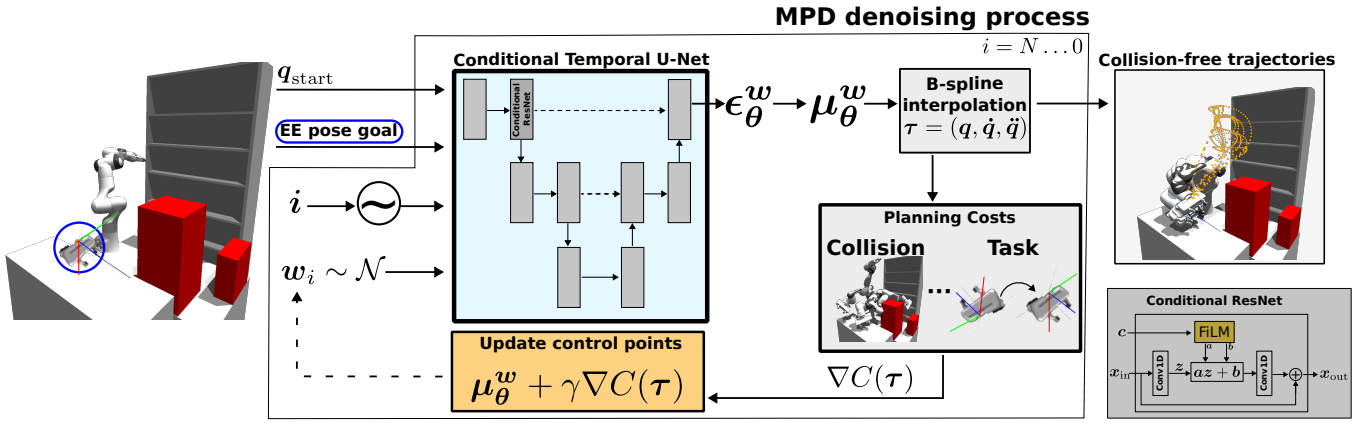


Fig. 2. Overview of inference using Motion Planning Diffusion. An initial joint position $\boldsymbol{q}_{\text{start}}$ and an end-effector goal pose ${}^{W}\boldsymbol{H}_{\text{goal}}^{EE}$ are context variables to a conditional diffusion model. The diffusion model consists of a conditional U-Net architecture (made of conditional ResNet blocks) that takes as input the context, the current time index i and the B-spline control points $\boldsymbol{w}_i$, and outputs the denoising vector $\boldsymbol{\epsilon}_{\boldsymbol{\theta}}$. The control points mean $\boldsymbol{\mu}_i$ are obtained with eq. (6) and used to compute the gradient of a motion planning cost function, which is used to bias the samples towards higher (negative) cost likelihood regions. These particles are injected as input to the U-Net at the next denoising step. In the conditional ResNet block, the time and context embeddings are stacked and used as input for computing the parameters $\boldsymbol{a}$ and $\boldsymbol{b}$ of a FiLM encoder, which are used to transform the embedded control points ($\boldsymbol{x}_{\text{in}}$), computed with a one-dimensional convolution. MPD's output are joint space trajectories that balance the prior distribution and the cost likelihood.

where the costs C_{vel} and C_{acc} promote short and smooth paths, and are detailed in section III-G. In this work we consider a fixed trajectory duration T, similarly to other motion planning algorithms [2], [3], [33]. The optimization of minimum-time trajectories is left for future work.

As the optimization can become unfeasible while trying to satisfy the constraints, it is common practice to relax and include them in the cost function as soft constraints [2], [3], [81]. As we do batch optimization, we can filter the solutions for the ones respecting the constraints. Hence, we solve

$$\arg\min_{\boldsymbol{\tau}} \sum_j \lambda_j C_j(\boldsymbol{\tau}), \tag{2}$$

where λ_j are positive weights to balance the costs. These costs are detailed in section III-G.

B. Motion Planning as Inference

Solving eq. (1) with gradient-based methods relies on an initial trajectory. In some cases, we would like the solution to remain close to the initialization to keep properties such as smoothness or if the initial trajectory is derived from human demonstrations. Then, instead of a single particle, it is useful to describe a prior distribution over trajectories. Hence, the motion planning problem can be formulated as inference [82], [83]. The goal is to either sample from or maximize the posterior distribution of trajectories given the task objective

$$p(\boldsymbol{\tau}|\mathcal{O}) \propto p(\mathcal{O}|\boldsymbol{\tau}) p(\boldsymbol{\tau})^{\lambda_{\text{prior}}}, \tag{3}$$

where $p(\boldsymbol{\tau})$ is a prior over trajectories with temperature λ_{prior}, and $p(\mathcal{O}|\boldsymbol{\tau})$ is the likelihood of achieving the task objectives. A common assumption is that the likelihood factorizes as [34]

$$p(\mathcal{O}|\boldsymbol{\tau}) \propto \prod_j p_j(\mathcal{O}_j|\boldsymbol{\tau})^{\lambda_j} \tag{4}$$

with $\lambda_j > 0$. The costs relate to distributions by $p_j(\mathcal{O}_j|\boldsymbol{\tau}) \propto \exp(-C_j(\boldsymbol{\tau}))$. Then, performing Maximum-a-Posteriori (MAP) on the trajectory posterior

$$\begin{aligned}
&\arg\max_{\boldsymbol{\tau}} \log p(\mathcal{O}|\boldsymbol{\tau}) p(\boldsymbol{\tau})^{\lambda_{\text{prior}}} \\
&= \arg\max_{\boldsymbol{\tau}} \sum_j \log \exp(-C_j(\boldsymbol{\tau}))^{\lambda_j} + \log p(\boldsymbol{\tau})^{\lambda_{\text{prior}}} \\
&= \arg\min_{\boldsymbol{\tau}} \sum_j \lambda_j C_j(\boldsymbol{\tau}) - \lambda_{\text{prior}} \log p(\boldsymbol{\tau})
\end{aligned}$$

is equivalent to eq. (2) regularized with the prior. Contrary to classical optimization-based motion planning, planning-as-inference has several advantages. Notably, it provides a principled way to introduce informative priors to planning problems, e.g., GPMP2 [3] utilizes a GP to encode dynamic feasibility and smoothness. Additionally, specialized methods can be employed to sample from the posterior. We use diffusion models as a prior over trajectories, and instead of computing the MAP solution, sample from the posterior to obtain a trajectory distribution. Sampling from the posterior was done previously for GPMP with Gaussian approximations using variational inference, which are inherently unimodal [84]. Instead, the diffusion framework provides an approach to obtain a multimodal distribution, which allows for a more expressive posterior and a natural way to treat stochasticity.

C. Diffusion Models for Trajectory Priors

Generative modeling methods, such as Generative Adversarial Networks (GAN) [15] and Variational Auto Encoders (VAE) [14] are trained to maximize the data log-likelihood, and sampling is done by applying deterministic transformations to a random variate of an easy-to-sample distribution. Instead, diffusion models [17], [85] perturb the original data distribution $p_{\text{data}}(\boldsymbol{x})$ via a diffusion process to obtain noise, and learn to reconstruct it by denoising using the score-function $\boldsymbol{\nabla}_{\boldsymbol{x}} \log p_{\text{data}}(\boldsymbol{x})$. In this work, we use Denoising Diffusion Probabilistic Models (DDPM) [17].

We first consider an unconditional diffusion model. Let $\boldsymbol{\tau}_0$ be a sample from the data distribution $\boldsymbol{\tau}_0 \sim q(\boldsymbol{\tau}_0)$, which

is transformed into Gaussian noise by a Markovian forward diffusion process $q(\boldsymbol{\tau}_{0:N}) = q(\boldsymbol{\tau}_0)\prod_{i=1}^{N} q(\boldsymbol{\tau}_i|\boldsymbol{\tau}_{i-1}, i)$ with

$$q(\boldsymbol{\tau}_i|\boldsymbol{\tau}_{i-1}, i) = \mathcal{N}\left(\boldsymbol{\tau}_i; \sqrt{1-\beta_i}\boldsymbol{\tau}_{i-1}, \beta_i \boldsymbol{I}\right)$$
$$q(\boldsymbol{\tau}_N) \approx \mathcal{N}\left(\boldsymbol{\tau}_N; \boldsymbol{0}, \boldsymbol{I}\right),$$

where $i = 1, \dots, N$ is the diffusion time step, N is the number of diffusion steps, and β_i is the noise scale at time step i. Common schedules for β are linear and cosine [17], [86].

The denoising process transforms noise back to the data distribution such that $p(\boldsymbol{\tau}_0) \approx q(\boldsymbol{\tau}_0)$

$$p(\boldsymbol{\tau}_{0:N}) = p(\boldsymbol{\tau}_N)\prod_{i=1}^{N} p(\boldsymbol{\tau}_{i-1}|\boldsymbol{\tau}_i, i), \quad p(\boldsymbol{\tau}_N) = \mathcal{N}\left(\boldsymbol{\tau}_N; \boldsymbol{0}, \boldsymbol{I}\right)$$

Sampling from $p(\boldsymbol{\tau}_0)$ is done by first sampling from an isotropic Gaussian and sequentially sampling from the posterior distribution $p(\boldsymbol{\tau}_{i-1}|\boldsymbol{\tau}_i, i)$. For $N \to \infty$ and $\beta_i \to 0$, this posterior converges to a Gaussian distribution [87]. Hence, during training, the goal is to learn to approximate a Gaussian posterior with parameters $\boldsymbol{\theta}$ such that

$$p_{\boldsymbol{\theta}}(\boldsymbol{\tau}_{i-1}|\boldsymbol{\tau}_i, i) = \mathcal{N}\left(\boldsymbol{\tau}_{i-1}; \boldsymbol{\mu}_i = \boldsymbol{\mu}_{\boldsymbol{\theta}}(\boldsymbol{\tau}_i, i), \boldsymbol{\Sigma}_i\right) \approx p(\boldsymbol{\tau}_{i-1}|\boldsymbol{\tau}_i, i) \tag{5}$$

For simplicity, only the mean is learned, and the covariance is set to $\boldsymbol{\Sigma}_i = \sigma_i^2 \boldsymbol{I} = \tilde{\beta}_i \boldsymbol{I}$, with $\tilde{\beta}_i = (1-\bar{\alpha}_{i-1})/(1-\bar{\alpha}_i)\beta_i$, $\alpha_i = 1 - \beta_i$ and $\bar{\alpha}_i = \prod_{k=0}^{i} \alpha_k$ [17]. Instead of learning the posterior mean, we learn the noise vector $\boldsymbol{\epsilon}$, since

$$\boldsymbol{\mu}_{\boldsymbol{\theta}}(\boldsymbol{\tau}_i, i) = \frac{1}{\sqrt{\alpha_i}}\left(\boldsymbol{\tau}_i - \frac{1-\alpha_i}{\sqrt{1-\bar{\alpha}_i}}\boldsymbol{\epsilon}_{\boldsymbol{\theta}}(\boldsymbol{\tau}_i, i)\right). \tag{6}$$

$\boldsymbol{\epsilon}_{\boldsymbol{\theta}}$ is typically implemented using a neural network, whose architecture we detail in section III-F.

The network parameters are learned by maximizing a lower bound on the expected data log-likelihood $\mathbb{E}_{\boldsymbol{\tau}_0 \sim p_{\text{data}}}\left[\log p_{\boldsymbol{\theta}}(\boldsymbol{\tau}_0)\right]$. After simplifying, we minimize

$$\mathcal{L}(\boldsymbol{\theta}) = \mathbb{E}_{i, \boldsymbol{\epsilon}, \boldsymbol{\tau}_0}\left[\|\boldsymbol{\epsilon} - \boldsymbol{\epsilon}_{\boldsymbol{\theta}}(\boldsymbol{\tau}_i, i)\|_2^2\right] \tag{7}$$
$$i \sim \mathcal{U}(1, N),\ \boldsymbol{\epsilon} \sim \mathcal{N}\left(\boldsymbol{0}, \boldsymbol{I}\right),\ \boldsymbol{\tau}_0 \sim q(\boldsymbol{\tau}_0),\ \boldsymbol{\tau}_i \sim q(\boldsymbol{\tau}_i|\boldsymbol{\tau}_0, i).$$

For a complete derivation, we refer the readers to [88]. Due to the Markov property, and given the data is in Euclidean space, the distribution of the noisy trajectory at time step i given the original one is Gaussian and can be written in closed form as $q(\boldsymbol{\tau}_i|\boldsymbol{\tau}_0, i) = \mathcal{N}\left(\boldsymbol{\tau}_i; \sqrt{\bar{\alpha}_i}\boldsymbol{\tau}_0, (1-\bar{\alpha}_i)\boldsymbol{I}\right)$. This allows for efficient training by sampling from $q(\boldsymbol{\tau}_i|\boldsymbol{\tau}_0, i)$ without running the forward diffusion process. Intuitively, eq. (7) states that the goal of training is to learn a denoising network that removes the noise added to the data sample at step i.

At inference, we obtain a sample $\tilde{\boldsymbol{\tau}}_0 \sim p_{\boldsymbol{\theta}}(\boldsymbol{\tau}_0)$ through a series of denoising steps from eq. (5). In practice, to control stochasticity, we also pre-multiply the posterior variance, resulting in $\alpha\boldsymbol{\Sigma}_i$ [89], with $\alpha \in [0, 1]$. $\alpha = 0$ means no noise and $\alpha = 1$ is the DDPM sampling algorithm. This is particularly important in our work because it lowers the chance that trajectories that are already collision-free come into collision due to large noise values.

To generate a trajectory distribution based on a context $\boldsymbol{c}$, e.g., a start joint position and a desired end-effector pose, we extend to a conditional diffusion model $p_0(\boldsymbol{\tau}|\boldsymbol{c})$, and thus $p(\boldsymbol{\tau}_{0:N}|\boldsymbol{c}) = p(\boldsymbol{\tau}_N)\prod_{i=1}^{N} p(\boldsymbol{\tau}_{i-1}|\boldsymbol{\tau}_i, \boldsymbol{c})$. The only change to the above is that now we learn a noise model with an additional $\boldsymbol{c}$ input $\boldsymbol{\epsilon}_{\boldsymbol{\theta}}(\boldsymbol{\tau}_i, i, \boldsymbol{c})$. The expectation in eq. (7) is modified to include the context distribution $\boldsymbol{c} \sim p(\boldsymbol{c})$ and $\boldsymbol{\tau}_0 \sim q(\boldsymbol{\tau}_0|\boldsymbol{c})$.

