Score-Based Generative Models as Trajectory Priors for Motion Planning

Score-basierte generative Modelle als Vorwissen über Trajektorien in der Bewegungsplanung
Master thesis by Mark Baierl
Date of submission: January 23, 2023

1. Review: Prof. Dr. Jan Peters
2. Review: Joao Carvalho
Darmstadt

TECHNISCHE
UNIVERSITÄT
DARMSTADT

Erklärung zur Abschlussarbeit gemäß § 22 Abs. 7 und § 23 Abs. 7 APB der TU Darmstadt

Hiermit versichere ich, Mark Baierl, die vorliegende Masterarbeit ohne Hilfe Dritter und nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die Quellen entnommen wurden, sind als solche kenntlich gemacht worden. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Mir ist bekannt, dass im Fall eines Plagiats (§ 38 Abs. 2 APB) ein Täuschungsversuch vorliegt, der dazu führt, dass die Arbeit mit 5,0 bewertet und damit ein Prüfungsversuch verbraucht wird. Abschlussarbeiten dürfen nur einmal wiederholt werden.

Bei der abgegebenen Thesis stimmen die schriftliche und die zur Archivierung eingereichte elektronische Fassung gemäß § 23 Abs. 7 APB überein.

Bei einer Thesis des Fachbereichs Architektur entspricht die eingereichte elektronische Fassung dem vorgestellten Modell und den vorgelegten Plänen.

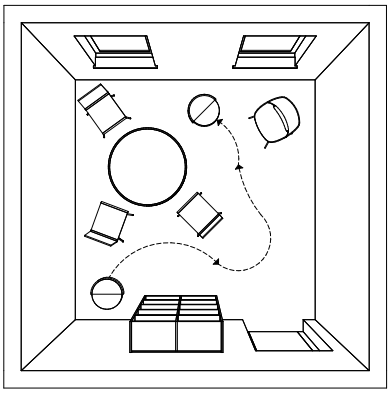
Darmstadt, 23. Januar 2023

M. Baierl

Contents

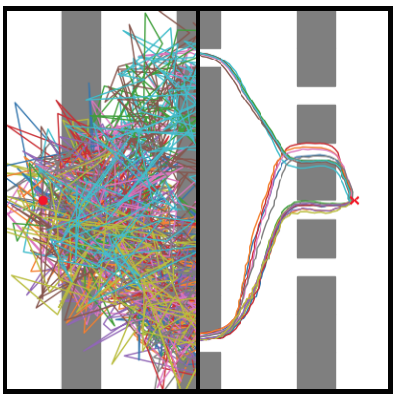
1 Introduction

Recent advances in new architectures and training methods of diffusion and score-based models (SBMs) have shown impressive results in image and text-to-image generation [50, 11, 46, 20, 42]. One area where these models have not been fully explored is robotics [15, 55]. Since whole robot trajectories have high dimensionality and their distribution is often multi-modal, it is worth investigating whether or not SBMs can be used as part of an intelligent robot system.



(a) Example of planning.

An important task is to plan collision-free movements (paths) between two locations. With access to an environment map, sampling-based methods such as variants of Probabilistic Road Maps (PRM) [19] and Rapidly exploring Random Trees (RRT) [26] or optimization-based methods such as STOMP [18], are commonly used. With repeated planning in the same or even different environments, it is natural to somehow reuse previously collected information. This hopefully speeds up future planning in similar environments. Several works have proposed neural motion planners that encode environment and motion information using neural networks, which are subsequently used as priorities for the planners [38, 40].



(b) SBM denoising mockup.

In this work, I investigate whether SBMs can be leveraged to generate trajectories and whether they are useful

improvements over previous methods. I conduct experiments to find out how they function as priors for StochGPMP [54], whether they improve performance over standalone StochGPMP, and if so, how significant the improvement is.

Since SBMs have such potential in generating high-dimensional data, I will experiment with higher-dimensional search spaces to see if SBMs can generate coherent trajectories in them and if the sampling time is reasonable. Especially in real robot systems, which often have seven or more degrees of freedom, many planning methods become very slow or even infeasible, so time is a big factor.

As you will see, SBMs have a place in motion planning as priors to optimization-based planners like StochGPMP. Their ability to model multi-modal, high-dimensional data makes them a valuable tool for improving planning time in tricky, high-dimensional environments. They can be conditioned on different features to generalize to many different tasks in many different environments.

2 Foundations

Since I am using SBMs for motion planning, I will first introduce some common motion planning techniques that are currently in use. Then I talk a bit about what kind of priors can be used by these motion planners. I will conclude this chapter with a little background on point cloud encoders and signed distance fields (SDF), which are referenced in later chapters.

2.1 Motion Planning

We consider two methods for motion planning [24, 25], sample-based and optimization-based. Both can benefit from priors derived from previously generated data to increase speed and/or quality, as we will see later.

2.1.1 Sampling-based motion planning

Sampling-based motion planning uses randomly sampled collision-free configurations and connects them. The major advantage of these methods is that they always find a path that connects the start and target positions. However, this tends to scale poorly for higher dimensions and they tend to produce non-smooth trajectories. This would result in jerky robot motion, so these approaches require post-processing. There are many ways to perform sampling-based motion planning, here is an overview.

Rapidly-Exploring Random Trees (RRT)

Rapidly-exploring random trees (RRT) [26] iteratively generates a tree-like structure that explores the search space. The root of the tree is usually the starting node of the planning problem. The basic version of RRT uses a uniform prior such that a uniformly random point x_{rand} in the search space is selected at each iteration. Then the closest node to x_{rand}, x_{close}, is selected and connected to it. Often a δ value is used, which describes the maximum distance between two nodes. In this case, a point with a distance of δ x_{near} on a straight line between x_{close} and x_{rand} is inserted into the tree rather than the random point itself. At some point, the target can be added to the tree. Then RRT is terminates.

A variant of RRT worth mentioning is RRT* [23]. In RRT* we have the ability to modify existing connections within the tree. Specifically, after adding an x_{next} to the tree, we check if nodes in a radius r around x_{next} would benefit from being connected to it instead. In this case, the parent of the nearby node would be set to x_{next}. This algorithm has the advantage of finding far better solutions than RRT. The solutions are usually smoother and shorter. The disadvantage, however, is that it is unclear when this algorithm should terminate. In RRT we are close enough to the goal at one point, but in RRT* we can always add nodes that further improve the trajectory. This means that RRT* finds the optimal trajectory at infinite time, but is much slower than RRT.

One method that attempts to improve RRT* is called Informed RRT* [9]. The idea is to restrict the sample range to an ellipse after an initial solution is found. As the solution improves, the ellipse is gradually reduced in size. This dramatically increases the probability that a randomly selected point will improve the current best solution.

Probabilistic Roadmaps (PRM)

Probabilistic Roadmaps (PRM) [19] works by first selecting random nodes that cover the entire search space. Then, a network is created between them by connecting each point to the k nearest other nodes. The advantage is that this network can be reused to query paths between any points in the network. In [19], the nodes are collision-free configurations and the edges are feasible paths between nodes. These paths are generated in the first phase by simple local planners.

2.1.2 Optimization-based motion planning

In optimization-based motion planning, we optimize the entire trajectory as one rather than many connected points, as in sampling-based approaches. Optimization-based methods optimize a particular initial trajectory (-distribution) and can use either gradient or stochastic optimization. They also have the advantage of being able to create smooth trajectories that avoid collisions by including these features as costs to be optimized. However, they can have a problem with high-dimensional data and converge to local minima. The methods often use an uninformed initial distribution at the beginning of the optimization, but as we will see later, there are ways to incorporate prior knowledge to choose a better initial distribution.

Covariant Hamiltonian Optimization for Motion Planning (CHOMP)

Covariant Hamiltonian Optimization for Motion Planning (CHOMP) [44] uses covariant gradient optimizations to transform a possible naive first guess into a feasible collision-free trajectory. Here, a simple straight line is often used as the first guess. The cost of a trajectory $\boldsymbol{\tau}$, which is needed for the optimization according to [44, 63] is given by

$$\mathcal{C}(\boldsymbol{\tau}) = \mathcal{C}(\boldsymbol{\tau})_{prior} + \mathcal{C}(\boldsymbol{\tau})_{obs} \tag{2.1}$$

with

$$\mathcal{C}(\boldsymbol{\tau})_{prior} = \frac{1}{2}\|\mathbf{K}\boldsymbol{\tau} + \boldsymbol{e}\|^2$$

where $\mathbf{K}$ is a finite difference matrix and $\boldsymbol{e}$ is a vector that handles the boundary conditions. $\mathcal{C}(\boldsymbol{\tau})_{obs}$ can be any differentiable value, while $\mathcal{C}(\boldsymbol{\tau})_{prior}$ enforces the smoothness of the trajectory.

The update rule

$$\boldsymbol{\tau}_{k+1} = \boldsymbol{\tau}_k - \frac{1}{\lambda}M^{-1}\nabla\mathcal{C}(\boldsymbol{\tau}_k)$$

can derived through a first order taylor expansion as described in [44, 63].

CHOMP converges quickly to a locally optimal trajectory. Depending on the problem, this may be good enough. However, in many cases, a little more exploration is needed that could lead to a solution closer to the globally optimal trajectory. Since CHOMP works with a single trajectory, we also get only a single solution.

Stochastic Trajectory Optimization for Motion Planning (STOMP)

Stochastic Trajectory Optimization for Motion Planning (STOMP) [17] considers motion planning as a stochastic optimization problem to find a smooth trajectory that minimizes cost. STOMP can be viewed as a stochastic version of CHOMP, but instead of updating the trajectory using only the gradient of the cost, the update is computed using a softmax distribution. It is also iterated to convergence, but at each iteration N noisy trajectories $\tilde{\boldsymbol{\tau}}_1 ... \tilde{\boldsymbol{\tau}}_N$ are generated with $\boldsymbol{\tau} + \epsilon_n$, where $\epsilon_n \sim \mathcal{N}(0, (\mathbf{K}^T\mathbf{K})^{-1})$. The used noises ϵ are then weighted by a softmax over their respective costs to obtain the gradient. So given a softmax distribution

$$P(\tilde{\boldsymbol{\tau}}_n, i) = \frac{\exp\left(-(1/\lambda)c(\tilde{\boldsymbol{\tau}}_{n,i})\right)}{\sum_{k=1}^{N} \exp\left(-(1/\lambda)c(\tilde{\boldsymbol{\tau}}_{k,i})\right)}$$

we get the gradient

$$\nabla \boldsymbol{\tau}_i = \mathbf{M} \sum_{n=1}^{N} P(\tilde{\boldsymbol{\tau}}_n, i)\epsilon_{n,i},$$

where $\mathbf{M} = (\mathbf{K}^T\mathbf{K})^{-1}$ with each column scaled such that the maximum element is $1/H$, and the update rule

$$\boldsymbol{\tau}_{k+1} = \boldsymbol{\tau}_k + \nabla \boldsymbol{\tau}_k.$$

Then, the cost for the entire trajectory can be evaluated to see if convergence is achieved. One advantage of STOMP is that the costs do not have to be differentiable.

Gaussian Process Motion Planning (GPMP)

Gaussian Process Motion Planner (GPMP) [28, 29] uses a Gaussian Process Prior, and likelihoods of the exponential family. It does a maximum-a-posteriori (MAP) trajectory optimization to find a single trajectory, by doing gradient optimization with Gauss-Newton or Levenberg-Marquardt. One disadvantage of CHOMP and STOMP that GPMP overcomes is the need for a very large number of support states. Gaussian processes (GP) represent trajectories as continuous-time functions, so they can be queried at any time while still using viewer support states. Given a vector-valued mean function $\boldsymbol{\mu}(t)$ and a matrix-valued covariance function $\mathbf{K}(t, t')$, we obtain the vector-valued GP

$$\boldsymbol{\theta}(t) \sim \mathcal{GP}\left(\boldsymbol{\mu}(t), \mathbf{K}(t, t')\right).$$

Since this is a GP we can say that for any collection of times $\boldsymbol{t} = \{t_0, ..., t_N\}$, $\boldsymbol{\theta}$ has a joint Gaussian distribution

$$\boldsymbol{\theta} \sim \mathcal{N}\left(\boldsymbol{\mu}, \mathbf{K}\right).$$

The vectors $\boldsymbol{\theta}_0, ..., \boldsymbol{\theta}_N$ are support states that parameterize the continuous-time trajectory $\boldsymbol{\theta}(t)$. Since this is a GP we automatically get a prior over the space of trajectories

$$p(\boldsymbol{\theta}) \propto \exp\{-\frac{1}{2}\|\boldsymbol{\theta} - \boldsymbol{\mu}\|^2_{\mathbf{K}}\} \tag{2.2}$$

where $\|\boldsymbol{\theta} - \boldsymbol{\mu}\|^2_{\mathbf{K}} = (\boldsymbol{\theta} - \boldsymbol{\mu})^T \boldsymbol{K}^{-1}(\boldsymbol{\theta} - \boldsymbol{\mu})$ is the Mahalobis distance. This prior encourages smoothness, so it achieves a similar function as the finite difference matrix $\mathbf{K}$ in CHOMP and STOMP. The cost function looks very similar to the one in equation 2.1:

$$\mathcal{C}(\boldsymbol{\theta}(t)) = \mathcal{C}(\boldsymbol{\theta}(t))_{prior} + \mathcal{C}(\boldsymbol{\theta}(t))_{obs}.$$

The difference is that

$$\mathcal{C}(\boldsymbol{\theta}(t))_{prior} = \lambda \mathcal{C}(\boldsymbol{\theta}(t))_{gp} = \frac{\lambda}{2}\|\boldsymbol{\theta} - \boldsymbol{\mu}\|^2_{\mathbf{K}}$$

which we get by taking the negative logarithm of equation 2.2. We get the update rule

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \frac{1}{\eta}\mathbf{K}\nabla\mathcal{C}(\boldsymbol{\theta}_k), \nabla\mathcal{C}(\boldsymbol{\theta}_k) = \lambda\mathbf{K}^{-1}(\boldsymbol{\theta}_k - \boldsymbol{\mu}) + \nabla\mathcal{C}(\boldsymbol{\theta}_k)_{obs}$$

again through Taylor expansion as described in [28]. They note that this can be interpreted as a generalization of the update rule in CHOMP.

GPMP achieves really good results, but similarly to CHOMP it can only be used if we have access to the gradient of $\mathcal{C}(\boldsymbol{\theta}(t))_{obs}$, which may not be the case. We can condition the GP on a fixed set of time-parameterized states, e.g. the first and last, to compute the posterior mean at any time of interest.

Stochastic Gaussian Process Motion Planning (StochGPMP)

Stochastic GPMP (StochGPMP) [54] combines elements of STOMP and GPMP. StochGPMP still uses the GP prior, but instead of updating a current trajectory with it, N trajectories $\{\boldsymbol{\tau}_n\}_{n=1}^N \sim \mathcal{GP}\left((\boldsymbol{\mu}, \mathbf{K})\right)$ are sampled at each iteration. Just as in STOMP, the sampled trajectories are then evaluated and the costs are combined into softmax weights:

$$w_n = \frac{\exp\left(-(1/\lambda)\mathcal{C}(\boldsymbol{\tau}_n)\right)}{\sum_{k=1}^N \exp\left(-(1/\lambda)\mathcal{C}(\boldsymbol{\tau}_k)\right)}.$$

These weights are then used to update the mean of the proposal distribution (the GP) with

$$\boldsymbol{\mu} \leftarrow \boldsymbol{\mu} + \gamma \sum_{n=0}^{N} w_n(\boldsymbol{\tau}_n - \boldsymbol{\mu}).$$

The advantage is that costs do not have to be differentiable, while still making use of the GP prior. This however takes longer to converge than GPMP, since it cannot simply follow the gradient, but instead has to explore the space.

2.2 Priors for Motion Planning

The advantage of using prior knowledge in motion planning is obvious. If we have already created many trajectories, why would we start from scratch when planning another trajectory instead of using a more informed method to sample or initialize the optimization. Below is a curated list of ways to incorporate better prior knowledge into motion planning.

2.2.1 Priors for Sampling-Based Motion Planning

Normally, sampling-based motion planners have to search the entire search space uniformly. They always find a solution this way, but it can take a long time, especially if the dimensions of the search space are very large. The hope is that we can use prior knowledge to reduce the time it takes to find what we want to find.

Neural Motion Planning

The idea is not to sample a uniformly random point as in RRT, for example, but to learn a conditioned one-step neural planner that outputs the next point. Previous work [38, 40, 39, 41, 57] uses dropout to ensure nondeterminism and generate more than one solution.

Bency et al. [2] trains an oracle network as a (multilayer) Long Short-Term Memory (LSTM) [12] network for a given environment. At each planning step, the oracle network takes the current state and the hidden state of the LSTM and generates the next point to be queried. This idea is very close to [39], but without explicit use of an environment encoder.

Strudel et al. [52] uses Reinforcement Lerning (RL) to learn a neural motion planning policy based on environment observations in the form of point clouds, which are then processed using a PointNet architecture [37] to create a feature vector of the end-effector's current local environment. However, since the planner is a policy that outputs the next configuration, there may be failure cases where the agent gets stuck. They only show experiments with a "robot" that is a free-floating object, leaving robot arms to do future work.

Ichnowski et al. [14] uses Deep Learning to learn an initial trajectory to accelerate motion planning when grasping an object. Their initial trajectory is used to initialize a sequential quadratic programming (SQP) problem, as used in TrajOpt [48], and shows an order of magnitude improvement.

Local Samplers

Chamzas et al. [3, 4] create local samplers for difficult regions of the configuration space, such as narrow passages. Local primitives attempt to capture the features of the workspace by decomposing the workspace into multiple regions (through an engineered solution).

The local primitive associated with a local sampler in their work is a Gaussian mixture model where each Gaussian has a fixed covariance and the categorical distribution is a uniform distribution.

Optimal Sampling distribution

Cheng et al. [6] computes an optimal sampling distribution for use in a sampling-based motion planner. The search space is a voxelized 3D grid where each voxel represents the probability that this point belongs to a successful path, providing a basic heuristic for adopting this point in a sampling-based planner such as RRT. They show results for a robot with 31 degrees of freedom operating in a shelf environment. They learn a heuristic basis function $f : \mathcal{S} \rightarrow \mathcal{A}$, where $S \subseteq \mathbb{R}^{64642}$ is a 3D voxel map with 2 channels (state of the robot, start and goal positions, and binary occupancy grid) and $\mathcal{A} \subseteq \mathbb{R}^{10\times10\times10}$. f is a ResNet-18 network. This gives only a rough representation of the environment.

Learned Rejection Sampling

Zhang et al. [62] learns rejection sampling distributions for sampling-based motion planning from experience by formulating the problem as a reinforcement learning problem. The action space is 2-dimensional - accept or reject a point to add it to the tree.

Transformer Motion Planning

Johnson et al. [16] proposes Motion Planning Transformers. This is not a trajectory generator, but rather a method for outputting a region in which to sample points for RRT/IRRT. Problem with MPNET: “However, these approaches assume a fixed size of the input environment map and often require redefinition of network architectures and re-training for different map sizes.

2.2.2 Priors for Optimization-Based Motion Planning

There are many ways to use previous data to learn a prior distribution that supports optimization-based motion planners. The question is which distribution we use to model the prior and how useful it is as a prior for multi-modal high-dimensional data.

Behavior Cloning (BC)

Several works use behavior cloning (BC), also known as learning from demonstration (LfD), to encode trajectory priors [43, 22]. They fit a density model to a set of demonstrations and then use them as priors for motion optimization, but they do not capture multiimodal trajectories well in high-dimensional spaces.

Probabilistic Movement Primitives (ProMP)

Koert et al. [22] first learns a Probabilistic Movement Primitive (ProMP) [32] from human demonstrations. In the presence of obstacles, it defines a cost for obstacle avoidance. Then, the new parameters of the ProMP are computed (offline) such that the expected cost under this distribution and the Kullback-Leibler divergence to the ProMP encoding the demonstrations is minimized. This optimization is performed using Relative Entropy Policy Search (REPS) [34]. Due to the ProMP, this method only encodes unimodal trajectory distributions.

Energy Based Models (EBM)

Urain et al. [54] learned energy-based models (EBMs) that represent various costs. These are easily combined and then used as costs for their Stochastic Gaussian Process Motion Planning (StochGPMP) algorithm. They probably did not learn the trajectory as an EBM because sampling from this EBM might be too difficult. Representing the entire trajectory distribution with a single EBM could be difficult due to high dimensionality. [54].

In Implicit Behavioral Cloning [8] they use Autoregressive Derivative-Free Optimization, where they have to construct one EBM per dimension and query one degree of freedom at a time. From a philosophical point of view, this seems less coherent than simultaneously sampling all degrees of freedom, although I have no data to confirm this. Their EBMs are learned by (one-step) denoising, which is similar to score-based models. Also, their phase-based EBMs do not encode temporal relationships between successive points, which can lead to non-smooth trajectories. They counter this by using costs to smooth trajectories.

Sampling on Constraint Manifolds

Ortiz-Haro et al. [31] learns to sample from a constraint manifold, by combining generative models with local optimization to project to the constraint manifold. They use an image representation to generalize to different numbers of objects (and obstacles) and shapes. To scale to large problems, they explore factorization for sampling using a factor graph formulation. Factorization helps to sample from a distribution with disconnected supports. It is unclear if this problem only occurs for GANs, or it also occurs for SBMs.

2.3 Pointcloud Encoder

There are many ways to process 3D data. Voxelization or rendering is often used, but a simpler representation is a point cloud. A point cloud is an unordered list of points that can be returned from 3D sensors such as lidar. In the future work chapter 6 I will refer to some of this.

2.3.1 PointNet

The motivation for PointNet [37] was to have a deep-net architecture that uses a raw point cloud as input, rather than voxelization or rendering. The architecture aims to process unordered points that can interact with each other while being invariant to certain transformations such as rotations and translations. This is desirable because the point cloud of a cup should still be classified as such even if it is lying on its side. PointNet applies input and feature transformations, mixed with some Multi Layer Perceptron (MLP) layers in between. The global features of the point cloud can be extracted after applying a max-pool layer.

2.3.2 Dynamic Graph CNN (DGCNN)

Dynamic Graph CNN (DGCNN) [58] seeks to enrich pointcloud networks with insights from CNNs for image analysis. To this end, Wang et al. [58] has introduced EdgeConv, which uses a local neighborhood graph to apply localized convolution-like operations similar to a neural graph network. This neighborhood changes after each layer of the

network so that the proximity in the feature space is different from the proximity in the input space, making this a non-local operation.

2.3.3 Vector Neural Networks (VNN)

Vector Neural Network (VNN) [7] aims to be a lightweight framework for building SO(3) equivariant and invariant point cloud networks. The main change compared to PointNet is the use of vector neurons (VN), replacing scalar neurons with 3-vector neurons. They also introduce a learned generalization of ReLU that can handle 3D inputs. Instead of activating over 0, as ReLU does, their nonlinearity learns a direction k that it can use to check whether the input feature q is in the half-space defined by k. Since Vector Neurons are a general framework, they can be used in any number of existing architectures.

Using the VN framework, Deng et al. [7] have shown that extending the architecture of other pointcloud networks, such as Pointnet and DGCNN, with VNs makes them more robust to rotations and permutations.

2.4 Deep Signed Distance Functions (SDF)

Signed Distance Functions (SDF)[1] are commonly used in motion planning, e.g. CHOMP, GPMP. SDFs are functions that map a coordinate in a given environment to the shortest distance to the object for which the SDF applies. The distance can be, for example, to obstacles in an environment or to a single object that we want to represent by a continuous function. Values inside obstacles/objects are negative and represent the shortest distance to a boundary, hence the "Signed" in Signed Distance Function.