D. Blending Sampling and Optimization

With a prior trajectory diffusion model and an optimality variable $\mathcal{O}$ (e.g., representing collision avoidance), our goal is to sample from the posterior distribution

$$p(\boldsymbol{\tau}_0|\mathcal{O}) \propto p(\mathcal{O}|\boldsymbol{\tau}_0)p(\boldsymbol{\tau}_0). \tag{8}$$

To show that this is equivalent to sampling from the prior while biasing the trajectories towards the high-likelihood regions, for completeness, we replicate here the proof from [23]. By definition of the Markovian reverse diffusion

$$p(\boldsymbol{\tau}_0|\mathcal{O}) = \int p(\boldsymbol{\tau}_N|\mathcal{O})\prod_{i=1}^{N} p(\boldsymbol{\tau}_{i-1}|\boldsymbol{\tau}_i, i, \mathcal{O})\mathrm{d}\boldsymbol{\tau}_{1:N}, \tag{9}$$

where $p(\boldsymbol{\tau}_N|\mathcal{O})$ is standard Gaussian noise by definition. Hence, to sample from $p(\boldsymbol{\tau}_0|\mathcal{O})$, we iteratively sample from the objective-conditioned posterior using Bayes' law

$$p(\boldsymbol{\tau}_{i-1}|\boldsymbol{\tau}_i, i, \mathcal{O}) \propto p(\mathcal{O}|\boldsymbol{\tau}_{i-1})p(\boldsymbol{\tau}_{i-1}|\boldsymbol{\tau}_i, i), \tag{10}$$

where $p(\boldsymbol{\tau}_{i-1}|\boldsymbol{\tau}_i, i) \approx p_{\boldsymbol{\theta}}(\boldsymbol{\tau}_{i-1}|\boldsymbol{\tau}_i, i)$ is the approximate learned prior, and $p(\mathcal{O}|\boldsymbol{\tau}_{i-1}) = p(\mathcal{O}|\boldsymbol{\tau}_{i-1}, \boldsymbol{\tau}_i, i)$, since due to the Markov property in diffusion, $\mathcal{O}$ and $\boldsymbol{\tau}_i$ are conditionally independent given $\boldsymbol{\tau}_{i-1}$. The objective-conditioned posterior cannot be sampled in closed-form, but given the learned denoising prior model over trajectories is Gaussian, its logarithm equates to

$$\begin{aligned}\log p_{\boldsymbol{\theta}}(\boldsymbol{\tau}_{i-1}|\boldsymbol{\tau}_i, i) &= \log \mathcal{N}\left(\boldsymbol{\tau}_i; \boldsymbol{\mu}_i = \boldsymbol{\mu}_{\boldsymbol{\theta}}(\boldsymbol{\tau}_i, i), \boldsymbol{\Sigma}_i\right) \\ &\propto -\frac{1}{2}(\boldsymbol{\tau}_{i-1} - \boldsymbol{\mu}_i)^\top \boldsymbol{\Sigma}_i^{-1}(\boldsymbol{\tau}_{i-1} - \boldsymbol{\mu}_i).\end{aligned} \tag{11}$$

By definition of the noise schedule β, as the denoising step approaches zero, so does the noise covariance $\lim_{i\to 0}\|\boldsymbol{\Sigma}_i\| = 0$. Therefore, $p_{\boldsymbol{\theta}}(\boldsymbol{\tau}_{i-1}|\boldsymbol{\tau}_i, i)$ concentrates its mass close to the mean $\boldsymbol{\mu}_i$, and the task log-likelihood is approximated with a first-order Taylor expansion around $\boldsymbol{\mu}_i$

$$\log p(\mathcal{O}|\boldsymbol{\tau}_{i-1}) \approx \log p(\mathcal{O}|\boldsymbol{\tau}_{i-1} = \boldsymbol{\mu}_i) + (\boldsymbol{\tau}_{i-1} - \boldsymbol{\mu}_i)^\top \boldsymbol{g} \tag{12}$$
$$\text{with} \quad \boldsymbol{g} = \boldsymbol{\nabla}_{\boldsymbol{\tau}_{i-1}} \log p(\mathcal{O}|\boldsymbol{\tau}_{i-1} = \boldsymbol{\mu}_i).$$

Combining eq. (11) and eq. (12) we obtain

$$\begin{aligned}&\log p(\boldsymbol{\tau}_{i-1}|\boldsymbol{\tau}_i, i, \mathcal{O}) \\ &\propto -\frac{1}{2}(\boldsymbol{\tau}_{i-1} - \boldsymbol{\mu}_i)^\top \boldsymbol{\Sigma}_i^{-1}(\boldsymbol{\tau}_{i-1} - \boldsymbol{\mu}_i) + (\boldsymbol{\tau}_{i-1} - \boldsymbol{\mu}_i)\boldsymbol{g} \\ &\propto -\frac{1}{2}(\boldsymbol{\tau}_{i-1} - \boldsymbol{\mu}_i - \boldsymbol{\Sigma}_i\boldsymbol{g})^\top \boldsymbol{\Sigma}_i^{-1}(\boldsymbol{\tau}_{i-1} - \boldsymbol{\mu}_i - \boldsymbol{\Sigma}_i\boldsymbol{g})\end{aligned} \tag{13}$$

$$= \log p(\boldsymbol{z} = \boldsymbol{\tau}_{i-1}), \quad p(\boldsymbol{z}) = \mathcal{N}(\boldsymbol{z}; \underbrace{\boldsymbol{\mu}_i + \boldsymbol{\Sigma}_i\boldsymbol{g}}_{\boldsymbol{\mu}_z}, \boldsymbol{\Sigma}_i). \tag{14}$$

In motion planning-as-inference we have from eq. (4)

$$\boldsymbol{g} = \boldsymbol{\nabla}_{\boldsymbol{\tau}_{i-1}} \log p(\mathcal{O}|\boldsymbol{\tau}_{i-1}) = -\sum_j \lambda_j \boldsymbol{\nabla}_{\boldsymbol{\tau}_{i-1}} C_j(\boldsymbol{\tau}_{i-1} = \boldsymbol{\mu}_i),$$

where the costs are assumed to be differentiable w.r.t. the trajectory. Hence, sampling from the task-conditioned posterior is equivalent to sampling from a Gaussian distribution with mean and covariance as $p(\boldsymbol{z})$. At every denoising step, we sample from the prior, move the particle toward low-cost regions, and repeat this process until $i = 0$. This formulation benefits us by allowing us to use gradient-based techniques to guide the sampling process while running the denoising process for the prior. The updated mean of $\boldsymbol{z}$ in eq. (14) can be interpreted as taking a single gradient step starting from $\boldsymbol{\mu}_i$, but several aspects are crucial for making guided sampling work in practice [90].

First, the entries in $\boldsymbol{\Sigma}_i$ approach 0 with $i \to 0$. Hence, the guidance function vanishes towards the end of the denoising process. This is desirable if the prior covers a very large space of the data manifold where the objective likelihood is high. However, if there is no (or little) data in those regions, the generative model might be unable to move samples towards high objective likelihood regions. To counteract this and keep the influence of the task likelihood, we drop the covariance $\boldsymbol{\Sigma}_i$. This is equivalent to changing the cost weights per denoising step $\lambda_j \coloneqq \boldsymbol{\Sigma}_i^{-1}\lambda_j$, which keeps the likelihood relevancy.

Second, instead of performing one gradient step, we take M gradient steps and choose the step size such that the optimized mean of $\boldsymbol{z}$ is within a small deviation δ of the prior mean as motivated in [91]. This is equivalent to solving

$$\min_{\boldsymbol{\mu}_z} \quad \sum_j \lambda_j C_j(\boldsymbol{\mu}_z), \quad \text{s.t. } |\boldsymbol{\mu}_z - \boldsymbol{\mu}_i| \leq \delta, \tag{15}$$

using first-order gradients starting from $\boldsymbol{\mu}_i$, and δ prevents the optimization from moving *too* far from the prior, which could lead to a distribution shift.

Third, recall from eq. (3) that we can control the influence of the prior by sampling from $p(\boldsymbol{\tau}_0|\mathcal{O}) \propto p(\mathcal{O}|\boldsymbol{\tau}_0)p(\boldsymbol{\tau}_0)^{\lambda_{\text{prior}}}$. In image generation, typically $\lambda_{\text{prior}} = 1$, and the classifier weight λ_j is increased to generate images belonging to a desired class. This balance works well because image models are trained with large amounts of data, and the desired class is covered in the prior distribution. For instance, when generating images of dogs, we steer particles towards regions of the prior that include dogs. In motion planning, we want to obtain collision-free trajectories in *new environments*. This means our prior might not have samples in the new collision-free region. Hence, we treat λ_{prior} as a hyperparameter to control the influence of the prior. From the relation between the score function and the predicted noise we have [18]

$$\begin{aligned}\boldsymbol{\nabla}_{\boldsymbol{\tau}_i} \log p(\boldsymbol{\tau}_i) &= -\frac{1}{\sqrt{1-\bar{\alpha}_i}}\boldsymbol{\epsilon}(\boldsymbol{\tau}_i, i) \propto \boldsymbol{\epsilon}(\boldsymbol{\tau}_i, i) \\ \boldsymbol{\nabla}_{\boldsymbol{\tau}_i} \log p(\boldsymbol{\tau}_i)^{\lambda_{\text{prior}}} &= \lambda_{\text{prior}} \boldsymbol{\nabla}_{\boldsymbol{\tau}_i} \log p(\boldsymbol{\tau}_i) \propto \lambda_{\text{prior}} \boldsymbol{\epsilon}(\boldsymbol{\tau}_i, i).\end{aligned} \tag{16}$$

Therefore, we replace $\boldsymbol{\epsilon}_{\boldsymbol{\theta}}(\boldsymbol{\tau}_i, i)$ in eq. (6) with $\lambda_{\text{prior}}\boldsymbol{\epsilon}_{\boldsymbol{\theta}}(\boldsymbol{\tau}_i, i)$.

Accelerated sampling. A standard sample method uses the same number of time steps as the ones from training, which leads to a slow sampling process. However, the initial steps of the denoising process include regions with large noise levels, and it was shown that several steps can be discarded by making the diffusion model non-Markovian, using Denoising Diffusion Implicit Models (DDIM) [92]. Instead of sampling with N steps as in DDPM, we select a subset of time indices $i \in M = \{1, \dots, N\}$, with typically $|M| \ll N$. To adapt DDIM to cost guidance, to sample from the posterior in eq. (10), we make use of eq. (16) and write

$$\begin{aligned}&\boldsymbol{\nabla}_{\boldsymbol{\tau}_{i-1}} \log p(\boldsymbol{\tau}_{i-1}|\boldsymbol{\tau}_i, i, \mathcal{O}) && (17)\\ &\propto \boldsymbol{\nabla}_{\boldsymbol{\tau}_{i-1}} \log p(\mathcal{O}|\boldsymbol{\tau}_{i-1}) + \boldsymbol{\nabla}_{\boldsymbol{\tau}_{i-1}} \log p(\boldsymbol{\tau}_{i-1}|\boldsymbol{\tau}_i, i) && (18)\\ &= -\frac{1}{\sqrt{1-\bar{\alpha}_i}}\boldsymbol{\epsilon}(\boldsymbol{\tau}_i, i) + \boldsymbol{\nabla}_{\boldsymbol{\tau}_{i-1}} \log p(\mathcal{O}|\boldsymbol{\tau}_{i-1}) \\ &= -\frac{1}{\sqrt{1-\bar{\alpha}_i}} \underbrace{\left(\boldsymbol{\epsilon}(\boldsymbol{\tau}_i, i) - \sqrt{1-\bar{\alpha}_i}\boldsymbol{\nabla}_{\boldsymbol{\tau}_{i-1}} \log p(\mathcal{O}|\boldsymbol{\tau}_{i-1})\right)}_{=\boldsymbol{\epsilon}},\end{aligned}$$

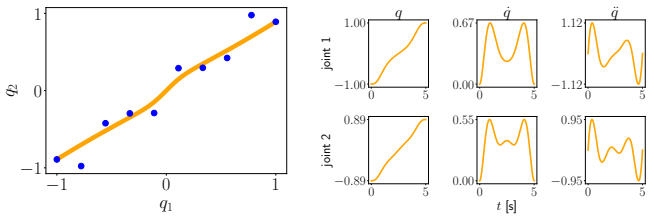
and replace $\boldsymbol{\epsilon}$ in the original DDIM equations [23]. In practice, we note that, similarly to DDPM, $\sqrt{1-\bar{\alpha}_i} \to 0$ with $i \to 0$, and therefore remove this term to maintain the influence of the cost function by adjusting the weights $\lambda_j \coloneqq (\sqrt{1-\bar{\alpha}_i})^{-1}\lambda_j$.

A note on classifier-free guidance. An orthogonal approach to introduce guidance in diffusion models is to use classifier-free guidance [89]. This method works by embedding information from the objective function (or constraints) into a context variable. This is not as flexible since the constraints need to be known during training. On the contrary, cost guidance can easily integrate new constraints via differentiable cost functions, which makes it easier to adapt to new problems.

E. Trajectory Parametrization

Until this point we did not specify how the trajectory $\boldsymbol{\tau}$ is parametrized. In previous work [24], the diffusion prior over trajectories was represented as a time-discretized vector of joint positions $\boldsymbol{\tau} = [\boldsymbol{q}_0, \dots, \boldsymbol{q}_{H-1}] \in \mathbb{R}^{H \times d}$, where H is the number of waypoints. Given a planning frequency $1/\Delta t$, the trajectory duration is $T = H\Delta t$. On the positive side, the waypoint representation ensures the trajectory passes on the waypoints, which is useful when precise path-following is required, especially for tasks that require strict position control. Waypoints can be placed freely, making them suitable for representing paths with sharp turns or discontinuities. However, velocities and accelerations were computed with finite differences, which did not provide smoothness guarantees. A cost function/regularizer (the GP prior cost) was needed to enforce smoothness, which needed extra hyperparameter tuning. Moreover, if part of a trajectory is in collision and the corresponding waypoints are moved, then an already smooth trajectory might become non-smooth, so a balance between the two costs needs to be accounted for.

In turn, in this work, we propose using a B-spline parametrization instead [93]. There are several properties of B-splines that are interesting for our application. As B-splines are smooth and have continuous derivatives, if part of the trajectory that is in collision is moved, then the overall trajectory remains smooth. This representation has a locality property, which means that modifying control points only affects neighbor trajectory segments. Additionally, as denoising is computationally expensive due to the iterative denoising process, a lower-dimensional parametrization with fewer parameters than $H \times d$ is desirable. In B-splines, the



(a) Control points and curve (b) Trajectories in time

Fig. 3. The figures show the resulting trajectories when using linear phase-time scaling. In (a), the control points (blue dots) and path of a clamped 5th order B-spline are displayed. (b) shows the trajectory in time of positions, velocities, and accelerations for the two degrees of freedom.

parametrization is $n_b \times d$, where n_b is the number of control points, and $n_b \ll H$. For these reasons, we choose B-splines as a trajectory representation.