In Power and Berenson [36], a Signed distance field is used as input to a variational encoder. The generated latent vector is then used to condition a normalization flow to produce a control sequence. Like many other approaches, this approach assumes that the SDF is accessible to the environment it represents. There are approaches to generating an SDF from another environment representation such as a point cloud. Deep SDF [33] models an SDF as an implicit continuous function represented by a learned deep network. This network is learned based on probabilistic autodecoders. They focus on creating the highest quality SDF possible for individual objects rather than environments.

[1] SDF often refers to Signed Distance Fields, which are the output of the Signed Distance Function

Ortiz et al. [30] focuses on creating full SDFs of large scenes in real time. Their system, iSDF, uses a stream of posed depth images captured by a moving camera. The SDF itself is modeled by a multilayer perceptron (MLP) that maps a 3D coordinate to the signed distance value at that point. This approach is useful for an agent that is dropped into a new environment and needs to explore it.

Neural descriptor fields [49] are an approach to encode both points and relative poses between an object and a target. They use an occupancy network that takes both an object point cloud and a query point as input. The features after each layer are linked together to produce a combined feature vector. To construct the full neural pose descriptor field, many points from a query point cloud are passed as query points, and the resulting features are concatenated again.

SDFs can be used in many ways. One of the main uses is to apply their gradient during planning or fine tuning to push trajectories away from obstacles.

In chapter 6, you will also see a way to use the SDF loss function to generate features that encode the environment for use in other methods.

3 Score-Based Models as Trajectory Generative Models

Several generative modeling techniques such as Generative Adversarial Networks (GAN) [10], Variational Auto Encoders (VAE) [21], and Normalizing Flows (NFs) [45] are trained to maximize the log-likelihood of the data, and sampling is performed by applying deterministic transformations to a random variant of a simple distribution. Instead, SBMs are implicit models [13, 56, 50] that perturb the data with diffusion and learn to reconstruct it by denoising using the score function-the gradient of the log-likelihood w.r.t. of the input. The sampling is done by iteratively transforming a sample from a simple distribution according to the (parameterized) score function $\boldsymbol{s}_{\boldsymbol{\theta}}(\boldsymbol{\tau}_t, t) \approx \nabla_{\boldsymbol{\tau}_t} \log p(\boldsymbol{\tau}_t, t)$, where $t \in [0, 1]$ is the time of the denoising step.

The goal is to move an initial noisy sample $\boldsymbol{\tau}_1$ to a sample from the data distribution $\boldsymbol{\tau}_0$. SBMs can be trained with Denoising Score Matching (DSM) by minimizing

$$\mathcal{L}_{\text{score}}(\boldsymbol{\theta}) = \mathbb{E}_{\boldsymbol{\tau}\sim p(\boldsymbol{\tau}), t\sim\mathcal{U}(0,T)} \left[\lambda(t)\mathbb{E}_{\boldsymbol{\tau}_t\sim p(\boldsymbol{\tau}_t|\boldsymbol{\tau},t)} \left[\|\boldsymbol{s}_{\boldsymbol{\theta}}(\boldsymbol{\tau}_t, t) - \nabla_{\boldsymbol{\tau}_t} \log p(\boldsymbol{\tau}_t|\boldsymbol{\tau}, t)\|^2\right]\right], \quad (3.1)$$

where $p(\boldsymbol{\tau}_t|\boldsymbol{\tau}, t) = \mathcal{N}\left(\boldsymbol{\tau}_t; \boldsymbol{\tau}, \sigma^2(t)\mathbf{I}\right)$ is the density of the perturbed trajectory at time t, $\lambda(t) = (a^{2t} - 1)/(2 \log a)$ and $\sigma(t) = \sqrt{\lambda(t)}$. This loss is an adapted form of the one in Song et al. [51] to fit trajectories. In [15] a discrete-time denoising diffusion probabilistic model is used instead [11], but from now on I will refer only to the former setup.

3.1 Motion Planning as Inference

Let $\boldsymbol{s} \in \mathcal{S} \subseteq \mathbb{R}^d$ encode the state of a robot (agent) and its environment. In motion planning, a trajectory is represented in discrete-time with horizon H as a sequence of states

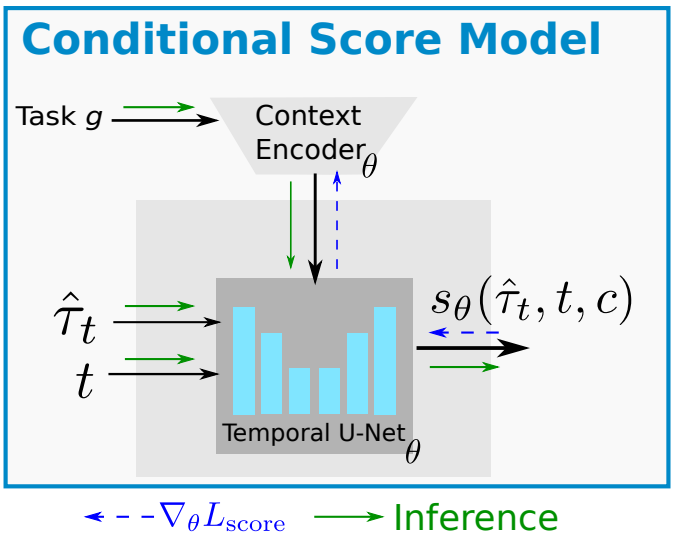


Figure 3.1: Architecture of the basic SBM setup. The task embeddings are used to condition the SBM and are usually the start and goal positions of the trajectory, but can also be e.g. text or images. The score model is implemented as a temporal U-net conditioned on context encodings.

$\boldsymbol{\tau} \triangleq (\boldsymbol{s}_1, \dots, \boldsymbol{s}_H) \in \mathbb{R}^{H \times d}$. It is common to optimize $\boldsymbol{\tau}$ given a *context* $\mathcal{C}$, which can include an occupancy map, obstacle locations, start and final positions, etc. Optimization-based motion planning formulates the problem as trajectory optimization $\boldsymbol{\tau}^* = \arg\min_{\boldsymbol{\tau}} \sum_i c_i(\boldsymbol{\tau}, \mathcal{C})$, where c_i are different costs related to trajectory smoothness, obstacle avoidance or goal reaching [54]. The connection between trajectory optimization and probabilistic inference is well established [1, 53, 27, 35, 60, 59]. Following the notation in [15], the probability of a trajectory (a random variable) factorizes (up to a normalizing constant) as $\tilde{p}(\boldsymbol{\tau}|\mathcal{C}) \propto p(\boldsymbol{\tau}|\mathcal{C})h(\boldsymbol{\tau}|\mathcal{C})$, where $p(\boldsymbol{\tau}|\mathcal{C})$ is a prior distribution (that will be will be learned from data) and $h(\boldsymbol{\tau}|\mathcal{C})$ is a task-specific distribution, e.g. a delta distribution for the starting state of the trajectory. Trajectory optimization computes the maximum-a-posteriori solution $\boldsymbol{\tau}^* = \arg\max_{\boldsymbol{\tau}} \log \tilde{p}(\boldsymbol{\tau}|\mathcal{C})$.

On the other hand, in inference we sample from $\tilde{p}(\boldsymbol{\tau}|\mathcal{C})$, which allows us to get multiple solutions. Langevin dynamics [61] is a common approach to sample from $\tilde{p}$, for which we need $\nabla_{\boldsymbol{\tau}} \log \tilde{p}(\boldsymbol{\tau}|\mathcal{C}) = \nabla_{\boldsymbol{\tau}} \log p(\boldsymbol{\tau}|\mathcal{C}) + \nabla_{\boldsymbol{\tau}} h(\boldsymbol{\tau}|\mathcal{C})$. I will approximate the gradient of the log-prior distribution of trajectories with a *conditioned* SBM $\boldsymbol{s_\theta}(\boldsymbol{\tau}, \boldsymbol{c}) \approx \nabla_{\boldsymbol{\tau}} \log p(\boldsymbol{\tau}|\mathcal{C})$.

3.2 Conditioned Score-Based Generator for Collision-Free Trajectories

Given a dataset of task features $\boldsymbol{g}$ (e.g. initial and final positions) and collision free trajectories $\mathcal{D} = \{(\boldsymbol{\tau}^i, \boldsymbol{g}^i)\}_{i=1}^N$, I model a conditional distribution $p(\boldsymbol{\tau}|\boldsymbol{c})$.

A context embedding $\boldsymbol{c} = \boldsymbol{c_\theta}(\boldsymbol{g})$, is used as a conditioning variable for the SBM. For the experiments I learn an implicit representation of $p(\boldsymbol{\tau}|\boldsymbol{c})$ by modelling a conditioned score function $\boldsymbol{s_\theta}(\boldsymbol{\tau}_t, t, \boldsymbol{c}) \approx \nabla_{\boldsymbol{\tau}_t} \log p(\boldsymbol{\tau}_t, t|\boldsymbol{c})$, implemented as a ***conditional temporal U-Net***. The network architecture uses the Diffuser from [15] as an unconditional model, with the addition of conditioning at the end of each temporal residual block of the U-Net, similar to [46]. Here the conditioning can be done through a cross-attention mechanism. This increased performance a lot dependent on the setting.

3.2.1 Learning Score Model

The Score model can be learned using collision free-trajectories and the corresponding context. For this we have to optimize the score loss

$$\mathcal{L}_{\text{score}}(\boldsymbol{\theta}) = \mathbb{E}_{\boldsymbol{\tau},\boldsymbol{g}\sim\mathcal{D}}\mathbb{E}_t \left[\lambda(t)\mathbb{E}_{\boldsymbol{\tau}_t}\left[\|\boldsymbol{s_\theta}(\boldsymbol{\tau}_t, t, \boldsymbol{c_\theta}(\boldsymbol{g})) - \nabla_{\boldsymbol{\tau}_t} \log p(\boldsymbol{\tau}_t|\boldsymbol{\tau}, t)\|^2\right]\right]. \tag{3.2}$$

Following [51, 46], the context encoder $\boldsymbol{c_\theta}$ and the conditioned SBM $\boldsymbol{s_\theta}$ are jointly trained. As you will see in chapter 6 I did experiment with including environment features to generalise to different environments. Any time you see something like that, the feature encoders have been trained separately, which simplifies training and also just worked better. Also, this solution is modular, allowing the use of other types of pre-trained modules needed for robotics without retraining everything, e.g. one could use dense object features to create movements conditioned on a particular object. The context encoder $\boldsymbol{c_\theta}$ is still trained jointly with the SBM every time.

3.2.2 Trajectory Generation

Once we have the SBM, we now have to use it to sample trajectories with it. Given a new task we compute the context vector $\boldsymbol{c}$. To generate trajectories we solve a reverse-time Stochastic Differential Equation (SDE) as described in [51]. What I use for all experiments

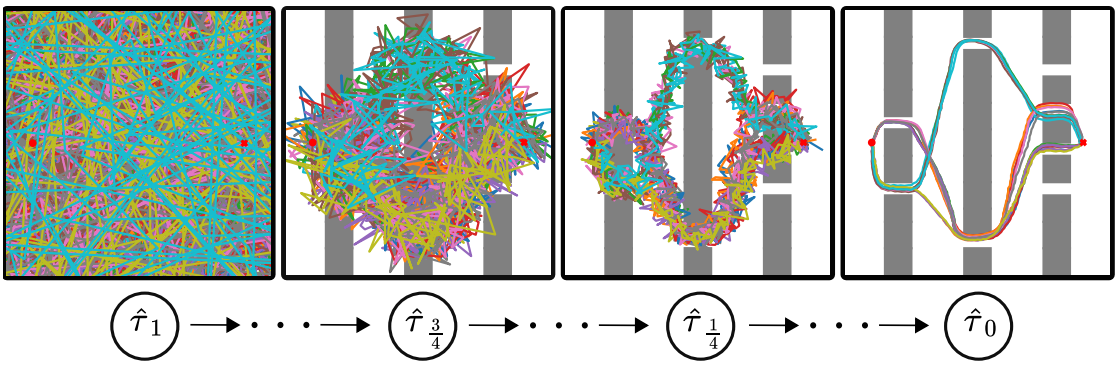


Figure 3.2: A visualisation of how denoising works with an SBM. The left image is random Gaussian noise. The noisy samples are increasingly denoised by many incremental steps until we get to the desired distribution.

however is an Ordinary Differential Equation (ODE) solver that is very similar but omits adding noise. According to Chen et al. [5] since the score is calculated by a Neural Network, this makes this a neural ODE. Using this neural ODE setup the sampling is faster than using something like Langevian dynamics where noise is added at every step. From my experience it also seemed like the ODE produces smoother trajectories than the SDE.