A B-spline joint position trajectory as a function of a phase variable $s \in [0, 1]$ is defined as a linear combination of n_b basis-splines [94]

$$\boldsymbol{q}(s) = \sum_{i=0}^{n_b-1} B_{i,p}^{\boldsymbol{u}}(s)\boldsymbol{w}_i = \boldsymbol{B}(s)\boldsymbol{w}, \tag{19}$$

where $B_{i,p}^{\boldsymbol{u}} \in \mathbb{R}$ is a basis-spline of degree $p \in \mathbb{Z}^+$ with support between knots u_i and u_{i+p+1} defined in a knot vector $\boldsymbol{u} = [u_0, \dots, u_m]$, with $m \geq n_b - p - 1$ and non-decreasing entries, $\boldsymbol{B}(s) = [B_{0,p}^{\boldsymbol{u}}(s) \dots B_{n_b-1,p}^{\boldsymbol{u}}(s)] \in \mathbb{R}^{1\times n_b}$, $\boldsymbol{w}_i \in \mathbb{R}^{1\times d}$ and $\boldsymbol{w} = [\boldsymbol{w}_0 \dots \boldsymbol{w}_{n_b-1}]^\top \in \mathbb{R}^{n_b\times d}$ are the B-spline control points (or coefficients). Note that with a fixed basis $\boldsymbol{B}_p$, the control points are the degrees of freedom and completely define the B-spline. We learn a diffusion model over $\boldsymbol{w}$. The basis functions are defined recursively with De Boor's algorithm [94]. With $s \in [0, 1]$, we limit the knots vector to be in the same range, and to distribute the basis functions evenly, we set subsequent knots to be equidistant [93]

$$\boldsymbol{u} = \Big[\underbrace{0 \dots 0}_{p+1 \text{ times}} \underbrace{\frac{1}{n_b - p} \dots \frac{n_b - p - 1}{n_b - p}}_{n_b-p-1 \text{ elements}} \underbrace{1 \dots 1}_{p+1 \text{ times}}\Big], \tag{20}$$

such that $|\boldsymbol{u}| = n_b + p + 1$. To ensure the boundary constraints on start and final positions, zero velocities, and accelerations as in eq. (1), we repeat boundary knots (see eq. (20) and set the first and last control points as [95]

$$\boldsymbol{w}_0 = \boldsymbol{w}_1 = \boldsymbol{w}_2 = \boldsymbol{q}_{\text{start}} \tag{21}$$

$$\boldsymbol{w}_{n_b-1} = \boldsymbol{w}_{n_b-2} = \boldsymbol{w}_{n_b-3} = \boldsymbol{q}_{\text{goal}}. \tag{22}$$

Note that if $\boldsymbol{q}_{\text{goal}}$ is not specified, but rather a desired end-effector pose ${}^{W}\boldsymbol{H}_{\text{goal}}^{EE}$, then the last control point $\boldsymbol{w}_{n_b-1}$ is generated by the diffusion model. To accelerate computations, similarly to [93], we discretize the phase variable into n_s steps and pre-compute a matrix $\boldsymbol{B} \in \mathbb{R}^{n_s\times n_b}$ of B-splines basis, where n_s is equivalent to H in the dense waypoint representation. A joint position trajectory in phase-space is given by $\boldsymbol{Q} = \boldsymbol{B}\boldsymbol{w} \in \mathbb{R}^{n_s\times d}$.

A trajectory in time is obtained by transforming the phase variable. Let f be a monotonically increasing function such that $t = f(s)$, with $f(0) = 0$ and $f(1) = T$, where T is the trajectory duration. Hence, we have $\boldsymbol{q}(t) = \boldsymbol{q}(f(s))$. Therefore, the position trajectory is completely defined by the B-spline in phase space, and with slight notation abuse, we write $\boldsymbol{q}(t) \triangleq \boldsymbol{q}(s)$. Let the derivative of the phase w.r.t. time be a function of the phase variable [93], [96]

$$r(s) = \frac{\mathrm{d}s}{\mathrm{d}t}(s) = \left(\frac{\mathrm{d}t}{\mathrm{d}s}\right)^{-1}(s). \tag{23}$$

The first and second derivatives of $\boldsymbol{q}$ w.r.t. time, velocity and acceleration, respectively, can be computed as

$$\begin{aligned}\dot{\boldsymbol{q}}(t) &= \frac{\mathrm{d}\boldsymbol{q}(t)}{\mathrm{d}t} = \frac{\partial \boldsymbol{q}(s)}{\partial s}\frac{\mathrm{d}s}{\mathrm{d}t} = \frac{\partial \boldsymbol{q}(s)}{\partial s} r(s) \\ \ddot{\boldsymbol{q}}(t) &= \frac{\mathrm{d}^2\boldsymbol{q}(t)}{\mathrm{d}t^2} = \frac{\mathrm{d}}{\mathrm{d}t}\left(\frac{\partial \boldsymbol{q}(s)}{\partial s}\right) r(s) + \frac{\partial \boldsymbol{q}(s)}{\partial s}\frac{\mathrm{d}}{\mathrm{d}t} r(s) \\ &= \frac{\partial^2 \boldsymbol{q}(s)}{\partial s^2}\left(r(s)\right)^2 + \frac{\partial \boldsymbol{q}(s)}{\partial s}\frac{\partial r(s)}{\partial s} r(s)\end{aligned} \tag{24}$$

These expressions compute the trajectory in time using the phase variable and their derivative relation. For completeness, we must compute the derivatives of B-spline curves w.r.t. the phase variable and define the phase-time function.

The k-th order derivative $\partial \boldsymbol{q}(s)^{(k)}/\partial s^{(k)}$ is also a B-spline [95], [97], which can be written as

$$\frac{\partial \boldsymbol{q}}{\partial s}(s) = \sum_{i=0}^{n_b-2} B'^{\boldsymbol{u}}_{i,p}(s)\boldsymbol{w}_i \tag{25}$$

$$B'^{\boldsymbol{u}}_{i,p}(s) \triangleq p\left(\frac{B_{i,p-1}(s)}{u_{i+p} - u_i} - \frac{B_{i+1,p-1}(s)}{u_{i+p+1} - u_{i+1}}\right). \tag{26}$$

Note that $B'^{\boldsymbol{u}}_{i,p}$ is not a proper basis. Higher order derivatives can be computed similarly $\partial^2\boldsymbol{q}/\partial s^2 = \sum_{i=0}^{n_b-3} B''^{\boldsymbol{u}}_{i,p}(s)\boldsymbol{w}_i$, and the new basis matrices can be pre-computed for a fixed number of steps n_s.

One of the benefits of phase-time decoupling is being able to speed up and slow down movements. As common in the literature [98], we assume a linear relation, but nonlinear functions can also be used [93]

$$\begin{aligned} s = f^{-1}(t) = t/T, \quad r(s) = 1/T, \quad \frac{\partial r(s)}{\partial s} = 0 \\ \dot{\boldsymbol{q}}(t) = \frac{\partial \boldsymbol{q}(s)}{\partial s}\frac{1}{T}, \quad \ddot{\boldsymbol{q}}(t) = \frac{\partial^2 \boldsymbol{q}(s)}{\partial s^2}\frac{1}{T^2}. \end{aligned}$$

Figure 3 illustrates the properties of the B-spline trajectory representation in 2-dimensional example. This parametrization allows for a low-dimensional representation of smooth trajectories, which are useful for motion planning.

Other trajectory representations The definition in eq. (19) encompasses other types of movement primitives by modifying the basis functions, such as Bézier curves [99], Probabilistic Movement Primitives (ProMP) [98] or Probabilistic Dynamic Movement Primitives (ProDMP) [64]. However, some properties of B-splines are suited to our tasks. Unlike other spline methods such as Bézier curves, the B-spline has a *local property* instead of the global property [99]. This property can be important because the collision cost is local in gradient-based optimization. By moving control points locally, we ensure that only the part of the trajectory that is in collision

is affected. Compared to ProMPs and ProDMPs, B-spline trajectories lie within the convex hull defined by the control points, which allows for the definition of constraints on the coefficients instead of constraints on the whole trajectory. Moreover, commonly, these movement primitives are defined by individual basis functions with support in $\mathbb{R}$, such as an exponential kernel, making it difficult to set boundary conditions for position, velocity, and acceleration [96]. Establishing these constraints is easier with B-splines due to the basis being defined over closed intervals determined by the knots vector.

F. Implementation Details

For learning, we assume having a dataset of pairs of C contexts and N_C (possibly) multimodal trajectories per context $\mathcal{D} = \{\{(\boldsymbol{c}_j, \boldsymbol{\tau}_{jk})\}_{k=1}^{N_C}\}_{j=1}^{C}$. Following section III-C, we learn a prior over trajectories by learning the conditioned denoising model $\boldsymbol{\epsilon}_{\boldsymbol{\theta}}(\boldsymbol{w}_i, i, \boldsymbol{c})$, where $\boldsymbol{w}_i \in \mathbb{R}^{n_b \times d}$ are the control points of a B-spline. To ensure zero velocities and accelerations at the boundaries, the first and last control points are fixed according to eqs. (21) and (22). Hence, the diffusion model uses the inner control points of size $n_b - 6$, or $n_b - 5$ if the end-effector goal pose is used, meaning the last joint configuration is not fixed. The conditioning variable $\boldsymbol{c}$ consists of the current (start) joint position, and if a goal joint position is defined, $\boldsymbol{c} = [\boldsymbol{q}_{\text{start}}, \boldsymbol{q}_{\text{goal}}]$, otherwise, if a desired end-effector goal is provided, then $\boldsymbol{c} = [\boldsymbol{q}_{\text{start}}, {}^{W}\boldsymbol{H}^{EE}_{\text{goal}}]$. The end-effector pose input to the network consists of the position concatenated with a flattened rotation matrix. We separate these two conditioning cases because the latter option is more natural for humans to specify. For instance, in a pick-and-place task, we want the manipulated object to be placed in a certain pose and might not care too much about the final joint positions. Moreover, by specifying a task space pose, we might get multiple robot configurations that achieve the same end-effector pose, which gives the user the freedom to decide which configuration to choose. However, if the user has access to an inverse kinematics solver, for instance, to bias the solutions to a neutral configuration, then she can use conditioning with goal joint positions. Nevertheless, with our framework, biasing the solution could also be achieved by adding another cost function.

[52] proposed a temporal U-Net architecture with 1-dimensional convolutions to encode local relations between states and actions, acting as a low-pass filter that prevents jumps between neighboring states. Due to its success in modeling trajectories, we adapt it and replace states with B-spline control points and extend this network architecture to include a conditioning variable, along with the diffusion step index, using Feature-wise Linear Modulation (FiLM) [100] (see fig. 2). Additionally, all data is normalized to $[-1, 1]$.

For sampling, we tested both DDPM and DDIM algorithms. In DDIM, the number of steps $|M|$ is commonly selected linearly from $\{1, \dots, N\}$. With this schedule, several steps closer to N add too much noise to samples during the denoising process. We can skip those first steps by choosing quadratically distributed steps, which means the denoising process jumps from larger indexes to ones closer to 0, leading to fewer denoising steps. Moreover, we use the deterministic version of DDIM with $\gamma = 0$, which slightly prevents the risk of adding noise to a trajectory that is already collision-free. During inference, cost guidance is computed w.r.t. the $\boldsymbol{\tau}_i$ at denoising step i. However, for higher noise levels, the trajectory is still quite noisy, and this gradient produces little effect. Hence, we only apply the cost gradients on the last i_{cost} steps of the denoising process.

We run our algorithm by sampling and optimizing a batch of trajectories in parallel. Therefore, to sample many trajectories, evaluate their costs, and compute gradients in our framework, we maximize parallelization by implementing all components in PyTorch [101] by leveraging GPU utilization. Although a batch of trajectories is used for planning, in the real world, only one trajectory can be executed. Choosing this trajectory is task-dependent and a user choice. For tasks where the context includes reaching a desired end-effector pose, one could choose the trajectory corresponding to the lowest end-effector pose error, and for other tasks, one could use the minimum length trajectory or other combinations.

G. Motion Planning Costs

In this section, we describe the motion planning costs used. The unconstrained cost function from eq. (2) is computed with the trajectory time-integral costs and can be converted into an integral in phase-space with a change of variables

$$C(\boldsymbol{\tau}) = \sum_j \lambda_j \int_0^T C_j(\boldsymbol{\tau}(t))\mathrm{d}t = \sum_j \lambda_j \int_0^1 C_j(\boldsymbol{\tau}(s))r(s)^{-1}\mathrm{d}s \tag{27}$$

where the integral is approximated by discretizing the phase variable into n_s segments (section III-E).

The costs we consider are computed in joint and task space, and the total derivative w.r.t. the B-spline trajectory parametrization $\boldsymbol{w}$ (the control points) is computed using the chain rule, e.g., for joint position costs

$$\frac{\mathrm{d}C_j(\boldsymbol{q})}{\mathrm{d}\boldsymbol{w}} = \frac{\partial C_j(\boldsymbol{q})}{\partial \boldsymbol{q}} \frac{\mathrm{d}\boldsymbol{q}(s)}{\mathrm{d}\boldsymbol{w}}, \tag{28}$$

where $C_j \in \mathbb{R}$, $\partial C(\boldsymbol{q})/\partial \boldsymbol{q} \in \mathbb{R}^{1\times d}$, and for B-splines $\mathrm{d}\boldsymbol{q}/\mathrm{d}\boldsymbol{w} = \boldsymbol{B} \in \mathbb{R}^{d \times n_b \times d}$, where the basis matrix is repeated along the degrees-of-freedom to match dimensions. If C_j is a task space cost, let $\boldsymbol{X}_m = \mathrm{FK}_m(\boldsymbol{q}) \in \mathrm{SE}(3)$ be the pose of a link m on the robot arm computed with forward kinematics, and consider a cost $C_j(\boldsymbol{X}_m)$. Then we have

$$\frac{\partial C_j(\boldsymbol{q})}{\partial \boldsymbol{q}} = \frac{\partial C_j(\boldsymbol{X}_m)}{\partial \boldsymbol{X}_m} \frac{\partial \boldsymbol{X}_m(\boldsymbol{q})}{\partial \boldsymbol{q}},$$

where $\partial C_j(\boldsymbol{X}_m)/\partial \boldsymbol{X}_m \in \mathbb{R}^{1\times 6} \in \mathfrak{se}(3)$ is a vector in the Lie algebra of SE(3), and

$$\partial \boldsymbol{X}_m(\boldsymbol{q})/\partial \boldsymbol{q} \triangleq \boldsymbol{J}_m(\boldsymbol{q}) = \begin{bmatrix} \boldsymbol{J}^p_m(\boldsymbol{q}) \\ \boldsymbol{J}^o_m(\boldsymbol{q}) \end{bmatrix} \in \mathbb{R}^{6\times d} \tag{29}$$

is the forward kinematics geometric Jacobian at link m, for position p and orientation o. This formulation is important because we do not need to use automatic differentiation for gradient computations, and make use of the library Theseus [102] to compute geometric Jacobians and Lie algebra

quantities without backpropagation through the computational graph. This separation exploits the structure of the Jacobian of kinematic chains, allowing faster and more stable computation, as well as the separation of position and orientation costs. Moreover, due to the B-spline linear parametrization, gradient computation can be easily parallelized in the GPU.

Velocity cost. We minimize the joint space velocity trajectory using a squared penalty, which promotes straight paths by using a squared penalty on the velocity trajectories

$$C_{\text{vel}}(\boldsymbol{\tau}(s)) = \frac{1}{2}\|\boldsymbol{q}'(s)\|_2^2,$$

where $\boldsymbol{q}'(s)$ is the first-order derivative of $\boldsymbol{q}$ w.r.t. s.

Acceleration cost. To prevent sharp turns in the robot trajectory, we maximize smoothness by using a squared penalty on the acceleration trajectories

$$C_{\text{acc}}(\boldsymbol{\tau}(s)) = \frac{1}{2}\|\boldsymbol{q}''(s)\|_2^2,$$

where $\boldsymbol{q}''(s)$ is the second-order derivative of $\boldsymbol{q}$ w.r.t. s.

Task cost. In several manipulation tasks, such as pick-and-place or pouring, it is much more intuitive to specify a desired end-effector goal pose instead of a goal joint position. Hence, we minimize the end-effector pose error obtained at the last joint position ($s = 1$) with

$$C_{\text{task}}(\boldsymbol{\tau}(s))|_{s=1} = d_{\text{SE(3)}}\left({}^{W}\boldsymbol{H}_{\text{goal}}^{EE},\, \text{FK}_{EE}(\boldsymbol{q}(1))\right).$$

$d_{\text{SE(3)}}$ defines the distance between two elements of SE(3). Given two poses (homogeneous transformations) $\boldsymbol{T}_1 = [\boldsymbol{R}_1, \boldsymbol{p}_1] \in \text{SE(3)}$ and $\boldsymbol{T}_2 = [\boldsymbol{R}_2, \boldsymbol{p}_2] \in \text{SE(3)}$, consisting of a translational and rotational part, we choose

$$d_{\text{SE(3)}}(\boldsymbol{T}_1, \boldsymbol{T}_2) = \frac{1}{2}\|\boldsymbol{p}_1 - \boldsymbol{p}_2\|_2^2 + \frac{1}{2}\|\text{LogMap}(\boldsymbol{R}_1^\top \boldsymbol{R}_2)\|_2^2,$$

where $\text{LogMap}(\cdot)$ is the operator that maps an element of the Lie group SO(3) to the vector representation of its tangent space at the identity element, the Lie algebra $\mathfrak{so}(3)$ [103]. With the Jacobian decomposition from eq. (29), we can map the derivatives of $d_{\text{SE(3)}}$ w.r.t. position and orientation task space errors to the joint space.

Collision costs. The task-space signed distance function of an environment $\text{SDF}(\boldsymbol{x})$ is the smallest signed Euclidean distance between a point in space $\boldsymbol{x} \in \mathbb{R}^3$ and the closest surface (negative if inside an obstacle, and positive otherwise). Given an environment with obstacles, we precompute and store the SDF using a fine voxel grid representation and project a continuous point $\boldsymbol{x}$ to the nearest point in the grid. The SDF representation is a common assumption in motion planning [2], [3], which can be obtained fast in the real world using newer software modules [104]. Also, if a new object is added to the scene, and we know its SDF, the resulting SDF is the minimum between the environment and the new object SDF. Similarly, we can precompute the SDF gradient $\nabla_{\boldsymbol{x}}\text{SDF}(\boldsymbol{x})$, which has L2-norm equal to 1. This gradient indicates a direction to push the robot out of collision when the $\text{SDF}(\boldsymbol{x}) < 0$. To compute robot collisions, we represent it with S spheres along its body $\mathcal{B}$ (see fig. 4), allowing for faster computation of the SDF between the robot and the environment compared to using a full mesh. We consider two types of collisions: with the environment and self collisions. The environment collision cost is defined as

$$C_{\text{coll-env}}(\boldsymbol{\tau}(s)) = \int_{\mathcal{B}} C_{\text{env}}(\boldsymbol{x}_m(\boldsymbol{q}(s)))\text{d}m \approx \sum_{m=0}^{S-1} C_{\text{env}}(\boldsymbol{x}_m)$$

with $C_{\text{env}}(\boldsymbol{x}_m) = \text{ReLU}(-\text{SDF}(\boldsymbol{x}_m) + r_m + \epsilon)$, where $\boldsymbol{x}_m \in \mathbb{R}^3$ is the position of the mth sphere center computed with forward kinematics at configuration $\boldsymbol{q}(s)$ and r_m its radius. To prevent (and control) the robot passing *too* close to objects, we use a safety margin $\epsilon \geq 0$ and activate the collision cost only if the robot is inside the safety margin.

The self-collision is similarly implemented, except the cost is computed between selected pairs of links

$$C_{\text{self}}(\boldsymbol{\tau}(s)) = \max_{i,j \in \mathcal{S}_c} \left(\text{ReLU}\left(-\|\boldsymbol{x}_i - \boldsymbol{x}_j\| + r_i + r_j + \epsilon\right)\right),$$

where S_c is the set containing all pairs of spheres of centers $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$, and radius r_i and r_j, considered for self-collision.

Joint limits. Joint limits in position, velocity, and acceleration are enforced by computing the L2-norm joint violations, with a small margin $\epsilon \geq 0$ on the d degrees-of-freedom

$$C_{\text{limits}}(\boldsymbol{\tau}(s)) = \sum_{\text{lim} \in \{\text{pos, vel, acc}\}} \sum_{i=0}^{d-1} C_{\text{lim}}(\boldsymbol{\tau}_d(s))$$

$$C_{\text{lim}}(\boldsymbol{\tau}_d(s)) = \begin{cases} \frac{1}{2}\|\boldsymbol{\tau}_{d,\text{min}}(s) + \epsilon - \boldsymbol{\tau}_d(s)\|_2^2 & \text{if } \boldsymbol{\tau}_d(s) < \boldsymbol{\tau}_{d,\text{min}}(s) + \epsilon \\ \frac{1}{2}\|\boldsymbol{\tau}_{d,\text{max}}(s) - \epsilon - \boldsymbol{\tau}_d(s)\|_2^2 & \text{if } \boldsymbol{\tau}_d(s) > \boldsymbol{\tau}_{d,\text{max}}(s) - \epsilon \\ 0 & \text{else} \end{cases}$$

where $\boldsymbol{\tau}_d(s)$ can be one of the partial derivatives w.r.t. s, $(\boldsymbol{q}_d(s), \boldsymbol{q}'_d(s), \boldsymbol{q}''_d(s))$.

When the constraints are in phase space, the minimum, and maximum limits are phase-dependent due to the dependency on $r(s)$ (eq. (23)). For positions, this plays no role since $\boldsymbol{q}_{\text{max}}(s) = \boldsymbol{q}_{\text{max}}\, \forall s$. However, for velocities we have $\boldsymbol{q}'_{\text{max}}(s) = \dot{\boldsymbol{q}}_{\text{max}} r(s)^{-1}$, with $\dot{\boldsymbol{q}}_{\text{max}}$ the maximum specified joint velocity. The acceleration limit follows similarly. With a linear phase-time relation we have $\boldsymbol{q}'_{\text{max}}(s) = \dot{\boldsymbol{q}}_{\text{max}} T$ and $\boldsymbol{q}''_{\text{max}}(s) = \ddot{\boldsymbol{q}}_{\text{max}} T^2$.

Algorithms 1 and 2 summarizes the procedures for learning and planning with MPD, respectively.

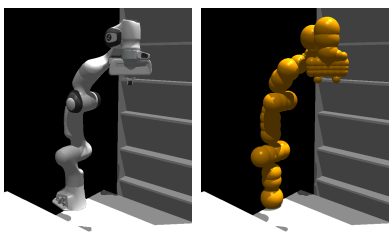


Fig. 4. Visualization of the Franka Emika Panda finite collision spheres model (right) used for faster collision cost computations.

IV. Experimental Evaluation

To understand the benefits of MPD, we construct experiments to answer the following questions:

Algorithm 1: Motion Planning Diffusion – Learning

Input: Collision-free trajectories dataset $\mathcal{D}$, B-spline parameters b (knots, degree, number of basis), denoising function $\boldsymbol{\epsilon}_{\boldsymbol{\theta}}$, noise schedule terms $\bar{\alpha}_i$, learning rate γ

while *not converged* **do**
- $\boldsymbol{c}, \boldsymbol{\tau}_0 \sim \mathcal{D}$ ▷ sample a batch of contexts and trajectories
- $\boldsymbol{w}_0 = \text{fit-B-spline}(b, \boldsymbol{\tau}_0)$ ▷ fit a B-spline to the trajectory and get control points
- ▷ compute the denoising loss function
- $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I}), i \sim \mathcal{U}(1, N), \boldsymbol{w}_i = \sqrt{\bar{\alpha}_i}\boldsymbol{w}_0 + \sqrt{1-\bar{\alpha}_i}\boldsymbol{\epsilon}$
- $\mathcal{L}(\boldsymbol{\theta}) = \|\boldsymbol{\epsilon} - \boldsymbol{\epsilon}_{\boldsymbol{\theta}}(\boldsymbol{w}_i, i, \boldsymbol{c})\|_2^2$
- $\boldsymbol{\theta} = \boldsymbol{\theta} - \gamma \boldsymbol{\nabla}_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta})$ ▷ gradient optimization step

Output: optimized $\boldsymbol{\theta}$

Algorithm 2: Motion Planning Diffusion – Planning

(DDPM version. For DDIM we adapt the posterior mean update with eq. (18))

Input: Pre-trained denoising model $\boldsymbol{\epsilon}_{\boldsymbol{\theta}}$, noise schedule terms $(\alpha_i, \bar{\alpha}_i, \sigma_i)$, start joint position $\boldsymbol{q}_{\text{start}}$ and goal end-effector pose ${}^{W}\boldsymbol{H}_{\text{goal}}^{EE}$, B-spline basis matrix $\boldsymbol{B}$, motion planning costs C_j and temperatures λ_j, prior temperature λ_{prior}, cost gradient start index i_{cost}, inner gradient parameters (steps M, step size γ, maximum step δ)

$\boldsymbol{c} = [\boldsymbol{q}_{\text{start}}, {}^{W}\boldsymbol{H}_{\text{goal}}^{EE}]$ ▷ build the conditioning variable

$\boldsymbol{w}_N \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$ ▷ sample noisy control points

for $i = N, \dots, 1$ **do**
- **if** $i > i_{cost}$ **then** $\lambda_{\text{prior}=1}$
- ▷ compute the diffusion prior mean eq. (6)
- $\boldsymbol{\mu}_i(\boldsymbol{w}_i, i, \boldsymbol{c}) = \frac{1}{\sqrt{\alpha_i}}\left(\boldsymbol{w}_i - \frac{1-\alpha_i}{\sqrt{1-\bar{\alpha}_i}}\lambda_{\text{prior}}\boldsymbol{\epsilon}_{\boldsymbol{\theta}}(\boldsymbol{w}_i, i, \boldsymbol{c})\right)$
- **if** $i < i_{cost}$ **then**
  - ▷ inner gradient steps eq. (15)
  - $\boldsymbol{\mu}_i^0 = \boldsymbol{\mu}_i$
  - **for** $k = 1, \dots, M$ **do**
    - ▷ compute B-spline trajectories eqs. (19), (24) and (25)
    - $\boldsymbol{\tau} = (\boldsymbol{q}, \boldsymbol{q}', \boldsymbol{q}'') = (\boldsymbol{B}\boldsymbol{\mu}_i^k, \boldsymbol{B}'\boldsymbol{\mu}_i^k, \boldsymbol{B}''\boldsymbol{\mu}_i^k)$
    - ▷ compute costs gradient eqs. (27) and (28)
    - $\boldsymbol{g} = -\sum_j \lambda_j \boldsymbol{\nabla}_{\boldsymbol{\mu}_i^k} C_j(\boldsymbol{\tau})$
    - ▷ Clip and apply the gradient
    - $\boldsymbol{\mu}_i^k = \boldsymbol{\mu}_i^{k-1} + \gamma \boldsymbol{g}$
    - $\Delta\boldsymbol{\mu} = \text{clip}(|\boldsymbol{\mu}_i^k - \boldsymbol{\mu}_i^0|, -\delta, \delta)$
    - $\boldsymbol{\mu}_i^k = \boldsymbol{\mu}_i^0 + \Delta\boldsymbol{\mu}$
- **else**
  - $\boldsymbol{\mu}_i^M = \boldsymbol{\mu}_i$
- ▷ sample from the posterior distribution
- $\boldsymbol{w}_{i-1} = \boldsymbol{\mu}_i^M + \boldsymbol{\Sigma}_i \boldsymbol{z},\ \boldsymbol{z} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$

▷ compute B-spline trajectories

$\boldsymbol{\tau}_0 = (\boldsymbol{q}_0, \boldsymbol{q}_0', \boldsymbol{q}_0'') = (\boldsymbol{B}\boldsymbol{w}_0, \boldsymbol{B}'\boldsymbol{w}_0, \boldsymbol{B}''\boldsymbol{w}_0)$

Output: batch of trajectories $\boldsymbol{\tau}_0$

Q1) Can MPD learn highly-multimodal collision-free trajectory distributions?
Q2) How does MPD compare to other generative models?
Q3) Is cost guidance during the denoising process necessary?
Q4) What are the impacts of the B-spline parametrization?
Q5) Can we encode trajectory priors from human demonstrations and adapt them at test time?

A. Environments, Tasks and Datasets

We consider a set of tasks that are representative of the challenges faced in robot motion planning, from simple to complex, including a 2D point mass navigating an environment with simple obstacles, a 2D point mass navigating a narrow passage, a 2-link planar robot, a 4-link planar robot, and a 7-dof robot arm manipulator in two settings: one illustrative environment with collision spheres; and a warehouse environment with a table and two shelves. The latter aims to motivate the applicability of our method in a real-world task, such as shelf rearrangements in distribution facilities, where shelves are static, but at deployment new objects can be placed on the table or shelves. The tasks are displayed in fig. 5, with naming convention [Environment]-[Robot]. The goal is to move the robot (without collisions) from start to goal configurations – a desired joint position or an end-effector pose.

For each task, we generate a dataset of collision-free paths with the OMPL library Python bindings [105], along with PyBullet [106] to setup the environment and perform collision checking. We randomly sample multiple contexts and use RRT Connect [5] to generate paths, which are afterward shortcutted and smoothed. Moreover, we generate an additional path by reverse-connecting the goal to the start configuration. Alternatively, if we know the task requires goal poses to be located in some regions of the workspace, we can specify the goal pose and use inverse kinematics to obtain the goal joint configuration. For example, in the EnvWarehouse-RobotPanda task, the goal pose is located on top of the table or on the shelves, and we sample paths from any initial configuration to those goal poses. There are two reasons for using RRT Connect to generate data. First, because it is faster than optimal planners like RRT*, and since we optimize these paths further using defined cost functions, we trade optimality for speed. Second, it is probabilistically complete, and its vertices cover all the configuration-free space (Corollary 1 of [5]), thus finding multiple modes for reaching the goal from the start configuration. The number of paths, which equals the number of contexts, generated per environment can be found in table II. Since we are planning in joint space, the number of points needed to cover the whole space increases exponentially with the number of joints. Thus, a sufficient amount of data is needed to counter the curse of dimensionality. E.g., for the discretization of 100 points per joint, a 4-dof robot arm requires $100^4 = 100$M points to cover the discretized space. When training the diffusion model prior, we sample a context and path from the dataset and fit a B-spline to the path using the `splprep` function from the SciPy library [107]. The number of B-spline control points is a hyperparameter and is chosen so the path almost does not collide with the environment. Note that we do not need to be 100% collision-free, as collision-avoidance costs are used during optimization.