$\boldsymbol{\tau}_{t-\Delta t} = \boldsymbol{\tau}_t + a^{2t}\boldsymbol{s}_{\boldsymbol{\theta}}(\boldsymbol{\tau}_t, t, \boldsymbol{c})\Delta t$

In practice, I sample a noisy trajectory $\boldsymbol{\tau}_1 \sim \mathcal{N}\left(\mathbf{0}, 0.5(a^2-1)\mathbf{I}\right)$ and iteratively run $\boldsymbol{\tau}_{t-\Delta t} = \boldsymbol{\tau}_t a^{2t}\boldsymbol{s}_{\boldsymbol{\theta}}(\boldsymbol{\tau}_t, t, \boldsymbol{c})\Delta t$, where Δt is a time discretization (in our case often $\Delta t = 1/25$) and $a = 2$. As a side-note if we wanted to change this to an SDE formulation, we would have to add the noise factor $a^t\sqrt{\Delta t}\boldsymbol{z}$ with $\boldsymbol{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ as well.

You can see a visualisation of what trajectories at different steps in the denoising process look like in figure 3.2.

There are two engineered improvements I made to the sampling. The first is to not use a linear time scale, but an exponential one. the second is to add a couple deterministic steps at the end with $t = 0$, which really seems to make a huge difference. Both aim to do more iterations where the noise scale is low in order to have very smooth trajectories.

3.2.3 Integrating Environment Contexts

In order for a SBM to generalise to different environments we can add different features contexts as well. We can for example encode environment point clouds using Vector Neuron Networks (VNN) [7] to gain contexts. This is very simple to integrate since it is simply another input into the context encoder. I do not investigate this further in the experiments, but there is a section in chapter 6 where I talk a bit more about my experience with this.

3.3 Integrating Score-Based Models Into GPMP

GPMP and StochGPMP have been used in [54] where something similar to an SBM has been used as a cost factor in addition to other costs like obstacle avoidance. These costs are then used to modify an initial trajectory distribution of GPMP, in a MPPI like manner. Since I also approximate a score function I could have combined an SBM with GPMP in a very similar way as well. I decided to research another direction though. I use an SBM to generate a better initial trajectory distribution that can then be used to initialize the GP distribution of StochGPMP. In practice this is done by sampling K trajectories from an SBM an using those as initial means for StochGPMP.

4 Experiments

For all of the following experiments, I use variations of the architecture shown in Figure 3.1. The only differences may be in the network size or the number of sampling steps, but these are consistent in each experiment.

SBMs are expected to model high-dimensional data very well. For this reason, we model entire trajectories and not just individual steps. Theoretically, this means that the sampling time is constant, regardless of the dimension of the environment.

Environments In this chapter we will use environments with two different numbers of dimensions. The first one is 2D. These environments are very easy to visualize, so we can explain many relevant concepts using these environments. The second is 3D, which is the same environment as 2D but extended in the z direction. Most motion planners take longer in higher dimensions, and we hope that SBM scales well and does not have this drawback.

See the Future Work chapter 6 for some preliminary results for a 7D Franka-Panda simulation.

Data Acquisition The data used in the following experiments comes from one of two sources. Either from RRT* or from GPMP. I will note which source was used in each experiment. There are a number of reasons to prefer one source over the other in a given situation. The main advantage of RRT* in off-line data generation for training an SBM is that although RRT* approaches the optimal solution given enough time, it can easily be stopped early to obtain less optimal but multimodal data. I tried to use RRT connect to generate training data, but it turned out that the quality of the data was not high enough to properly train the score model. That is, the data was not nearly smooth enough.

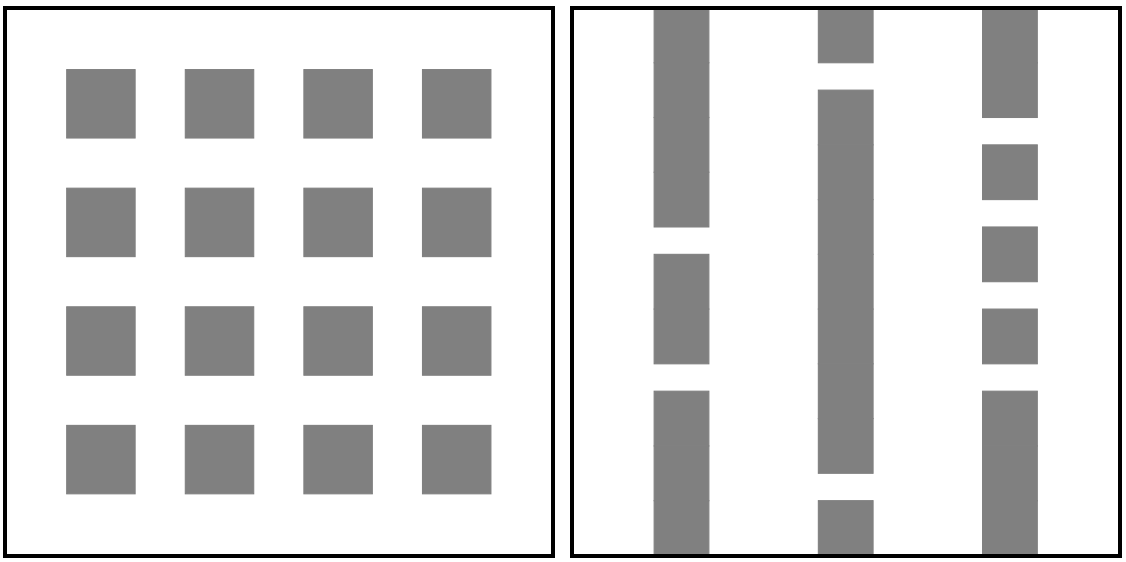
The other main method used is GPMP. This method was necessary because RRT* becomes simply unacceptably slow for 7D data. However, an experiment in 2D illustrates a disadvantage of using GPMP. GPMP tends to collapse to a single mean value. This is not a bad thing in itself, but a major advantage of SBMs is their ability to model multi-modal data, so to show this we would really like multi-modal data to model. As you will see, there are still multiple modes in this data, but not as many as if it had been generated with RRT*.

In summary, RRT* is preferable to GPMP for data generation, but it cannot be applied everywhere.

Data Preprocessing In 2D and 3D, the data are generated already normalized between -1 and 1. In the case of the Panda environment, the trajectories are generated in joint space, within the radius limits of the robot's joints. Again, normalizing the data to -1 and 1 seems to be very helpful for training, so I did that for training as well. Since I wanted to train the SBM with different trajectory lengths, I also apply a B-spline when loading the data. This also increases smoothness and fixes the distance between all points in the trajectory.

Baselines The baselines I look at are RRT connect, RRT* [23] and StochGPMP. All have different strengths, but also some weaknesses. RRT connect is very fast in low dimensional settings, but slows down as the dimension increases. It also produces very choppy trajectories that are often not very close to the optimal solution. RRT* is notoriously slow, but as the number of iterations approaches infinity, RRT* approaches the optimal solution. Both RRT variants are designed so that no generated trajectories collide, and both are difficult to scale due to their sequential nature. StochGPMP is the final basic variant that I will try to outperform and improve upon. It has the advantage of enforcing smoothness and simultaneously computing multiple solutions in parallel on the GPU. It does not have access to the gradient of the environment and therefore must explore the space randomly. This can work quite well if the environment is simple enough, but as we will see later, it becomes really unreliable and slow in more difficult environments.

Standalone SBMs Since I want to study SBMs in different environments, you will always find statistics on the performance of a stand-alone SBM for each of the experiments. I also explore how to incorporate the gradient of the SDF into the sampling process, as described in Chapter 3. As you will see, the SBM is able to enforce smoothness quite well, which may be due to the splines used in the data preprocessing.



(a) 2D grid environment. The SBM for this environment has been trained on GPMP generated data

(b) 2D narrow passage environment. The SBM for this environment has been trained on RRT* generated data.

Figure 4.1: Two 2D motion-planning environments. The grey areas are obstacles to be avoided.

StochGPMP Priors There are two priors that I consider and compare in the experiments. Both have the goal of improving the trajectories in some way while reducing the sampling time through better initialization. The first idea is to use RRT connect to generate an initial collision-free trajectory, which is then smoothed by StochGPMP. The second idea, of course, is to use an SBM as a prior. However, the motivation is different. I expect an SBM to already generate a smooth trajectory since it is trained on smooth data, but it does not check for collisions at all. So the task of StochGPMP is essentially to take a good trajectory and slightly perturb it so that it is also collision free.

Metrics There are five metrics that I analyze for each of the methods. The first is the **sampling time in seconds**. Since some of these methods can run well in parallel and others like RRT cannot, the times describe the time it takes to obtain a sample or batch of samples depending on the method. This may seem like an unfair advantage for RRT, but this is a limitation of the hardware, and there could easily be a system running 100 RRT

algorithms simultaneously, which would have the same time result. StochGPMP is a bit of a special case here. It uses 100 particles at once, resulting in 100 samples. This algorithm doesn't make much sense when running with just a single particle, but StochGPMP slows down when more particles are involved. The time of SBM is completely independent of the number of samples, since the inference is performed in parallel on the GPU.

The second and third metrics used are **percentage of collision-free samples** and **collision intensity**, which is the average percentage of steps in collision. The former tells how many out of 100 trajectories are expected to be collision-free, while the latter tells how severe the collisions are on average. If the collision intensity is very low, many of the collided trajectories can be saved by changing a few points.

The fourth metric is the distance along the trajectory. The shorter the better in general, but if there are many collisions, this metric loses its value. We can easily imagine a degenerate case where the start and finish are connected by a straight line. This would give the smallest possible distance, but the solution would be essentially useless if there are obstacles in the way.

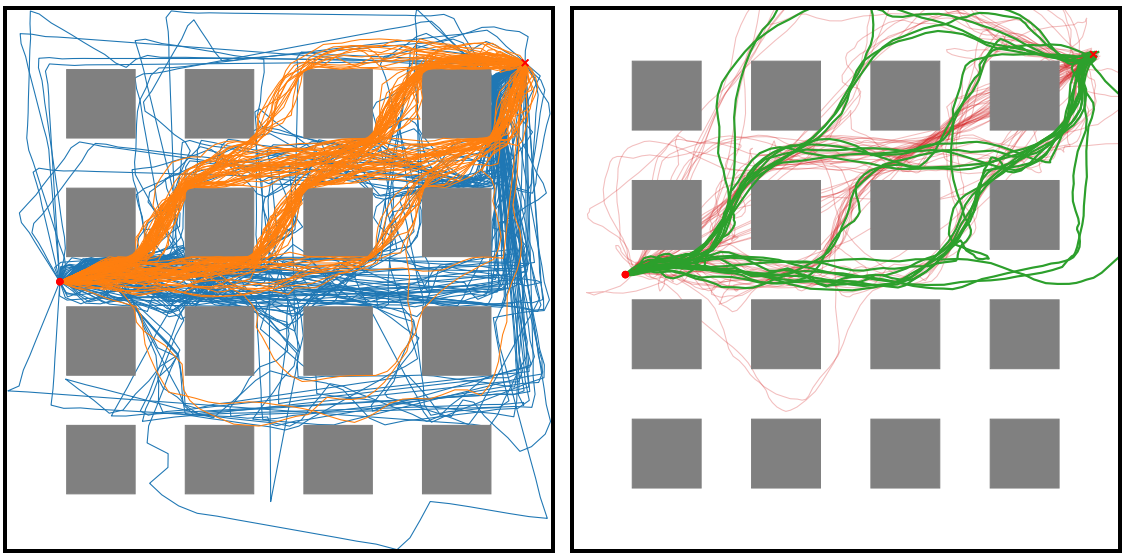
The last metric is the **inverse cosine similarity**. It is calculated pairwise between consecutive points. This metric tells us if there are many drastic changes in the direction of the velocity. In many of the following tables, you will see a red entry in this column for the RRT prior StochGPMP case. The reason this value is red is because it is misleading and does not measure what we want. When we use an RRT trajectory as a prior, we need to interpolate it to increase or decrease the number of points describing the entire trajectory. This is simply a requirement of StochGPMP that all trajectories have the same length H.

Since RRT generates connect long straight lines, the pairwise cosine similarity metric for most points in the interpolated trajectory will be 0.

4.1 2D Grid Environment

This environment, as shown in Figure 4.1a, is quite simple in terms of planning difficulty. There are many straight and wide corridors, which is very beneficial for any RRT algorithm since it is very likely to find a point that can be connected to the tree.

Both the RRT baselines and the StochGPMP baseline are shown in figures 4.2a and 4.2b, respectively. In terms of quality, this is an environment where the baselines can really shine.



(a) **RRT connect** (blue) and **RRT*** (orange). Notice how RRT* finds way more optimal and smoother paths that RRT connect.

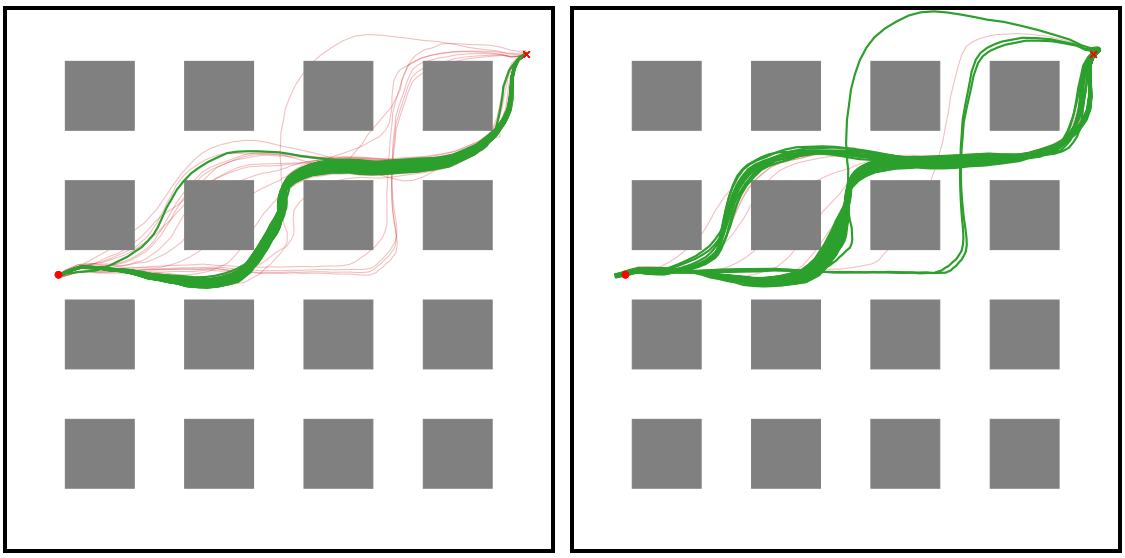
(b) **StochGPMP** run without a specific prior with (red) and without collision (green).

Figure 4.2: Baselines in 2D grid environment. Both sampling based baselines in 4.2a do not generate samples in collision. The optimization based baseline in 4.2b can create trajectories in collision.

For better or worse, **RRT connect** shows the ability to find many different paths. In this particular example, many samples take a very direct path that leads all the way to the right and then all the way up to the destination. This behavior is to be expected from RRT connect. Another behavior we expect from RRT connect is that it simply finds any connection between two points. This leads, as can be clearly seen in Figure 4.2a, to many sharp turns, often followed by very long straight lines. For a robot agent, a gradual change in velocity would be much better.

RRT*, like RRT connect, is capable of finding multiple paths to the goal. However, the modes found are arguably more useful for planning, since all modes seem to be reasonable paths to the destination. This behavior is also to be expected as RRT* converges to the optimal solution, meaning that given enough time, all modes will converge to a single optimal trajectory. In any case, these trajectories are much more practical for use with a robot. The trajectories exhibit far less extreme velocity changes.

StochGPMP also copes quite well with this environment. This baseline, unlike the two RRT baselines, can generate trajectories that are in collision. As can be seen in the Figure 4.2b, many of the trajectories are collision-free, but many are also in collision. The modes discovered by StochGPMP look similar to those found by RRT*. Since StochGPMP enforces smoothness, the trajectories are also fairly smooth. There are some outliers, but most have no jumps in velocity.



(a) **SBM** sampled trajectories with (red) and without collision (green).

(b) **SBM+SDF** sampled trajectories with (red) and without collision (green).

Figure 4.3: Here you see the effects of including the SDF into the sampling process of an SBM. Since SBMs do not enforce reduction of collisions at at, simply adding that as a separate gradient drastically decreases collision.

In Figure 4.3a you can see trajectories sampled with a standalone **SBM**. One thing that is really obvious is that the trajectories appear to have few modes. This is most likely due to the source of the training data in this environment. The training data here is from a GPMP. GPMPs tend to collapse to a few modes, so an SBM trained on GPMP data will learn the presented distribution of trajectories. As we will see in the next section, for data with more modes, the SBM is able to produce multi-model data just as well.

Since the training data were splines, the trajectories are very smooth and have virtually no jumps in velocity. However, there are many samples that collide, since there is nothing to force obstacle avoidance.

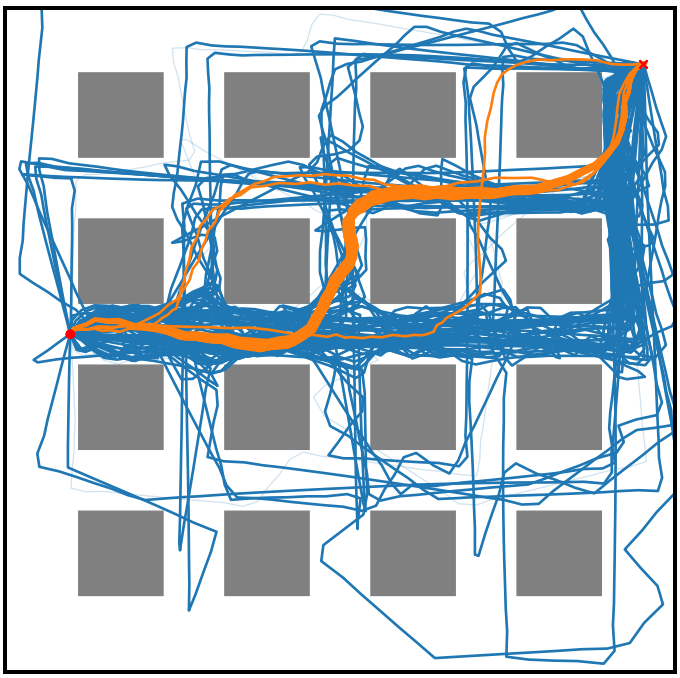


Figure 4.4: **StochGPMP** with two different priors. **SBM** prior in (orange) and **RRT connect** prior (blue). RRT connect already produces collisoin free trajectories, so there are few collision signals for StochGPMP to work with. With the SBM prior we gain the advantages of the smooth prior and the collision checking of StochGPMP

One way to combat these collisions is to include other gradients in the sampling process, as described in 3.1. The effects of applying the SDF gradient during sampling can be seen in Figure 4.3b. Far fewer samples collide, while the smoothness looks very similar.

Of course, this only works if we have access to the gradient of the SDF, which is not always the case. The last point to explore is the use of SBM samples as priors for StochGPMP.

In Figure 4.4 you can see the comparison of the execution of StochGPMP with two different priors. RRT connect and the SBM prior. The reason why RRT connect might conceptually be an interesting alternative prior is that it is fast and produces collision-free trajectories. However, just by visual inspection we can see that this is not true. Since RRT connect does not generate any collision-ridden trajectories, there is simply not enough cost signals for StochGPMP to improve upon. However, the trajectories sampled with the SBM prior show

very good results. In this example, there are no colliding trajectories, but this method can still produce colliding trajectories. The trajectories are also very smooth.

4.1.1 Statistical Data

In this section I will continuously refer to table 4.1 in which you can find the results for all the metrics in this environment.

Sampling Time

In this environment, the RRT Connect baseline is basically unbeatable. In reality, of course, it's a little over 0.00, but not by much. The other two baselines are quite slow in comparison. Both take several seconds to complete. However, for reasons explained in the next sections, these two are the main competitors for the SBM. The sampling time of the SBM and the SBM combined with the SDF gradient is basically identical. The only difference is due to variations in hardware performance. Other than that, both are the next fastest options in this comparison after RRT connect. This is followed by both versions of StochGPMP using a prior. The sampling time for the StochGPMP portion is the same for both the SBM and RRT priors, but since RRT is faster than SBM, so is StochGPMP when RRT is used as a prior. Both prior versions are significantly faster than the standalone StochGPMP, as they both require much fewer iterations to produce good samples.

Collisions

Both RRT baselines achieve perfect results in this regard, since they cannot create colliding trajectories. StochGPMP is a far more interesting baseline to compare to. Since it is an optimization-based motion planner, it is expected that many of the trajectories will collide in non-trivial environments. In this environment, StochGPMP performs quite well in terms of the number of collision-free trajectories. However, there is a large variance here, as the difficulty of the contexts tested varies widely. However, a look at the collision intensity shows that larger parts of the trajectories are in collision for StochGPMP than for all other methods.

Using SBM alone results in trajectories that are collision free more often. But when they collide, the collision intensity is a lot lower than StochGPMP. This can be drastically improved by adding the gradient of the SDF during sampling or some StochGPMP steps after

	Sampling Time (s)	Collision Free (%) ↑	Collision Intensity (%)	Distance	Cosine Sim
RRT connect	**0.00** ±0.00	**100.00** ±0.00	**0.00** ±0.00	1.71 ±0.60	0.76 ±0.42
RRT*	11.34 ±0.58	**100.00** ±0.00	**0.00** ±0.00	**1.48** ±0.18	**0.06** ±0.02
StochGPMP	7.10 ±0.10	72.46 ±22.6	1.78 ±1.57	1.56 ±0.37	0.09 ±0.03
SBM	**0.28** ±0.02	79.97 ±27.97	0.81 ±1.71	**1.47** ±0.40	0.07 ±0.04
SBM+SDF	**0.26** ±0.00	**94.5** ±14.79	0.22 ±0.76	1.50 ±0.43	0.07 ±0.04
SBM Prior + StochGPMP	1.00 ±0.23	93.43 ±17.07	0.23 ±0.75	**1.47** ±0.40	0.07 ±0.05
RRT Prior + StochGPMP	0.72 ±0.02	93.99 ±6.55	**0.11** ±0.12	1.70 ±0.60	0.03 ±0.01

Table 4.1: Different metrics and methods in the 2D grid environment. All derived from 100 different contexts (start and goal positions) with 100 sampled trajectories each. This environment has lots of straight lines, which is quite advantageous for all variants of RRT.

sampling. Both lead to similarly good results with about 94% collision-free trajectories. The collision intensity is also very low for both. The RRT prior we tested achieves similarly good results in these metrics, but is worse than the standalone prior in both cases.

Distance

Distance is a metric with no big surprises. The best in the baselines is RRT*, followed by StochGPMP. RRT connect generates the longest paths on average of all methods tested.