Choosing the number of control points for the B-spline and the denoising network size is a trade-off between model expressivity and computational cost. The hyperparameters for training are detailed in appendix A. The loss function in eq. (7) is optimized using mini-batch gradient descent with the Adam optimizer [108] and a learning rate 3×10^{-4}.

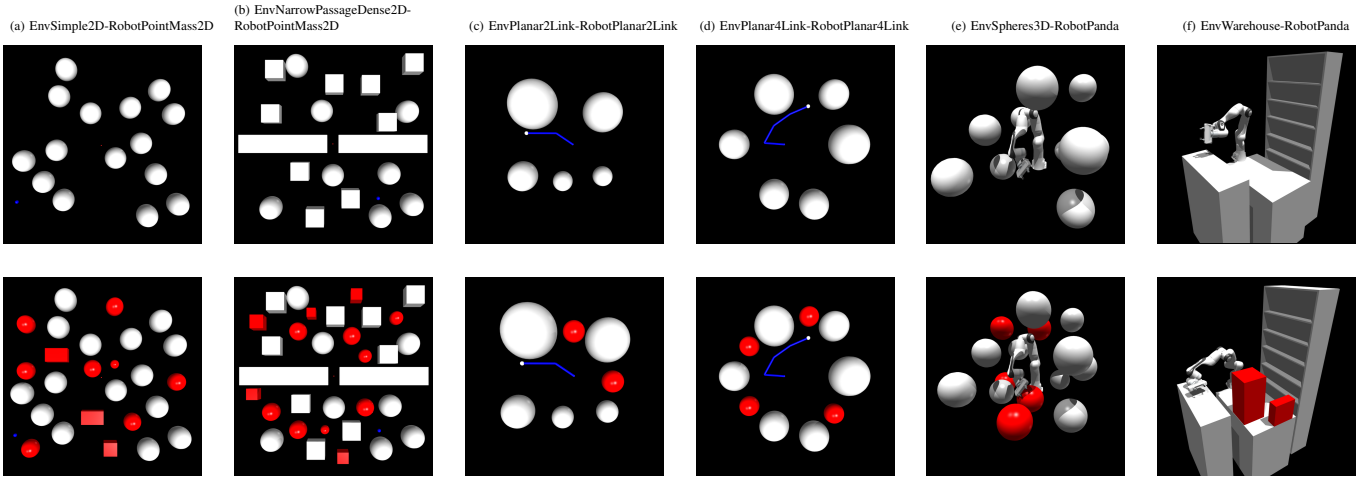


Fig. 5. The environments used for the motion planning experiments, with different robot models and increasing complexity. In 2D tasks the robots are depicted in blue (as a dot for RobotPointMass2D). The bottom row shows the tasks with additional obstacles (in red), which are not present during training.

Generalization is tested on contexts not seen during training and in the presence of new obstacles (see fig. 5), which assess the method's ability to exploit the prior knowledge to generate collision-free paths in unseen scenarios. The hyperparameters for inference are detailed in appendix B.

B. Baselines

We compare MPD against the following baselines:

- To compare the drawbacks of using an uninformed prior, we use a Gaussian Process (GP) prior that connects the start and goal configurations and optimizes the cost function using gradient descent. We name this baseline GPprior+Cost, and it is essentially CHOMP [2]. When the context includes the goal end-effector pose, we use a goal joint position computed with inverse kinematics.
- To assess the benefits of using a diffusion model, we use a baseline that learns a Conditional Variational Autoencoder (CVAE) [109] to model the trajectory distribution. It shares the same U-Net architecture as the denoising function $\boldsymbol{\epsilon_\theta}$, but at the lowest level of the U-Net the embedding is projected into a 32-dimensional latent space that encodes the mean and standard deviation of a Gaussian distribution. The optimized loss function is a sum of L2-norm of the predicted and ground-truth trajectory and the KL-divergence between the latent space posterior and a standard Gaussian distribution weighted by $\beta = 0.1$.
- We use a baseline named [Prior]+Cost to compare MPD against sampling first from the prior and then optimizing the cost function. The [Prior] is the diffusion (Dprior) or the CVAE model.

All baselines use the B-spline parametrization. For a similar comparison, we use the same number of cost function optimization steps in all baselines. To accelerate sampling from the diffusion model, we use DDIM with 15 steps and a quadratic step schedule. All methods are implemented with PyTorch, and the experiments were conducted on a machine with an AMD EPYC 7453 28-Core Processor and NVIDIA RTX 3090.

C. Metrics

We report the following average metrics. *Success rate* is 1 if at least one of the trajectories in the batch is not in collision and inside joint limits, and 0 otherwise. *Fraction of valid trajectories* is the percentage of generated trajectories that are collision-free and inside the joint limits. *Diversity* is measured by the Vendi score [110] with the similarity kernel $k(a, b) = \exp\left(-\|a - b\|_2^2\right)$ [111], for the dense interpolation of two trajectories a and b. A higher Vendi score means more diverse trajectories. *Error pos.* and *Error ori.* are the position and orientation errors, respectively, between the end-effector goal pose and the one at the end of the generated trajectory. The last four metrics are reported only for valid trajectories.

D. General Results in Simulation

In this experiment, we report the results using all motion planning costs. Here, we want to evaluate how the methods perform in terms of collisions and reaching the desired end-effector goal poses. Therefore, we use a large time duration of 10 seconds to prevent optimization over velocities and accelerations. We sample 100 contexts and optimize 100 trajectories per context. The results are displayed in fig. 6.

To answer Q1, we observe the results under *training environment* since they show how well the generative priors model the data distribution. They show that the diffusion-based models (Dprior, Dprior+Cost, MPD) obtain success rates that match or surpass the baselines while keeping a high diversity in generated trajectories. When comparing CVAE and Dprior, their success rate and validity fraction are close, but the diversity score is higher for the diffusion-based models, in particular for higher-dimensional environments (using RobotPanda). For instance, in the EnvWarehouse-RobotPanda, both CVAE and Dprior achieve 97% mean success rate, but Dprior shows more mean variability (50.9 vs. 60.3). Moreover, Dprior achieves smaller mean errors in the desired end-effector goal position and orientation (7.4cm vs. 5.2cm, 10.5° vs. 6.7°), which due to model approximations are not 0. These errors are improved during cost optimization. Both diffusion-based

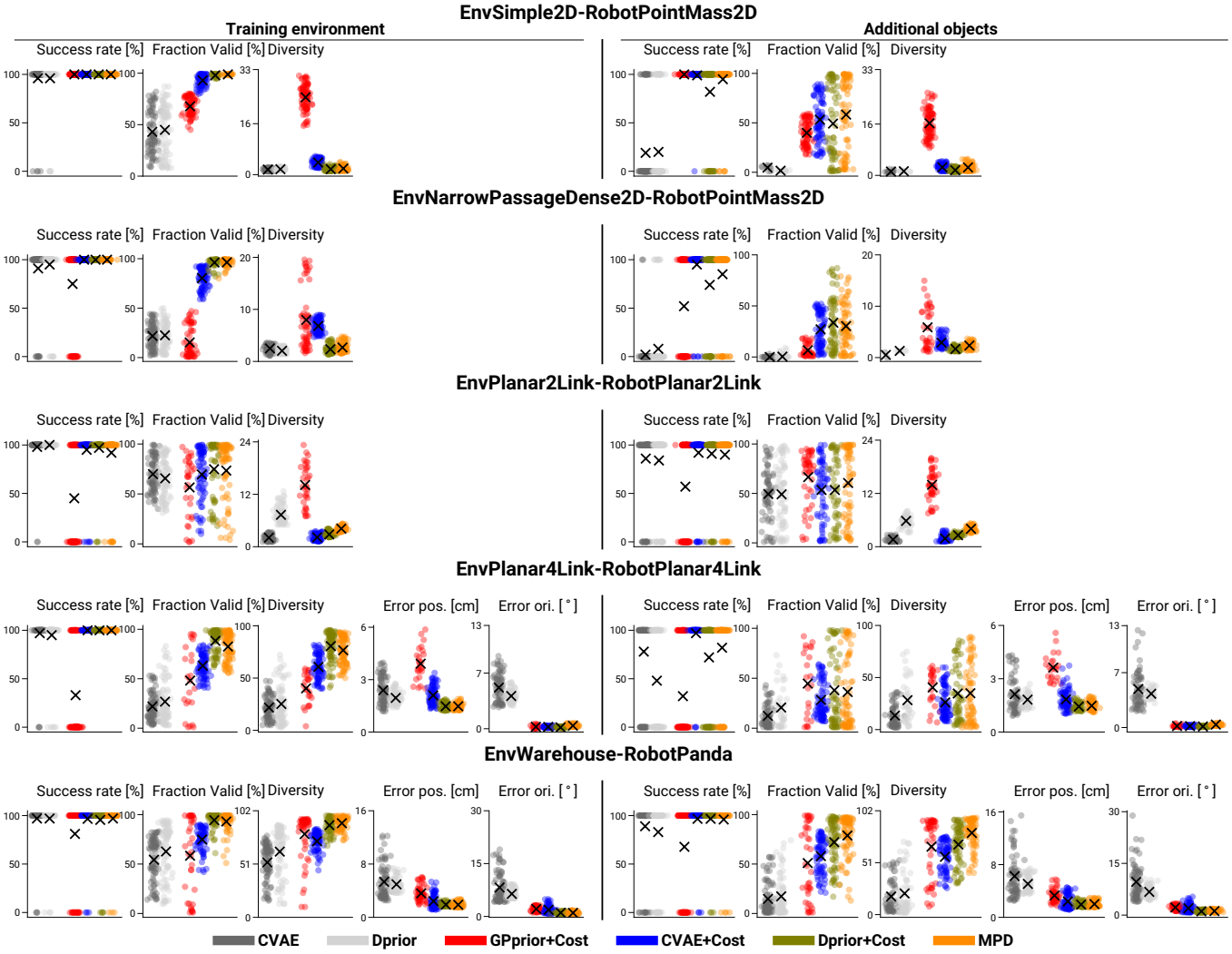


Fig. 6. Performance metrics for different algorithms on the tasks from fig. 5. The results report the swarm plot of sampling 100 contexts from the test set and optimizing 100 trajectories per context. The cross represents the mean value. The columns under *training environment* show the results without additional objects. In $2D$ environments, the errors in end-effector position and orientation are always 0 since a joint goal is provided.

models, Dprior+Cost and MPD, achieve similar mean position and orientation errors, e.g., in this task, 2.00cm and $1.1°$.

To answer Q2, we look at the results when using newer obstacles in the scene under *additional objects* columns. The success rate and validity fraction drop for all methods, in comparison to the training environment, which is expected, in particular for the prior methods, CVAE and Dprior, which do not have knowledge of the new obstacles. For instance, in the EnvNarrowPassageDense2D-RobotPointMass2D task, Dprior's mean success rate dropped from 95% to 8%. This is a task where an informed prior is particularly useful since traversing from the top to the bottom of the environment needs to be done through a very small passage. We focus now on the higher-dimensional task EnvWarehouse-RobotPanda. In the presence of new obstacles, all methods that perform cost optimization improve the success rate and validity fraction in comparison to using only the prior. For example, MPD increases the mean validity fraction of Dprior from 18.4 to 73.5%, which is the largest among all methods for this task. Moreover, MPD also shows more mean diversity in the valid trajectories in comparison to baselines that first sample from the prior and then optimize the cost (66.0 for Dprior+Cost vs. 74.8 for MPD). Note that these results are more pronounced for the *additional objects* tasks, and a discussion for why this happens is found in section IV-E. When comparing MPD with an uninformed prior GPprior+Cost, we observe a mean success rate increase from 67.8% to 96.0%, which supports the claim that the diffusion prior is useful to guide the trajectories through collision-free regions. In terms of diversity, we expect the GPprior to provide the largest value since this is an uninformed prior. MPD's diversity is the closest to the GPprior one or slightly larger for the EnvWarehouse-RobotPanda task. These results show that MPD produces a higher percentage of collision-free trajectories (and inside joint limits) while being diverse. It is also important to note the effect of optimizing the task cost. Methods that optimize this cost produce trajectories resulting in end-effector pose errors much smaller than the ones obtained just sampling from the priors. E.g., in the EnvWarehouse-RobotPanda task, the end-effector mean error in orientation drops from 7.1 to 1.2 between Dprior and MPD.

The optimization time is highly implementation-dependent (and on CPU/GPU hardware versions). We parallelize all

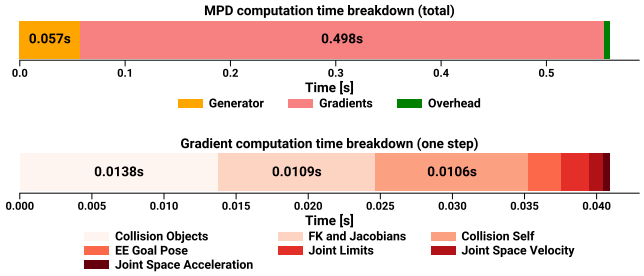


Fig. 7. Computation times breakdown for sampling 100 trajectories with MPD in the EnvSpheres3D-RobotPanda environment. (Top) MPD total inference time and breakdown per generator and gradient computations. (Bottom) Breakdown for the computation of one gradient step.

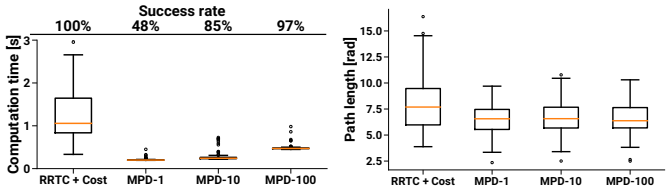


Fig. 8. (Left) computation time and success rate, and (right) shortest path lengths results in the EnvSpheres3D-RobotPanda task (additional objects) across 100 contexts. We compare one RRT Connect sample with cost optimization (RRTC + Cost), and MPD with increasing batches (1, 10, 100).

cost computations using vectorized operations in PyTorch, but computing the costs is still done sequentially in Python. Typically, a disadvantage of diffusion models is their slow sampling speed, but by using DDIM we reduce the number of sampling steps, thus achieving faster inference without a significant performance drop. In fig. 7, we show the computation time breakdown for MPD in the most complex environment (EnvSpheres3D-RobotPanda), using MPD with 15 DDIM steps and 4 intermediate gradient steps, which in total takes approximately 0.56s. The diffusion sampling alone takes 0.057s ($\approx$ 3.8ms per denoising network pass). The costly part is the gradient computation, which is similar for all methods. Therefore, the only difference between the baselines, is diffusion taking 0.057s more.