SBM produces very similar results to RRT* in this metric on average, but the standard deviation is higher, meaning that some trajectories are quite a bit longer. Adding the gradient of the SDF during sampling increases the average distance a bit, which can be explained by the gradients pushing the trajectory further away from the obstacles.

Running StochGPMP steps with the SBM prior gives the same length as sampling the SBM alone. This means that using the SBM prior reduces the distance for StochGPMP in this environment.

The RRT prior is basically the same as using RRT connect, which is not very good.

Smoothness

The cosine similarity I use here is 0 when there is no change in direction between two successive accelerations. This means that lower values are desirable because the signals to a robot controller would be more stable.

RRT* is the best baseline for this metric, followed by StochGPMP. RRT connect is by far the worst, so it is not suitable for real robot systems.

As mentioned earlier, this metric is unreliable for the RRT priority case. All SBM methods basically achieve exactly the same results and are only slightly behind RRT*.

Conclusion

All baselines have their weaknesses and advantages. RRT connect is fast, but very choppy and produces long trajectories. RRT* fixes both flaws of RRT connect, but takes a lot of time, especially in higher dimensions. StochGPMP is slightly faster than RRT*, but it is not guaranteed to generate trajectories without collisions. Using RRT connect as a prior for StochGPMP seems rather pointless in this environment, since StochGPMP only degrades the prior trajectories by perturbing them in positions that are now in collision.

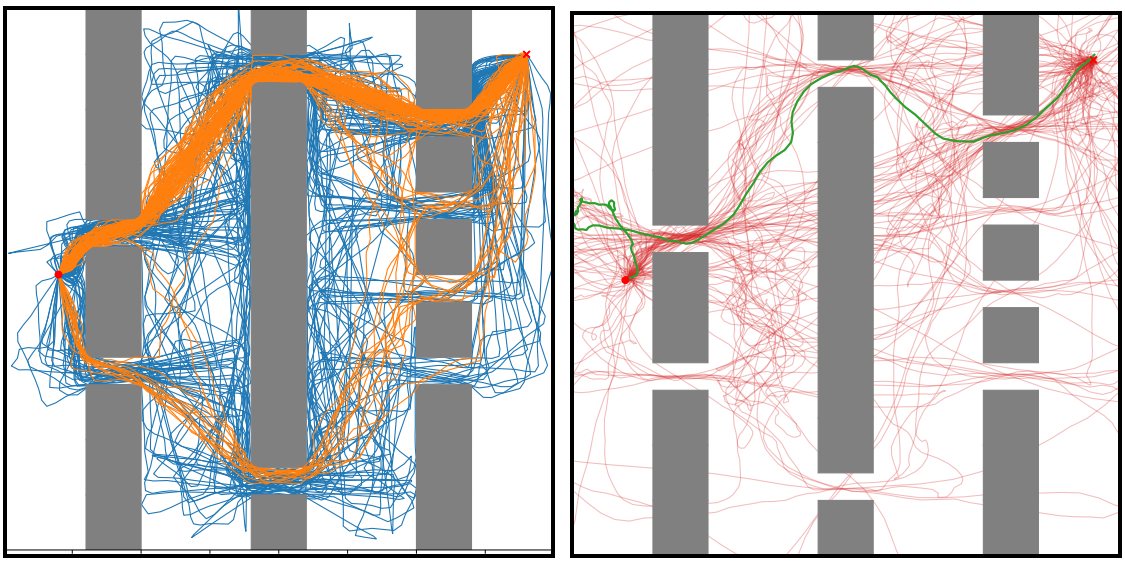
The standalone SBM already outperforms StochGPMP in every single metric. Some are only minor improvements, but a really big jump occurs in sampling time. The real value of the SBM comes when it is used either in conjunction with additional gradients or as a prior to StochGPMP. Both perform better than all baselines when all metrics are considered. The generated trajectories are smooth, short, and have few and short collisions, while the sampling times are low. These results suggest that it is best to use the gradient of the SDF when it is available. When it is not, using the SBM as a prior for StochGPMP produces similar results with only slightly higher computation time.

4.2 2D Narrow Passages Environment

In the previous section we looked at an example where planners like RRT and StochGPMP perform really well. Although using an SBM in this environment is already very useful, there are also environments where traditional planners struggle, while an SBM is able to generate trajectories in these environments just as well.

An example where this is the case can be seen in Figure 4.1b. Instead of a grid, this environment has very narrow passages that are not horizontally aligned. In order for RRT to find a path from left to right, very specific points must be randomly sampled, while StochGPMP basically fails because the collision cost is not very helpful in this environment.

As in the previous section, you can see some baseline trajectories in Figure 4.5. **Both RRT variants** work exactly as expected. RRT* generates multi-modal trajectories, but many more samples are generated in the shortest possible path region. Long sections of the trajectories generated by RRT connect are very close to the wall. This is expected behavior, but not good when generating a path for a robot to follow, as we prefer to keep distance from obstacles for safety reasons.



(a) **RRT connect** (blue) and **RRT*** (orange). Notice how RRT* finds way more optimal and smooother paths that RRT connect.

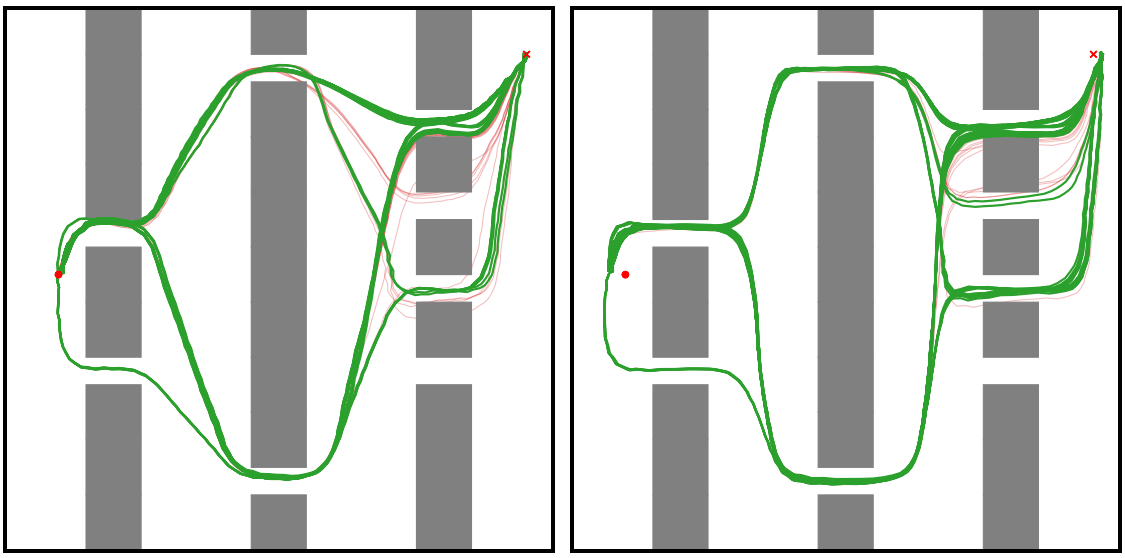
(b) **StochGPMP** run without a specific prior with (red) and without collision (green).

Figure 4.5: Baselines in the 2D narrow passages environment. Both sampling based baselines in 4.5a do not generate samples in collision. StochGPMP in 4.5b finds barely any solutions.

As can be seen in Figure 4.5b, StochGPMP is barely able to generate collision-free trajectories in this environment. With enough time and care in fine-tuning the parameters, **StochGPMP** should perform better, but using the same setup as before with a more difficult environment leads to very undesirable behavior.

In Figure 4.6a you can see how the trajectories sampled by a **SBM** look like in this environment. This time, the SBM was trained with data generated by RRT*, which was terminated a little earlier to generate multi-modal data. This enables the SBM to find many solutions with different modes.

By adding the gradient of the SDF during sampling, the number of collision-free trajectories can be increased again, as can be seen in Figure 4.6b. It should be noted that the indiscriminate addition of the gradient of the SDF results in the start and goal points not being reached. However, this is merely a matter of removing or fine-tuning the influence of the gradient at these points.



(a) **SBM** sampled trajectories with (red) and without collision (green).

(b) **SBM+SDF** sampled trajectories with (red) and without collision (green).

Figure 4.6: Just as in the previous environment, applying the gradient of the SDF decreases the number of trajectories in collision.

In Figure 4.7 you can see a comparison between the use of an RRT connect and an SBM prior for StochGPMP. Again, the RRT prior results in very choppy trajectories. However, it restores more modes. The SBM prior leads to very smooth trajectories. There are very few collisions in either set of trajectories.

4.2.1 Statistical Data

The data for this subsection can be seen in Table 4.2. We consider the same metrics as before, but keep in mind that the models and parameters used here may vary between environments.

Sampling Time

RRT connect still takes hardly any time. RRT* still takes about the same amount of time, since it terminates after a fixed number of nodes have been added to its tree. But

	Sampling Time (s)	Collision Free (%) ↑	Collision Intensity (%)	Distance	Cosine Sim
RRT connect	**0.02** ±0.03	**100.00** ±0.00	**0.00** ±0.00	2.01 ±0.97	0.88 ±0.48
RRT*	11.8 ±1.20	**100.00** ±0.00	**0.00** ±0.00	**1.70** ±0.32	**0.19** ±0.21
StochGPMP	23.89 ±0.06	22.81 ±17.26	3.49 ±0.75	4.02 ±0.25	0.25 ±0.32
SBM	**0.37** ±0.03	73.86 ±32.69	1.37 ±2.97	**1.66** ±0.89	0.17 ±0.33
SBM+SDF	**0.35** ±0.01	93.80 ±16.05	0.27 ±0.93	1.79 ±0.96	0.22 ±0.43
SBM Prior + StochGPMP	2.79 ±0.23	**97.36** ±8.89	0.15 ±0.69	1.71 ±0.89	**0.14** ±0.24
RRT Prior + StochGPMP	2.44 ±0.04	95.15 ±9.30	**0.07** ±0.15	2.00 ±0.98	0.02 ±0.01

Table 4.2: Statistics for the second 2D environment. Since the passages are so narrow, times for all methods increase. Most notable for StochGPMP, which is where we really see the speed advantage of using a better prior.

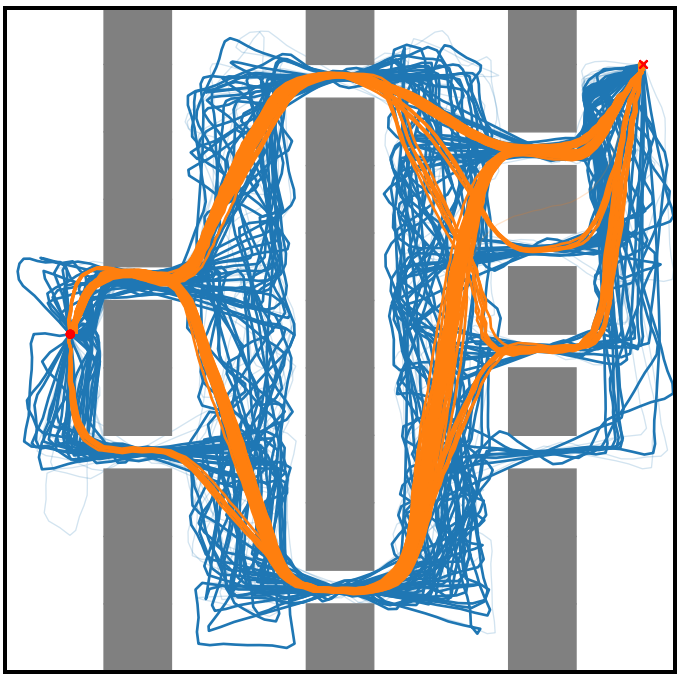


Figure 4.7: **StochGPMP** with two different priors. **SBM** prior in (orange) and **RRT connect** prior (blue). Like in the previous environment, using RRT connect as a prior results in jittery trajectories, while using SBM as a prior results in very smooth trajectories with few collisions.

StochGPMP takes much longer to generate trajectories than in the previous experiment. This shows that StochGPMP must be used judiciously. The parameters and the number of iterations have to be evaluated manually and changed according to the circumstances.

Apart from RRT connect, by far the fastest methods are pure SBM and SBM combined with the SDF gradient. They require essentially the same time because they are sampled with the same number of diffusion steps. The small differences in time must be due to computational variance.

Both StochGPMP versions with priors take slightly more time than in the previous environment, but they are much faster than running StochGPMP alone.

Collisions

Again, RRT does not produce collision-free trajectories, but StochGPMP does, and quite a lot at that. In this environment, StochGPMP produces less than 23% collision-free trajectories on average. The collision intensity is also very high, meaning that the collisions are probably not just due to too-tight turns, but really bad trajectories are being generated.

Looking at the SBM models, we see that even the stand-alone SBM significantly outperforms StochGPMP in terms of collision-free trajectories and collision intensity. However, when combined with the gradient of the SDF, the performance increases dramatically again. Over 93% of the trajectories are collision free and the collision intensity is tiny. Even better results are obtained with the StochGPMPs with priors. The StochGPMP using the SBM prior achieves over 97% collision-free trajectories with an even lower collision intensity. The StochGPMP using the RRT prior also performs very well in terms of collision-free trajectories, but worse than pure RRT connect, so there seems to be no reason to use RRT as a prior here.

Distance

The distance of RRT* is the shortest among the baselines, followed by RRT connect. It is possible that RRT* would have needed a bit more time to achieve even shorter distances. By far the worst with a distance twice as long as RRT connect is StochGPMP. The reason for this is the uninformed random nature of the exploration in StochGPMP, which results in long paths that are in collision most of the time.

The method that produces the shortest distances on average is standalone SBM, which outperforms even RRT*. This is closely followed by the SDF gradient version and the StochGPMP with the SBM prior. It is to be expected that the SDF version results in trajectories that are slightly longer than pure SBM trajectories, since they are pushed away from obstacles. Nevertheless, all SBM versions are at least very close to or exceed RRT*.

Of course, the StochGPMP with RRT prior offers only a slight improvement over the standalone RRT connect again.

Smoothness

So, there is much more variance in the cosine similarity than in the other environment. The reason for this is quite simple. When the start and finish positions are in the same section between the columns, the line is fairly straight and the cosine similarity is low, while it is higher when the planner has to navigate through one or more narrow passages. The cosine similarity is also generally higher, which is to be expected since more directional changes are required to navigate in such a difficult environment.

Apart from the fact that RRT connect performs poorly and that the RRT prior StochGPMP value cannot be trusted, there is not much more to say on this topic. All results are similarly good.

Conclusion

In a tricky environment like this one the value of reusing previously generated trajectories becomes apparent. If it is really easy to plan a path then why even bother with SBMs? Well here it takes considerable effort and time to plan a path between two positions. The statistics show were created by randomly sampling start and goal positions randomly and they capture the average between those. If we consider the visual results in figures 4.5b, 4.6a, and 4.7 it is easy to see that especially for long tricky paths, a good initialisation can have huge impact on the quality of the samples.

Besides sample quality the time it takes to sample in the environment is probably the biggest motivation to use SBMs as a prior or in conjunction with other gradients.

In a tricky environment like this one, the value of reusing previously generated trajectories becomes apparent. If it's really easy to plan a path, why bother with SBMs at all? Well, in this environment, planning a path between two positions is very tedious and time consuming. The statistics shown were created by randomly selecting start and finish positions and show the average between those positions. Looking at the visual results in Figures 4.5b, 4.6a, and 4.7, it is easy to see that especially for long, tricky paths, good initialization can have a huge impact on sample quality.

In addition to sample quality, the time required to sample the environment is probably the biggest motivation for using SBMs as a prior or in conjunction with other gradients.

4.3 3D Narrow Passages Environment

In this section we will look at the ability of SBMs to scale to higher dimensions. The environment we are considering is a 3D environment that mimics the 2D environment in Figure 4.1b. The difference, of course, is that there is a third dimension. This is visualized in Figure 4.8.

Many motion planners have difficulty with higher dimensions, so it is worth investigating how SBMs respond to higher degrees of freedom. We consider a drone that can fly freely in a 3D space.

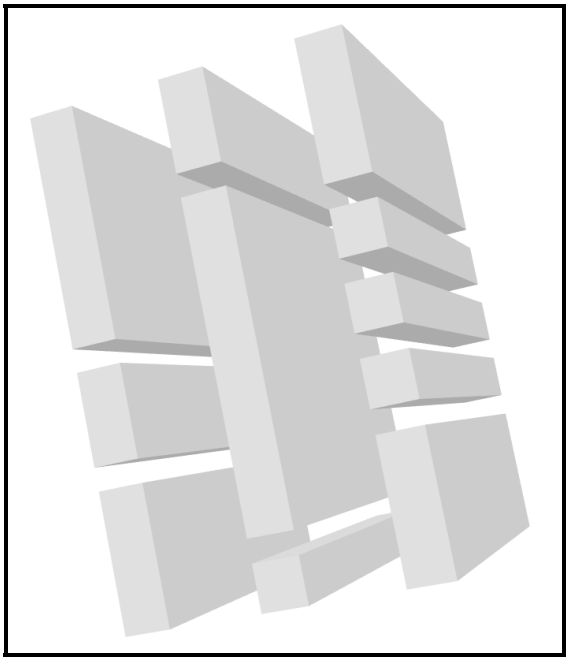
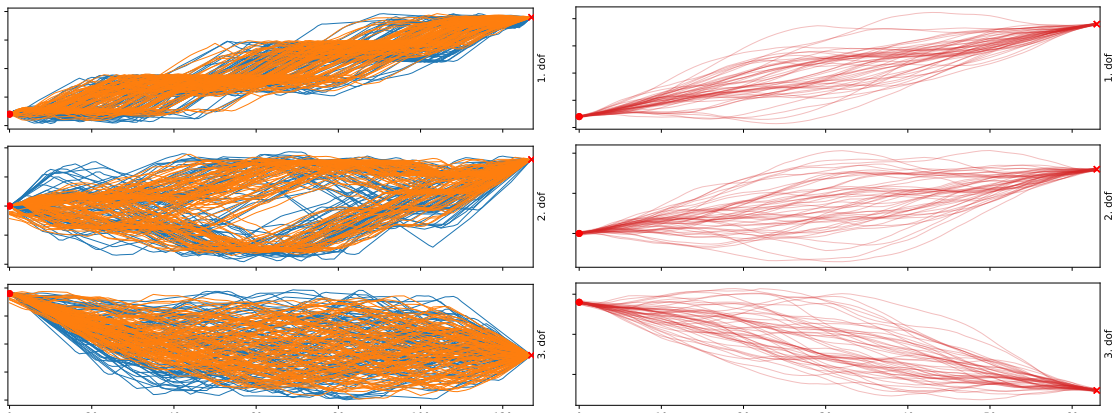


Figure 4.8: 3D version of the 2D environment in Figure 4.1b. The walls stretch from side to side, so to go from left to right, a trajectory must be planned to go through the narrow passages again.

Looking at the baselines in Figure 4.9, we can see that **RRT baselines** perform just as well as before in 2D. **StochGPMP** struggles even more and cannot find a single collision-free trajectory for the given start and target positions. This context is among the most



(a) **RRT connect** (blue) and **RRT*** (orange).

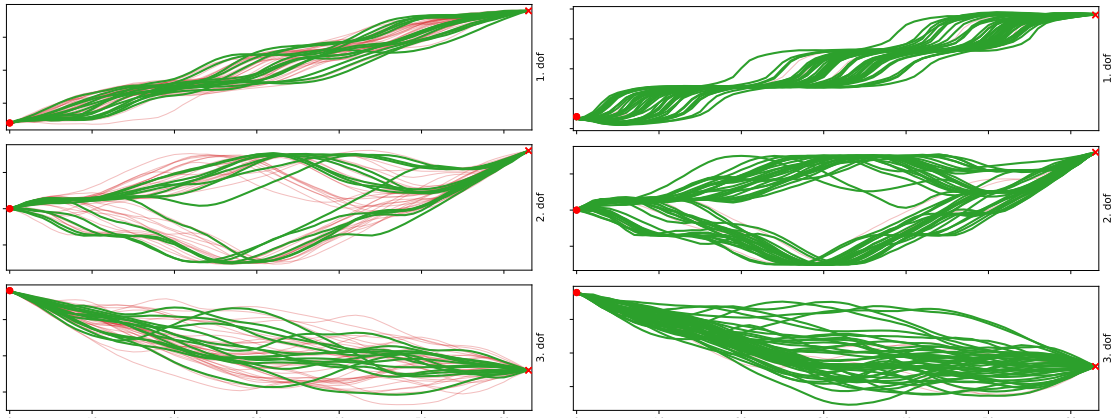
(b) **StochGPMP** run without a specific prior. All trajectories are in collision in this context.

Figure 4.9: Baselines in the 3D narrow passages environment. The x axis describes the horizon H. The x and y positions are the same as in the second 2D environment experiment.

difficult possible, and with enough steps and more fine-tuning, it would eventually find a collision-free path.

In Figure 4.9a, we can see that there is some multi-modality in the second dof. This is what we expect because the context is the same as in the 2D case, with the addition of a third dimension as the start and finish. Again, RRT* produces rather smooth trajectories compared to RRT connect.

In Figure 4.10a you can see some samples generated with a **SBM**. Just like in the previous experiment, the trajectories look smooth and show that the model is able to learn multi-modal trajectory distributions. There are also quite a few trajectories that are in collision. A look at Figure 4.10b shows once again that the inclusion of the **gradient of the SDF** during sampling reduces the number of collisions while maintaining smoothness. The **StochGPMP versions with a prior**, shown in Figure 4.11, show a similar picture to the 2D experiments. The RRT connect prior appears to be rather useless, while the SBM prior reduces the number of collisions just like the inclusion of the SDF gradient does.



(a) **SBM** sampled trajectories with (red) and without collision (green).

(b) **SBM+SDF** sampled trajectories with (red) and without collision (green).

Figure 4.10: As expected, including the gradient of the SDF in sampling decreases the number of trajectories in collision.

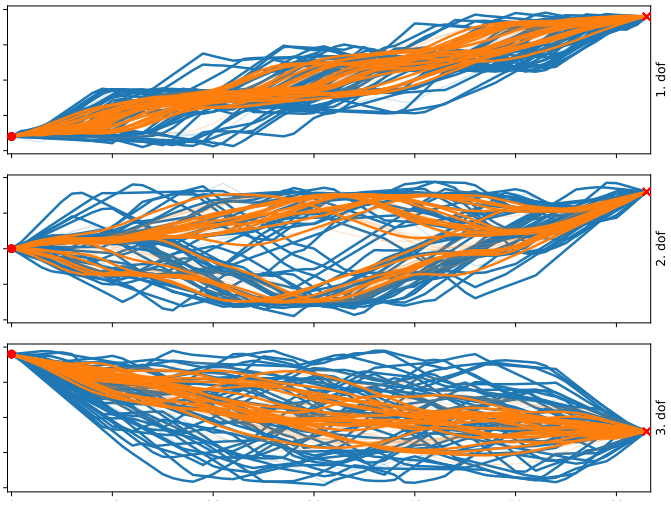


Figure 4.11: **StochGPMP** with two different priors. **SBM** prior in (orange) and **RRT connect** prior (blue). The SBM prior improves the performance of StochGPMP a lot, especially compared to using RRT connect as a prior.

4.3.1 Statistical Data

Increasing the number of dimensions significantly increases the search space for RRT motion planners as well as for StochGPMP. This is also the case for most other planners. This is reflected in the data. RRT connect and RRT* become very slow, especially RRT* though. RRT connect still takes the least time, but much longer than in 2D. StochGPMP is executed with a fixed number of steps. In this environment it might have made more sense to use more steps, but then of course the time increases.