Figure 1 presented a motivation for initializations in optimization-based planning methods. In our previous work [24], the experiments showed that (as expected) a sampling-based prior achieves 100% success rates, but with more computation time (hence, we do not include it here in the main baselines). In this work, we consider sampling *one* trajectory using RRTConnect, since parallel sampling incurs too much computation time as it is not as easy to parallelize. We use the same implementation as described in section IV-A. The question we aim to answer is whether MPD has a benefit over first running a sampling-based planner and then solving the optimization problem from eq. (1) (we used the same number of steps as for MPD). For this experiment, we chose the EnvSpheres3D-RobotPanda task with additional objects because it is the hardest task we considered (due to narrow passages). Figure 8 shows the results obtained when planning 100 contexts. We observe that sampling *one* trajectory with RRTConnect uses more computation time than sampling 100 with a learned diffusion model. MPD's success rate increases with the batch size, while computation time remains almost

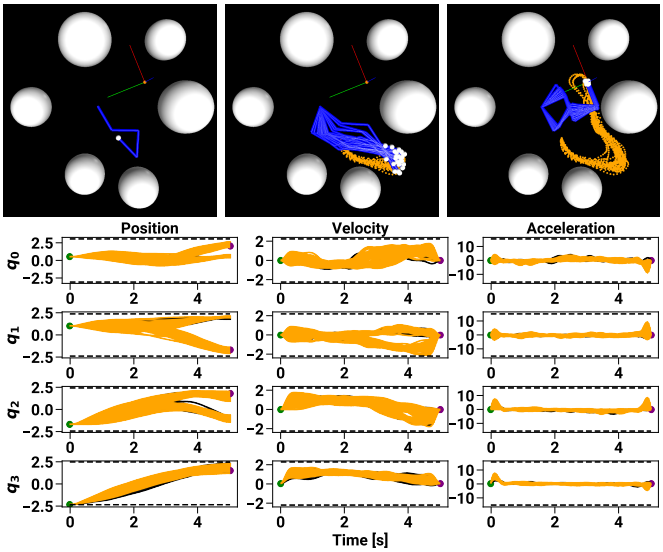


Fig. 9. (Top) Task-space trajectories generated by MPD in the EnvPlanar4Link-RobotPlanar4Link task. The reference frame shown is the desired end-effector pose. The orange line shows the end-effector trajectories from the start (left) to the goal (right). Notice how the model generates multimodal goal joint positions for the same desired end-effector goal pose. (Bottom) Joint trajectories in time, where orange are collision-free and within joint limits (depicted by the dashed lines), and others are in black.

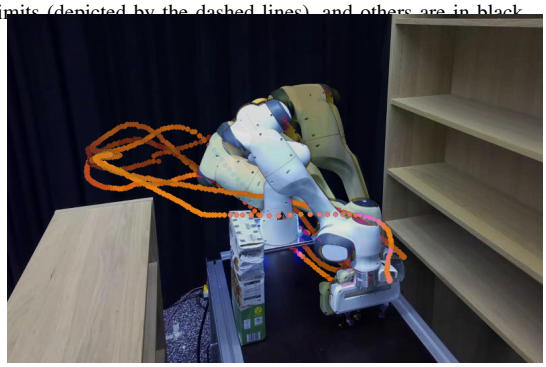


Fig. 10. Example of multimodal trajectories executed in the warehouse environment using MPD. Starting from the same joint configuration, the task is to reach an end-effector pose on the table while avoiding collisions with the environment. The orange lines depict the end-effector trajectories, and the overlayed images show the last frame on the trajectory. Our model generates multiple goal robot configurations for similar desired end-effector poses.

constant. The boxplot on the right side also shows that MPD produces trajectories with shorter lengths and less variance. This experiment shows that offloading sampling-based planner computations and compressing them into a trajectory diffusion model leads to faster and better planning results.

Figure 9 shows an example of a planning task in the EnvPlanar4Link-RobotPlanar4Link environment, using a trajectory duration of 5 seconds. Given a goal end-effector pose, the model generates multimodal trajectories in joint space, which are collision-free and within joint limits. The rightmost figure on the top row shows that final joint configurations achieve the same end-effector pose. We note that the diffusion model was trained using one trajectory per context, but when tested in a new context not seen during training, the model generates multimodal trajectories, which we attribute to the capacity of the denoising network to generalize to new contexts and model multimodal distributions quite well.

To show that our model is transferable from the digital

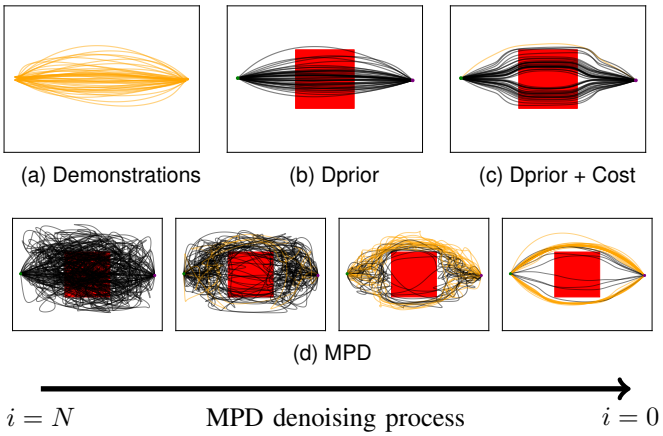


Fig. 11. The figures illustrate the benefits of sampling with MPD instead of first sampling from the prior and then optimizing the cost function. (a) The robot is a 2D point mass in an environment without any obstacles, and demonstration trajectories are obtained by sampling from a Gaussian Process prior. At inference, an obstacle (in red) is inserted into the environment. (b) Samples from the learned diffusion prior. (c) Results of first sampling from the prior and then optimizing the collision cost function. (d) Samples from the posterior distribution generated with MPD. The second row illustrates how MPD's denoising process moves from Gaussian noise to clean trajectories while avoiding the red obstacle. Trajectories that are in collision with the red obstacles are shown in black, and collision-free trajectories in orange.

twin to a real-world scenario, we execute MPD by replicating the EnvWarehouse-RobotPanda environment and adding additional objects to the scene, whose pose we obtain through a marker-based system. Figure 10 shows an example of how MPD generates multimodal trajectories with different joint goal configurations for similar end-effector poses.

E. MPD vs. Diffusion Prior and Cost Optimization

In this section, we answer Q3. One important aspect when using cost-guided diffusion in the context of motion planning is to understand in what situations can sampling from the denoising posterior distribution eq. (8) be beneficial, compared to first sampling from the diffusion prior and then optimizing the cost function starting from the proposed samples. In fig. 11 we show a simple example to motivate this difference. We consider a 2D point mass in an environment without any obstacle (fig. 11a), generate a small set of demonstrations with a Gaussian Process trajectory, and learn a trajectory diffusion prior. At inference we add a new obstacle, the red square, and do optimization using the object collision cost. The figures show two aspects of MPD. First, while sampling from the diffusion prior and then optimizing the cost fails to provide collision-free trajectories (fig. 11c), MPD can obtain a larger percentage of those (fig. 11d). This phenomenon happens because once a large portion of the trajectory is in collision with the obstacle, it becomes more difficult to remove it from collision. Second, in fig. 11d, we notice how trajectory samples from MPD evolve during the denoising process. The evolution shows that sampling from the posterior distribution moves the samples towards regions of the configuration space where the posterior has higher probability density, thus generating samples that are collision-free but close to the prior distribution. Therefore, the benefits of sampling from the posterior

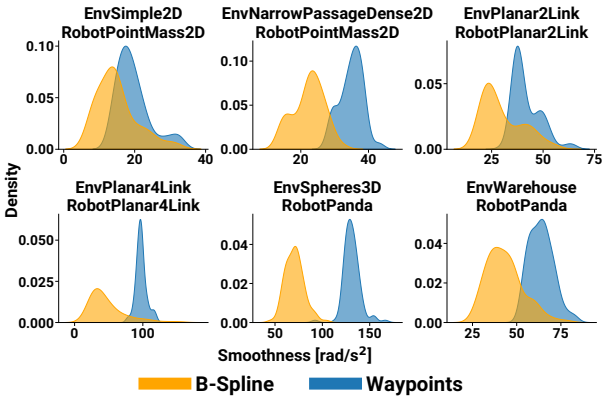


Fig. 12. KDE plots of the smoothness distribution of all contexts in all tasks with additional objects (lower values mean smoother trajectories).

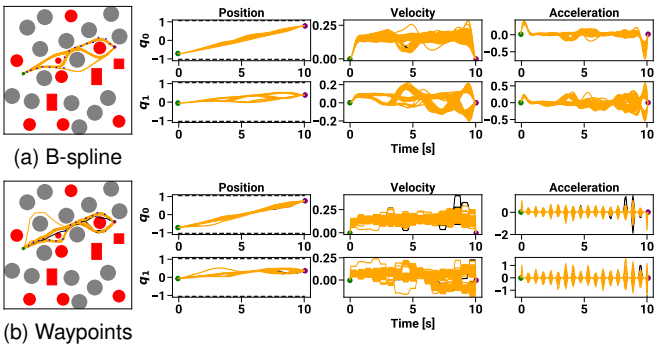


Fig. 13. MPD results in the EnvSimple2D-RobotPointMass2D task using different trajectory parameterizations. Trajectories in orange are collision-free, and black in collision. (a) The first row shows the generated trajectories using the B-spline parametrization, while (b) the second one is generated with waypoints. By inspecting the trajectories' evolution, we observe jumps in the acceleration profile due to using a few waypoints. At the same time, the B-spline parametrization ensures smoothness in velocity and acceleration profiles. The blue dots in the environment plots show the control points or waypoints of one trajectory.

are more diverse and collision-free trajectories, which can be observed in fig. 6 by comparing Dprior+Cost and MPD results, especially in the environments with additional objects.

F. Impact of the B-spline Parametrization

One of this work's contributions is performing diffusion in the space of B-spline coefficients instead of waypoints. Hence, to answer Q4, we compare both representations. The benefits have been detailed in section III-E, but perhaps the main one is generating smooth trajectories. As the velocity and acceleration of a linear waypoint trajectory representation are computed with central finite differences, they are constant between waypoints and lead to large jumps. There are two ways to get smoother trajectories with this representation. First, we can increase the number of waypoints, as commonly done in optimization-based algorithms. Second, after running the denoising process, we can do extra optimization steps to smoothen and optimize the trajectory to remain collision-free and within joint limits while minimizing the task cost. Both of these options have drawbacks in terms of additional computations, e.g., in the first case, the denoising network needs to process larger inputs. We performed an ablation experiment where we replaced the B-spline with waypoints. We used the same number of waypoints as B-spline coefficients, which guaran-

tees that the denoising network size and computations are similar. As expected, there are no major differences across all metrics except for smoothness values [1], which are better (lower) for the B-spline compared to the sparse waypoint representation. To better grasp the difference visually, fig. 12 shows the distribution of smoothness values obtained with MPD across all contexts and tasks. The waypoint representation consistently leads to worse smoothness. Figure 13 shows a visual comparison in the EnvSimple2D-RobotPointMass2D tasks. The figures show that even though the waypoint representation can generate valid trajectories, they are not smooth, as can be seen in the straighter path in the environment plot and in the velocity and acceleration time profiles. These types of trajectories are harder for a robot controller to follow and can lead to jerky movements.

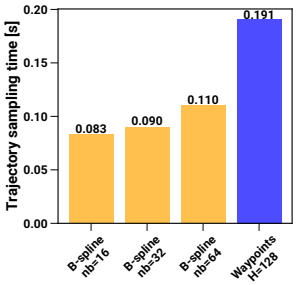


Fig. 14. Computation times for denoising a batch of 1000 trajectories with B-splines and dense Waypoints using $H = 128$ and $d = 7$ with 15 steps of DDIM.

A benefit of B-splines in comparison to using a dense trajectory is the faster generation time, as the denoising function processes inputs with lower dimensions. To generate trajectories with H points, we can sample from a model that outputs this trajectory size, or we can sample $n_b < H$ control points of a B-spline, and then interpolate the trajectory to H using eq. (19). Figure 14 shows the computation times when generating a batch of trajectories with B-splines for an increasing number of control points, resulting from sampling the diffusion model and interpolating with eq. (19), and sampling waypoint trajectories with $H = 128$. We consider only denoising times and interpolation (whose time is minimal), since the computation of costs/gradients only depends on the number of dense points H, and is the same for B-splines or dense waypoint representations. The results show the computational benefit of performing diffusion with lower-dimensional representations, instead of dense trajectories.

G. Learning and Adapting from Human Demonstrations

Learning the prior distribution on trajectories is agnostic to the expert demonstrations. Previously, we assumed that trajectories would be generated by a sampling-based path planner. However, in real-world tasks, it is sometimes desirable and easier for a human to provide demonstrations via kinesthetic teaching of what movement they would like the robot to execute. Therefore, to answer Q5, in this experiment, we learn the trajectory distribution directly from human demonstrations, showing that MPD can be used in this scenario as well. Using the EnvWarehouse-RobotPanda environment, we consider the task of moving one item (a tea box) from several locations in the small to the big shelf and letting a user perform multiple demonstrations from random start and goal poses by moving the robot in gravity compensation with kinesthetic teaching from regions on the small shelf to the big shelf, generating approximately 100 trajectories. These are augmented using small noise values to perform a total of approximately 4k trajectories. Examples of human-demonstrated trajectories are shown in fig. 15. The diffusion prior model is then trained on joint space trajectories similarly and with the same hyperparameters as the simulation experiments.

[1]Smoothness is measured as the sum of accelerations along the trajectory.

TABLE I
PERFORMANCE METRICS FOR A REAL-WORLD MOTION PLANNING TASK IN THE ENVWAREHOUSE-ROBOTPANDA ENVIRONMENT WITH ADDITIONAL OBJECTS USING DIFFUSION-BASED MODELS.

Algorithms	Success rate [%]	Fraction valid [%]	Diversity
Dprior	0.0	–	–
Dprior+Cost	100.0	37.0	27.95
MPD	100.0	78.0	59.50

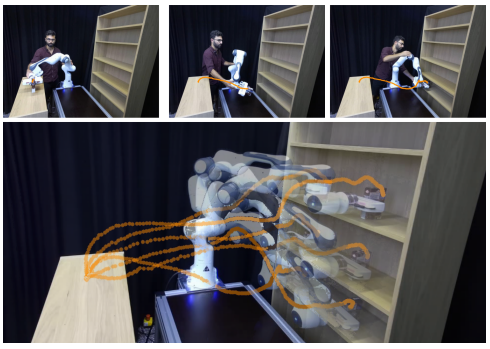
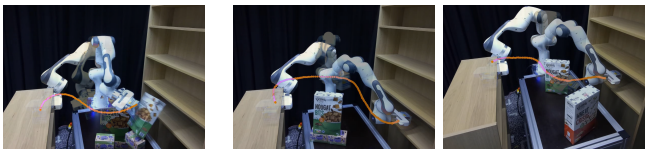


Fig. 15. (Top) Human demonstrations via kinesthetic teaching for a pick-and-place task in the EnvWarehouse-RobotPanda environment. The end-effector trajectories are depicted in orange. (Bottom) Overlay of several demonstrations. The robot configurations are shown at the end of the demonstration.



(a) Diffusion prior (b) MPD

Fig. 16. (a) As the demonstrations did not include the objects placed on the table, the diffusion prior failed to obtain a collision-free trajectory and crashed against the obstacle. (b) By using the collision-avoidance cost during posterior sampling, MPD solutions can stay close to the human demonstrations (see fig. 15) while avoiding collisions with the obstacles and reaching the desired end-effector pose.

For these experiments, we only consider diffusion-based methods. We tested MPD by placing additional objects in the scene (cereal and tea boxes) (see fig. 16b), and computed the environment's signed distance function using the updated object poses, estimated with a marker-based system. For inference, we use the same hyperparameters as the simulation experiments to generate trajectories from the current robot's joint position to a desired end-effector goal pose, which we assume to be determined by a higher-level task planner.

The relevant metrics and results across different contexts are found in table I. The diffusion prior fails to obtain any collision-free trajectory since the prior trajectory distribution passes through the objects, as shown in an example in fig. 16a. Both Dprior+Cost and MPD achieved a 100% success rate while finding different modes for solving the task. Figure 16b shows two valid trajectories obtained with MPD after placing the additional objects in different locations. These trajectories are collision-free and close to the demonstrations from fig. 15.

V. Limitations and Future Work

In this section, we present some limitations of MPD and leave proposals for future work.

To incorporate first-order gradients while sampling from the posterior distribution, we use the approach from classifier-guided diffusion. Instead, we can use a projection step at each denoising step, such that the sample lies on some manifold (for instance, ensuring safety constraints) [112]. However, the projection can be computationally expensive.

In this work, we kept the trajectory duration fixed, since optimizing it during the denoising process is not trivial, as the phase-time derivative would constantly change. Thus, the joint velocity and acceleration limits are not constant in phase space. A research direction would be to optimize the trajectory duration, possibly using a separate B-spline curve to represent the phase-time derivative from eq. (23) [93].

The current inference speed highly depends on the current Python implementation. One common appointed disadvantage of diffusion models is that they are slow to sample due to the number of denoising steps. However, in our problem, much computation time is used for the cost function gradients. Even though single-cost computation is not slow (done at ≈ 100Hz) and parallelizable, the costs must be computed sequentially due to the Python GIL. Using other programming frameworks, such as JAX [113], which uses JIT compilation, might be an alternative to improve speed further.

We considered environments that do not undergo major structural changes. These are common across several scenarios, such as warehouses, where shelves are not constantly moved, but new obstacles can appear. This work can be extended with an additional context variable that encodes the current environment. However, for proper generalization, this would require a large amount of data to be collected to cover a large possibility of arrangements across many environments [73]. Instead of generalizing, our work focuses on specializing in a single environment (e.g., as in [114]) and showing the properties of diffusion for motion planning.

VI. Conclusion

In this article, we proposed Motion Planning Diffusion (MPD), an algorithm that uses diffusion for learning-based motion planning tasks. A novelty in our approach is to parametrize the trajectory using B-spline coefficients and model the denoising network in this weight space. Compared to a waypoint representation, using the same number of coefficients, this parametrization ensures smoothness and processing of smaller inputs. During training, the diffusion model learns a distribution over control points obtained by fitting a B-spline to paths obtained from a sampling-based planner or human demonstrations. At inference time, given a motion planning cost function to optimize, which includes avoiding object collisions, promoting joint limits, and reaching an end-effector goal pose, we propose using planning-as-inference to obtain a distribution of trajectories that balances the prior distribution and the cost likelihood. We do this by sampling the denoising posterior distribution using cost-guided diffusion. Therefore, the generated samples are strongly biased toward prior solutions, which are more useful than a uniformed prior, such as a straight-line trajectory in configuration space. To show generalization, we add additional objects to the environments used during training, leading to adaptation to new environments.

Our experimental results show that diffusion models are desirable priors for encoding trajectory distributions because they model multimodality quite well, and blending sampling and optimization is beneficial in generating collision-free trajectories. We empirically show and provide intuition on why sampling from the posterior is beneficial compared to sampling from the prior and then optimize the cost function using those samples as initialization. Along with learning from paths generated in simulation, we also learn a real-world pick-and-place task from human trajectory demonstrations via kinesthetic teaching. Across all simulated and real-world tasks, MPD either matches or surpasses the evaluated baselines, showing it can generate a higher percentage of valid trajectories while keeping a higher diversity in the generated samples.

Acknowledgments

This work was funded by the German Federal Ministry of Education and Research projects IKIDA (01IS20045) and Software Campus project ROBOSTRUCT (01S23067), and by the German Research Foundation project METRIC4IMITATION (PE 2315/11-1) and supported by the Foundation for Polish Science (FNP).

Appendix A
Hyperparameters Training

Table II shows the hyperparameters used for training the diffusion model in each task. We generate one trajectory per context (start and goal configuration), so the dataset size equals the total number of contexts and trajectories. The number of control points is task-dependent and chosen such that after fitting a 5th order B-spline, we obtain over 99% of collision-free paths with a minimal collision rate. The *U-Net input dim* is the number of dimensions the control points are projected at the network input. The number of entries in the *U-Net dim multiplier* is the U-Net depth (layers) (3 or 4, depending on the task), while the numbers represent the channels in each layer of the U-Net, which are multiplied by the input dimension. *Context out dim* is context network output size. We use a fixed number of $N = 100$ diffusion steps for all tasks. The context and the denoising networks' parameters are optimized using mini-batch gradient descent with the Adam optimizer [108] and a learning rate 3×10^{-4}. The 2D models took approximately 12 hours to train, and the more complex ones 24 hours (these can be improved by early stopping). Large batch sizes are beneficial when training diffusion because if the dataset has D points, the model is trained for O optimization steps with a batch size B, then each data point should be seen roughly OB/D times (approximately the number of epochs). The diffusion loss function eq. (7) includes an expectation over the N diffusion time steps. Therefore, the pair of data (control points, diffusion step) is roughly seen $OB/(DN)$ times, which motivates using a larger batch size and a larger number of

TABLE II
HYPERPARAMETERS FOR TRAINING MPD.

Task	Dataset size	B-spline control points	UNet input dim	UNet dim multiplier	Context out dim	Batch size	Optimization steps
EnvSimple2D-RobotPointMass2D	10k	22	32	(1, 2, 4)	32	128	2M
EnvNarrowPassageDense2D-RobotPointMass2D	10k	30	32	(1, 2, 4)	32	128	2M
EnvPlanar2Link-RobotPlanar2Link	10k	22	32	(1, 2, 4)	32	128	2M
EnvPlanar4Link-RobotPlanar4Link	100k	22	32	(1, 2, 4, 8)	128	512	3M
EnvSpheres3D-RobotPanda	1M	30	32	(1, 2, 4, 8)	128	512	3M
EnvWarehouse-RobotPanda	500k	22	32	(1, 2, 4, 8)	128	512	3M

optimization steps than other generative models. The CVAE baseline model is trained with the same hyperparameters.

APPENDIX B
HYPERPARAMETERS INFERENCE

TABLE III
MOTION PLANNING COST WEIGHTS (EQ. (2)).

Costs	Collision	Joint limits	Task	Velocity	Acceleration
Weights	0.9	0.5	0.5	0.2	0.2

We use DDIM for sampling from the diffusion model with 15 steps and a quadratic noise schedule to focus more time steps in lower noise regions. Cost guidance is activated at the last $i_{\text{cost}} = 3$ steps of denoising, the number of intermediate gradient steps is set to $M = 4$, and $\lambda_{\text{prior}} = 0.25$ (see algorithm 2), amounting to 12 cost gradient steps. The gradient weight is $\gamma = 1.0$ and $\delta = 0.15$ (note that the update is done in the unit-normalized control points space). We found that good values for i_{cost} to be approximately $1/3$ (or less) of the total denoising steps, and λ_{prior} in $[0.25, 0.5]$. M and i_{cost} are a tradeoff between using more cost guidance vs. more denoising steps from the prior. If used for sampling within the training environments (without additional objects), M can be lowered. The cost weights reflect the emphasis put on each cost and are task-dependent, but not algorithm-dependent. These are shown in table III. The trajectory is interpolated to 128 points using a B-spline or waypoint parametrization to evaluate the costs and compute gradients on the dense trajectory representation. We did not vary the task duration and left its optimization for future work. Therefore, in section IV-D we use 10s (to report collision-free results), and 5s for the experiment in fig. 9 (to show the effect of joint limits costs). In the real-world tasks, we kept the duration at 6s to safely operate the robot.

REFERENCES

[1] S. M. LaValle, *Planning Algorithms*. Cambridge, U.K.: Cambridge University Press, 2006.

[2] N. Ratliff, M. Zucker, *et al.*, "Chomp: Gradient optimization techniques for efficient motion planning," in *IEEE International Conference on Robotics and Automation*, 2009.

[3] M. Mukadam, J. Dong, *et al.*, "Continuous-time gaussian process motion planning via probabilistic inference," *Int. J. Robotics Res.*, vol. 37, no. 11, 2018.

[4] J. Schulman, J. Ho, *et al.*, "Finding locally optimal, collision-free trajectories with sequential convex optimization," in *Robotics: Science and Systems IX*, 2013.

[5] J. Kuffner and S. LaValle, "Rrt-connect: An efficient approach to single-query path planning," in *IEEE ICRA*, 2000.

[6] J. E. Leu, G. Zhang, *et al.*, "Efficient robot motion planning via sampling and optimization," in *2021 American Control Conference, ACC 2021, New Orleans, LA, USA, May 25-28, 2021*, IEEE, 2021.

[7] A. D. Dragan, G. J. Gordon, *et al.*, "Learning from experience in manipulation planning: Setting the right goals," in *Robotics Research : The 15th International Symposium ISRR*, H. I. Christensen and O. Khatib, Eds. Cham: Springer International Publishing, 2017.

[8] T. S. Lembono, "Memory of motion for initializing optimization in robotics," en, Ph.D. dissertation, EPFL, Lausanne, 2022.

[9] T. S. Lembono, A. Paolillo, *et al.*, "Memory of motion for warm-starting trajectory optimization," *IEEE Robotics Autom. Lett.*, vol. 5, no. 2, 2020.

[10] D. Koert, G. Maeda, *et al.*, "Demonstration based trajectory optimization for generalizable robot motions," in *IEEE-RAS Humanoids*, 2016.

[11] M. A. Rana, M. Mukadam, *et al.*, "Towards robust skill generalization: Unifying learning from demonstration and motion planning," in *CoRL*, PMLR, 2017.

[12] K. P. Murphy, *Probabilistic Machine Learning: An introduction*. MIT Press, 2022.

[13] O. Arenz, M. Zhong, *et al.*, "Efficient gradient-free variational inference using policy search," in *Proceedings of the 35th International Conference on Machine Learning*, vol. 80, PMLR, 2018.

[14] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," in *2nd International Conference on Learning Representations*, 2014.

[15] I. Goodfellow, J. Pouget-Abadie, *et al.*, "Generative adversarial nets," in *Advances in Neural Information Processing Systems*, vol. 27, Curran Associates, Inc., 2014.

[16] Y. Lecun, S. Chopra, *et al.*, "A tutorial on energy-based learning," English (US), in *Predicting structured data*. MIT Press, 2006.

[17] J. Ho, A. Jain, *et al.*, "Denoising diffusion probabilistic models," in *NeurIPS*, Curran Associates Inc., 2020.

[18] Y. Song, J. Sohl-Dickstein, *et al.*, "Score-based generative modeling through stochastic differential equations," in *International Conference on Learning Representations*, 2021.

[19] M. Arjovsky, S. Chintala, *et al.*, "Wasserstein generative adversarial networks," in *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, vol. 70, PMLR, 2017.

[20] P. Florence, C. Lynch, *et al.*, "Implicit behavioral cloning," *Conference on Robot Learning (CoRL)*, 2021.

[21] R. Bayat, "A study on sample diversity in generative models: Gans vs. diffusion models," in *The First Tiny Papers Track at ICLR*, OpenReview.net, 2023.

[22] T. Osa, "Multimodal trajectory optimization for motion planning," *Int. J. Robotics Res.*, vol. 39, no. 8, 2020.

[23] P. Dhariwal and A. Q. Nichol, "Diffusion models beat gans on image synthesis," in *NeurIPS*, 2021.

[24] J. Carvalho, A. T. Le, *et al.*, "Motion planning diffusion: Learning and planning of robot motions with diffusion models," in *IROS*, 2023.

[25] L. Kavraki, P. Svestka, *et al.*, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE Transactions on Robotics and Automation*, 1996.

[26] S. M. Lavalle, *Rapidly-exploring random trees: A new tool for path planning*, 1998.

[27] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *The International Journal of Robotics Research*, vol. 30, no. 7, 2011.

[28] J. D. Gammell, S. S. Srinivasa, *et al.*, "Informed rrt*: Optimal sampling-based path planning focused via direct sampling of an admissible ellipsoidal heuristic," in *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2014.

[29] J. D. Gammell, T. D. Barfoot, *et al.*, "Batch informed trees (bit*): Informed asymptotically optimal anytime search," *The International Journal of Robotics Research*, vol. 39, no. 5, 2020.

[30] M. P. Strub and J. D. Gammell, "Adaptively informed trees (ait*): Fast asymptotically optimal path planning through adaptive heuristics," in

2020 IEEE International Conference on Robotics and Automation, ICRA 2020, Paris, France, May 31 - August 31, 2020, IEEE, 2020.
[31] A. A. Ravankar, A. A. Ravankar, *et al.*, "Path smoothing techniques in robot navigation: State-of-the-art, current and future challenges," *Sensors*, vol. 18, no. 9, 2018.
[32] E. Theodorou, J. Buchli, *et al.*, "A generalized path integral control approach to reinforcement learning," *J. Mach. Learn. Res.*, vol. 11, Dec. 2010.
[33] M. Kalakrishnan, S. Chitta, *et al.*, "Stomp: Stochastic trajectory optimization for motion planning," in *IEEE International Conference on Robotics and Automation*, 2011.
[34] J. Urain, A. Le, *et al.*, "Learning implicit priors for motion optimization," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2022.
[35] L. Petrović, I. Marković, *et al.*, "Mixtures of gaussian processes for robot motion planning using stochastic trajectory optimization," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 2022.
[36] A. T. Le, G. Chalvatzaki, *et al.*, "Accelerating motion planning via optimal transport," in *Advances in Neural Information Processing Systems*, 2023.
[37] M. Stolle and C. G. Atkeson, "Policies based on trajectory libraries," in *Proceedings of the 2006 IEEE International Conference on Robotics and Automation*, IEEE, 2006.
[38] C. Liu and C. G. Atkeson, "Standing balance control using a trajectory library," in *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*, IEEE, 2009.
[39] N. Mansard, A. DelPrete, *et al.*, "Using a memory of motion to efficiently warm-start a nonlinear predictive controller," in *2018 IEEE International Conference on Robotics and Automation, ICRA 2018, Brisbane, Australia, May 21-25, 2018*, IEEE, 2018.
[40] D. Forte, A. Gams, *et al.*, "On-line motion synthesis and adaptation using a trajectory database," *Robotics Auton. Syst.*, vol. 60, no. 10, 2012.
[41] R. Lampariello, D. Nguyen-Tuong, *et al.*, "Trajectory planning for optimal robot catching in real-time," in *IEEE International Conference on Robotics and Automation*, IEEE, 2011.
[42] T. Power and D. Berenson, "Variational inference mpc using normalizing flows and out-of-distribution projection," *Robotics: Science and Systems 2022.*, 2022.
[43] T. Osa, J. Pajarinen, *et al.*, "An algorithmic perspective on imitation learning," *Found. Trends Robotics*, vol. 7, no. 1-2, 2018.
[44] J. Peters, K. Mülling, *et al.*, "Relative entropy policy search," in *AAAI*, AAAI Press, 2010.
[45] A. Graikos, N. Malkin, *et al.*, "Diffusion models as plug-and-play priors," in *Advances in Neural Information Processing Systems*, 2022.
[46] I. Kapelyukh, V. Vosylius, *et al.*, *Dall-e-bot: Introducing web-scale diffusion models to robotics*, 2022.
[47] A. Ramesh *et al.*, *Hierarchical text-conditional image generation with clip latents*, 2022.
[48] W. Liu, T. Hermans, *et al.*, *Structdiffusion: Object-centric diffusion for semantic rearrangement of novel objects*, 2022.
[49] J. Urain, N. Funk, *et al.*, "Se(3)-diffusionfields: Learning smooth cost functions for joint grasp and motion optimization through diffusion," in *IEEE ICRA*, 2023.
[50] Z. Weng, H. Lu, *et al.*, "Dexdiffuser: Generating dexterous grasps with diffusion models," *IEEE Robotics and Automation Letters*, 2024.
[51] Z. Zhang, L. Zhou, *et al.*, "Dexgrasp-diffusion: Diffusion-based unified functional grasp synthesis pipeline for multi-dexterous robotic hands," *CoRR*, vol. abs/2407.09899, 2024.
[52] M. Janner, Y. Du, *et al.*, "Planning with diffusion for flexible behavior synthesis," in *ICML*, 2022.
[53] C. Chi, S. Feng, *et al.*, "Diffusion policy: Visuomotor policy learning via action diffusion," in *Robotics: Science and Systems XIX, Daegu, Republic of Korea, July 10-14, 2023*, 2023.
[54] M. Reuss, M. Li, *et al.*, "Goal-conditioned imitation learning using score-based diffusion policies," in *Robotics: Science and Systems XIX, Daegu, Republic of Korea, July 10-14, 2023*, 2023.
[55] Y. Ze, G. Zhang, *et al.*, "3d diffusion policy: Generalizable visuomotor policy learning via simple 3d representations," in *Proceedings of Robotics: Science and Systems (RSS)*, 2024.
[56] T.-W. Ke, N. Gkanatsios, *et al.*, "3d diffuser actor: Policy diffusion with 3d scene representations," in *Conference on Robot Learning*, PMLR, 2025, pp. 1949–1974.
[57] Q. Feng, H. Li, *et al.*, "Language-guided object-centric diffusion policy for collision-aware robotic manipulation," in *International Conference on Robotics and Automation (ICRA)*, 2025.
[58] C. Hao, K. Lin, *et al.*, "Language-guided manipulation with diffusion policies and constrained inpainting," *CoRR*, vol. abs/2406.09767, 2024.
[59] K. Mizuta and K. Leung, "Cobl-diffusion: Diffusion-based conditional robot planning in dynamic environments using control barrier and lyapunov functions," *CoRR*, vol. abs/2406.05309, 2024.
[60] D. Wang, S. Hart, *et al.*, "Equivariant diffusion policy," in *Conference on Robot Learning*, PMLR, 2025, pp. 48–69.
[61] J. Yang, Z. Cao, *et al.*, "Equibot: Sim (3)-equivariant diffusion policy for generalizable and data efficient learning," in *CoRL 2024 Workshop on Whole-body Control and Bimanual Manipulation: Applications in Humanoids and Beyond.*
[62] Z. Huang, Y. Lin, *et al.*, "Subgoal diffuser: Coarse-to-fine subgoal generation to guide model predictive control for robot manipulation," in *IEEE International Conference on Robotics and Automation*, IEEE, 2024.
[63] P. M. Scheikl, N. Schreiber, *et al.*, "Movement primitive diffusion: Learning gentle robotic manipulation of deformable objects," *IEEE Robotics Autom. Lett.*, vol. 9, no. 6, 2024.
[64] G. Li, Z. Jin, *et al.*, "Prodmp: A unified perspective on dynamic and probabilistic movement primitives," *IEEE Robotics Autom. Lett.*, vol. 8, no. 4, 2023.
[65] K. Saha, V. R. Mandadi, *et al.*, "EDMP: ensemble-of-costs-guided diffusion for motion planning," in *IEEE International Conference on Robotics and Automation*, IEEE, 2024.
[66] T. Power, R. Soltani-Zarrin, *et al.*, "Sampling constrained trajectories using composable diffusion models," in *IROS 2023 Workshop on Differentiable Probabilistic Robotics: Emerging Perspectives on Robot Learning*, 2023.
[67] W. Xiao, T.-H. Wang, *et al.*, "Safediffuser: Safe planning with diffusion probabilistic models," in *The Thirteenth International Conference on Learning Representations*.
[68] A. Dastider, H. Fang, *et al.*, "Apex: Ambidextrous dual-arm robotic manipulation using collision-free generative diffusion models," in *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2024, pp. 9526–9533.
[69] J. Carvalho, M. Baierl, *et al.*, "Conditioned score-based models for learning collision-free trajectory generation," in *NeurIPS 2022 Workshop on Score-Based Methods*, 2022.
[70] Y. Luo, C. Sun, *et al.*, "Potential based diffusion motion planning," in *International Conference on Machine Learning*, OpenReview.net, 2024.
[71] S. Huang, Z. Wang, *et al.*, *Diffusion-based generation, optimization, and planning in 3d scenes*, 2023.
[72] H. Huang, B. Sundaralingam, *et al.*, "Diffusionseeder: Seeding motion optimization with diffusion for rapid motion planning," in *8th Annual Conference on Robot Learning*, 2024.
[73] A. Fishman, A. Murali, *et al.*, "Motion policy networks," in *Proceedings of the 6th Conference on Robot Learning (CoRL)*, 2022.
[74] S. Teng, X. Hu, *et al.*, "Motion planning for autonomous driving: The state of the art and future perspectives," *IEEE Transactions on Intelligent Vehicles*, vol. 8, no. 6, 2023.
[75] J. Lu, K. Wong, *et al.*, "Scenecontrol: Diffusion for controllable traffic scene generation," in *IEEE International Conference on Robotics and Automation*, IEEE, 2024.
[76] K. Kondo, A. Tagliabue, *et al.*, "CGD: constraint-guided diffusion policies for UAV trajectory planning," *CoRR*, vol. abs/2405.01758, 2024.
[77] B. Yang, H. Su, *et al.*, "Diffusion-es: Gradient-free planning with diffusion for autonomous and instruction-guided driving," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2024.
[78] J. Liu, M. Stamatopoulou, *et al.*, "Dipper: Diffusion-based 2d path planner applied on legged robots," in *IEEE International Conference on Robotics and Automation*, IEEE, 2024.
[79] M. Stamatopoulou, J. Liu, *et al.*, "Dippest: Diffusion-based path planner for synthesizing trajectories applied on quadruped robots," in *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2024, pp. 7787–7793.
[80] J. Urain, A. Mandlekar, *et al.*, *Deep generative models in robotics: A survey on learning from multimodal demonstrations*, 2024.
[81] M. Zucker, N. Ratliff, *et al.*, "Chomp: Covariant hamiltonian optimization for motion planning," *The International Journal of Robotics Research*, vol. 32, no. 9-10, 2013.
[82] H. Attias, "Planning by probabilistic inference," in *Proceedings of the Ninth International Workshop on Artificial Intelligence and Statistics*, vol. R4, PMLR, 2003.

[83] M. Toussaint, "Robot trajectory optimization using approximate inference," in *ICML*, Association for Computing Machinery, 2009.
[84] H. Yu and Y. Chen, "A gaussian variational inference approach to motion planning," *IEEE Robotics Autom. Lett.*, vol. 8, no. 5, 2023.
[85] Y. Song and S. Ermon, "Generative modeling by estimating gradients of the data distribution," in *NeurIPS*, 2019.
[86] A. Q. Nichol and P. Dhariwal, "Improved denoising diffusion probabilistic models," in *ICML*, PMLR, 2021.
[87] J. Sohl-Dickstein, E. A. Weiss, *et al.*, "Deep unsupervised learning using nonequilibrium thermodynamics," in *ICML*, JMLR.org, 2015.
[88] C. Luo, *Understanding diffusion models: A unified perspective*, 2022.
[89] A. Ajay, Y. Du, *et al.*, "Is conditional generative modeling all you need for decision making?" In *The Eleventh International Conference on Learning Representations*, OpenReview.net, 2023.
[90] J. Ma, T. Hu, *et al.*, "Elucidating the design space of classifier-guided diffusion generation," in *International Conference on Learning Representations*, OpenReview.net, 2024.
[91] Z. Zhong, D. Rempe, *et al.*, "Guided conditional diffusion for controllable traffic simulation," in *IEEE International Conference on Robotics and Automation*, IEEE, 2023.
[92] J. Song, C. Meng, *et al.*, "Denoising diffusion implicit models," in *International Conference on Learning Representations*, OpenReview.net, 2021.
[93] P. Kicki, P. Liu, *et al.*, "Fast kinodynamic planning on the constraint manifold with deep neural networks," *IEEE Trans. Robotics*, vol. 40, 2024.
[94] C. de Boor, "Package for calculating with b-splines," *SIAM Journal on Numerical Analysis*, vol. 14, no. 3, 1977.
[95] C.-K. Shene, *Derivatives of a B-spline Curve — pages.mtu.edu*, https://pages.mtu.edu/~shene/COURSES/cs3621/NOTES/spline/B-spline/bspline-derv.html, [Accessed 24-09-2024].
[96] P. Kicki, D. Tateo, *et al.*, "Bridging the gap between learning-to-plan, motion primitives and safe reinforcement learning," in *8th Annual Conference on Robot Learning*, 2024.
[97] L. A. Piegl and W. Tiller, *The NURBS Book*. Springer, 1995.
[98] A. Paraschos, C. Daniel, *et al.*, "Using probabilistic movement primitives in robotics," *Auton. Robots*, vol. 42, no. 3, 2018.
[99] W. Tiller, "A unified and elegant derivation of bézier, b-spline, and other CAGD concepts: Bézier and b-spline techniques," *Comput. Aided Des.*, vol. 36, no. 1, 2004.
[100] E. Perez, F. Strub, *et al.*, "Film: Visual reasoning with a general conditioning layer," in *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence*, AAAI Press, 2018.
[101] A. Paszke *et al.*, "Pytorch: An imperative style, high-performance deep learning library," in *NeurIPS*, Curran Associates, Inc., 2019.
[102] L. Pineda, T. Fan, *et al.*, "Theseus: A library for differentiable nonlinear optimization," in *Advances in Neural Information Processing Systems*, 2022.
[103] J. Sola, J. Deray, *et al.*, "A micro lie theory for state estimation in robotics," *arXiv preprint arXiv:1812.01537*, 2018.
[104] A. Millane, H. Oleynikova, *et al.*, "Nvblox: Gpu-accelerated incremental signed distance field mapping," in *IEEE International Conference on Robotics and Automation*, IEEE, 2024.
[105] I. A. Sucan, M. Moll, *et al.*, "The open motion planning library," *IEEE Robotics Autom. Mag.*, vol. 19, no. 4, 2012.
[106] E. Coumans, "Bullet physics simulation," in *Special Interest Group on Computer Graphics and Interactive Techniques Conference, SIGGRAPH*, ACM, 2015.
[107] P. Virtanen, R. Gommers, *et al.*, "Scipy 1.0: Fundamental algorithms for scientific computing in python," *Nature methods*, vol. 17, no. 3, pp. 261–272, 2020.
[108] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *International Conference on Learning Representations*, 2015.
[109] K. Sohn, H. Lee, *et al.*, "Learning structured output representation using deep conditional generative models," in *Advances in Neural Information Processing Systems*, 2015.
[110] D. Friedman and A. B. Dieng, "The vendi score: A diversity evaluation metric for machine learning," *Trans. Mach. Learn. Res.*, 2023.
[111] A. Li, Z. Ding, *et al.*, *Diffusolve: Diffusion-based solver for non-convex trajectory optimization*, 2024.
[112] J. K. Christopher, S. Baek, *et al.*, "Constrained synthesis with projected diffusion models," in *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024.
[113] J. Bradbury, R. Frostig, *et al.*, *JAX: Composable transformations of Python+NumPy programs*, version 0.3.13, 2018.
[114] M. J. Bency, A. H. Qureshi, *et al.*, "Neural path planning: Fixed time, near-optimal path generation via oracle imitation," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, IEEE, 2019.

BIOGRAPHIES

João Carvalho is a Postdoctoral Researcher at the Intelligent Autonomous Systems Lab of the Technical University of Darmstadt, where he obtained his Ph.D. in January 2025. He previously obtained his M.Sc. degree in Computer Science from the University of Freiburg. His research interests are centered around learning approaches for robot manipulation, from generative models for robot motion planning, imitation learning, and grasping.

An Thai Le is a Ph.D. candidate at the Intelligent Autonomous Systems Lab of the Technical University of Darmstadt. He obtained his M.Sc. degree from the University of Stuttgart (Germany). His research interests are related to scaling planning methods to long-horizon, high-dimensional state-space, number of plans, and number of agents, using batch planning methods.

Piotr Kicki is a post-doc in the robotics team at IDEAS and an assistant professor at the Institute of Robotics and Machine Intelligence at Poznan University of Technology. He received his B.Eng. and M.Sc. degrees in automatic control and robotics from Poznan University of Technology, Poland in 2018 and 2019, respectively. He completed his Ph.D. from the same university in 2024. His primary research interests center on the application of machine learning to improve robot motion planning and control.

Dorothea Koert is an Independent Research Group Leader of the interdisciplinary BMBF junior research group IKIDA, which started in October 2020. She completed her Ph.D. from the Technical University of Darmstadt in February 2020. In 2019 she was awarded the AI-Newcomer award by the German Society for Computer Science (GI). During her Ph.D. she has worked on imitation learning and interactive reinforcement learning, for autonomous and semi-autonomous acquisition of motion skill libraries in human-robot collaboration.

Jan Peters is a full professor (W3) for Intelligent Autonomous Systems at the Computer Science Department of the Technical University of Darmstadt, department head of the research department on Systems AI for Robot Learning (SAIROL) at the German Research Center for Artificial Intelligence, and a founding research faculty member of The Hessian Center for Artificial Intelligence. He has received the Dick Volz Best 2007 US Ph.D. Thesis Runner-Up Award, RSS - Early Career Spotlight, INNS Young Investigator Award, and IEEE Robotics & Automation Society's Early Career Award, as well as numerous best paper awards. He received an ERC Starting Grant and was appointed an IEEE fellow, AIAA fellow, and ELLIS fellow.