Sampling Time

Using the SBM now takes only twice as long as RRT connect. And that is crucial. The time required to sample an SBM scales very slowly with the dof, and only when the number of sampling steps or the network size changes. In this case, since we were able to use the same network and the same number of sampling steps, the time remains constant. Adding the gradient of the SDF takes much longer, but this should be neglected as it is due to a weak implementation of the SDF calculation. The time difference between using RRT as a prior and using the SBM as a prior for StochGPMP is now rather negligible, and both massively reduce the sampling time for StochGPMP.

Collisions

Collision free trajectories are of course always generated with RRT. StochGPMP has a really hard time and only about 30% of the trajectories of the sample are collision free. However, there is a lot of variance here. Simple contexts are handled with ease and difficult contexts like in Figure 4.9b are almost impossible for StochGPMP to solve. The number of iterations is set quite low, so there is plenty of room for improvement through fine-tuning, but the really difficult contexts remain really difficult.

The standalone SBM produces on average twice as many collision-free trajectories as StochGPMP. However, when you add the gradient of the SDF, this value is very close to 100% with an insanely low collision intensity. So if you have access to the SDF, this is certainly the way to go in this environment. Using SBM as a prior for StochGPMP increases the number of collision-free trajectories by over 12% over just SBM, but is not nearly as good as the SDF. This can be explained by the fact that StochGPMP is not perfectly fine-runed. However, when comparing StochGPMP to the SBM prior StochGPMP, the

	Sampling Time (s)	Collision Free (%) ↑	Collision Intensity (%)	Distance	Cosine Sim
RRT connect	**0.14** ±0.06	**100.0** ±0.00	**0.00** ±0.00	3.21 ±1.48	0.82 ±0.33
RRT*	38.67 ±0.36	**100.0** ±0.00	**0.00** ±0.00	**2.10** ±0.41	0.47 ±0.06
StochGPMP	7.83 ±0.12	33.76 ±30.37	6.39 ±4.24	2.14 ±0.28	**0.08** ±0.02
SBM	**0.27** ±0.14	67.88 ±26.43	3.28 ±5.38	**2.18** ±0.88	0.03 ±0.03
SBM+SDF	1.06 ±0.02	**98.76** ±3.47	**0.03** ±0.10	2.37 ±0.98	0.03 ±0.03
SBM Prior + StochGPMP	1.38 ±0.16	79.89 ±22.6	1.67 ±3.42	**2.18** ±0.89	0.03 ±0.03
RRT Prior + StochGPMP	1.25 ±0.08	98.74 ±1.96	**0.02** ±0.03	3.18 ±1.45	0.07 ±0.04

Table 4.3: Statistics for the 3D environment. Since there is a third dimension, this increases the number of trivial trajectories for contexts that can be connected through a straight line, while also increasing the search space a lot for RRT.

difference in performance is still significant. Using the prior more than doubles the number of collision-free trajectories.

Finally, the RRT prior StochGPMP achieves good performance in these metrics, but is again worse than RRT connect.

Distance

As in all previous experiments, RRT* achieves the best average distance. It is closely followed by StochGPMP, but since so many of the StochGPMP collide, the distance of the non-colliding samples would be much higher. RRT connect produces the longest trajectories on average.

The SBM and the StochGPMP with SBM prior samples achieve the lowest distance after RRT*. The SDF gradient version performs somewhat worse. This is again because the SDF gradient pushes the trajectories away from the obstacles, making the path less direct.

Finally, the RRT prior StochGPMP is a tiny bit better than standalone RRT connect, though not significantly so.

Smoothness

StochGPMP generates the smoothest trajectories of all baselines. RRT* and RRT connect both do not generate very smooth trajectories. RRT* could produce smoother trajectories if more time were available, but it already takes so much more time than all the other methods that this does not seem reasonable to me.

All of the SBM methods outperform the baselines by quite a bit and produce similarly good results. Again, this is likely due to splining of the training data, but it is still good to see that SBMs can produce smooth samples even in higher dimensions.

The metric for StochGPMP with the RRT prior of course cannot be trusted as mentioned previously.

Conclusion

There are a couple of interesting takeaways from this experiment. The most crucial one is that the time it takes to sample trajectories with an SBM is independent of the state space. There can be some fluctuation in sampling time if the model size increases or the sampling steps are changes, but for the experiments I ran this was not necessary. This is a real advantage of SBMs especially when moving to systems that have even larger dofs. A Franka Panda robot usually as 7 dof and as we will briefly see later on the time does not increase even for that.

Another insight is that especially for hard environment the gains of using a SBM as a prior for StochGPMP are immense when compared to not using a prior. This enables us to even use a very badly tuned StochGPMP to get decent results. Even better than using an SBM as a prior is of course using the gradient of the SDF but since we do not always have access to that its not always an option.

Several interesting findings can be drawn from this experiment. The most important is that the time required to sample trajectories with an SBM is independent of the state space. There may be some fluctuation in the sampling time as the model size increases or the sampling steps are changed, but for the experiments I performed, this was not necessary. This is a real advantage of SBMs, especially when moving to systems with even larger dofs. A Franka Panda robot typically has a dof of 7, and as we will see later, the time itself does not increase for sampling with an SBM.

Another finding is that, especially in difficult environments, the benefits of using an SBM as a prior for StochGPMP are immense compared to not using a prior. This allows us to achieve acceptable results even with a very poorly tuned StochGPMP. Even better than using an SBM as a prior is of course using the gradient of the SDF, but since we do not always have access to it, this is not always an option.

5 Discussion

In this chapter, I discuss the results of the experiments and compare a bit how the different methods perform in different environments.

First, the two 2D environments. Although both are 2D, they serve a very different function in demonstrating the capabilities of SBMs in trajectory generation. The first grid environment is very easy for most motion planners to solve, and the SBM is trained with GPMP-generated trajectories. Even in this very favorable environment, the SBM methods perform better than the baselines when all metrics are considered. One could argue that RRT* is still better, but it takes so much longer than even the slowest SBM method that it is probably worth considering the trade-off between the metrics at this point. Even if an SBM method generates more trajectories in collision, this is only the case if you look at the absolute numbers. Looking at collision-free trajectories generated per second, standalone SBM is indeed the best option by quite a lot. Very roughly rounded, RRT* generates only 100 collision-free trajectories every 10 seconds, while the SBM can generate about 3000 trajectories in the same time. Assuming that 80% are collision-free, that's 2400 collision-free trajectories in the same time that RRT* can generate 100.

The second environment with the narrow passages, on the other hand, is very difficult to solve using conventional means. You will notice that all the sampling times of the methods running on the GPU are slower than the previous environment, which is likely due to the fact that they were running on a slower cluster node, but for this experiment they were all running on the same node. The data for the SBM was generated using RRT*, which made the distribution of the training data multi-modal. Here, of course, the effects observed in the first environment are amplified, as some of the baselines really struggle in this difficult environment. StochGPMP becomes very slow and generates less than 25% collision-free trajectories. RRT* is basically the same as in the first environment. The standalone SBM also produces very similar results to the first environment with the added benefit of producing multi-modal trajectories due to the RRT* training data.

The last environment is the 3D environment with the narrow passages. If we consider what affects the sampling times of all methods, we can make some predictions about what we would expect to see. RRT* and StochGPMP should become very slow and/or do not find many collision-free solutions. Since the SBM sampling time is independent of the search space and depends only on the number of sampling steps in the ODE solver, we would expect the sampling times to be fairly constant if the network size does not change. And this is exactly what we see in the statistical results. RRT* becomes terribly slow, StochGPMP becomes very unreliable, and the SBM methods perform very similarly to the 2D environments, with minor degradations in collision intensity and collision-free trajectories for some of them.

SBM + SDF Gradient Including the gradient of the SDF in the sampling for the SBM improves all metrics except distance for all environments I tested. Perhaps the extra distance is actually a good thing because it means that a robot maintains its distance from obstacles in the environment, which is important for safety. However, one thing that should be changed about the way I used it in the experiments is that some parts of the trajectories should not be affected by the gradient of the SDF. These are the start and goal positions or any other position the SBM was conditioned on. As for the time required, the sampling time increases compared to the stand-alone SBM only if the SDF calculation is poorly implemented, as was the case in the 3D environment. Of course, we may not always have access to the SDF. In this case, either the standalone SBM or the StochGPMP with the SBM prior are good options.

5.1 StochGPMP With And Without SBM Prior

This is what I was most interested in and where I saw the greatest likelihood for a useful application of SBMs in motion planning. Since SBMs do not use collision checking, this task must be done by something else if we are to run SBM trajectories on a real system. So what does the data show? Can SBMs be used to generate useful initial trajectories for optimization-based motion planners like StochGPMP? If so, does the performance gain justify the extra effort to train the SBM?

All the experiments I have conducted show the same results for SBMs. SBMs are a powerful tool to generate whole trajectories. To know how an SBM prior affects StochGPMP, we first need to know how StochGPMP performes on its own. The results here are a bit mixed. In the first simple environment, StochGPMP's performance was admirable, albeit slow. In

more complicated tasks, the sampling time and/or failure rate increases sharply. Of course, these results should be viewed with some caution. StochGPMP is a method that benefits from a lot of fine-tuning, so it could have performed better in some of the environments if given enough time and care in fine-tuning.

This is one of the main reasons why a strong prior is so useful for StochGPMP. With a strong prior, there is much less need to fine tune StochGPMP. The reason is that the exploration part of StochGPMP has already been done by the prior.

Apart from the fact that StochGPMP is robust to the choice of hyperparameters, the statistical results of the StochGPMP with the SBM prior are very promising compared to pure StochGPMP.

In both 2D environments, StochGPMP shows significantly more collision-free trajectories with the SBM prior version, taking only a fraction of the time. This effect becomes more pronounced as the planning task becomes more difficult. While in the simple 2D grid environment the performance increase is relatively small, in the second narrow passages 2D environment the number of collision-free trajectories increases by more than 70% when using the SBM prior, while at the same time taking $1/10$ of the time to sample the trajectories. In 3D, the number of collision-free trajectories increases by almost 50% with $1/5$ of time required.

So, in summary, SBMs are very promising in increasing the sampling speed and quality of optimization-based motion planners by using them to generate better initial trajectories.

As a very useful side effect the SBM prior mitigates the need to fine tune StochGPMP.

6 Future Work

Here I will present some unfinished or unpolished work. I think they still have some value since they show promise. Afterwards I will present a couple of general improvements and extensions.

6.1 Franka Panda Proof Of Concept

In this section we will look at what SBMs potentially could do in higher dof settings. The reason this is not in the experiments section is because there were some issues with the choice of environment and the time it takes to generate enough data that simply did not permit me to research this to the point of a full meaningful experiment.

What I was able to do, is to run some of the methods used in the experiments section with the problematic data I had. RRT* at this point was not feasible to run, I am not sure how long it takes to run RRT* in this environment, but after running it for three times as long as in 3D, it still didn't even find an initial solution. The SBM that uses the gradient of the SDF could not be tested since the gradient of the SDF is in taskspace while the SBM is in jointspace. Here we let the SBM generate trajectories in a 7 dof joint space, normalized to $(-1, 1)$. You can see the statistical results in Table 6.1.

Lets talk first about what looks promising and then I'll speculate on what went wrong. The first very promising thing is that the SBM is now faster than RRT connect, which was previously unbeatable in terms of sample time. This also makes it of course a lot faster than StochGPMP (and RRT* since RRT connect is a lower bound for it). Using it as a prior for StochGPMP also is a lot faster than pure StochGPMP. Both SBM methods also produce shorter and smoother trajectories.

Now the problem is the ability to produce collision free trajectories. On paper it looks like the SBM methods outperfor StochGPMP, but in reality either all trajectories by the SBM

were collision free or none of them for a given context. There are a couple of reasons I can think of why this might happen.

The first one is the choice of environment and the second one is the means of generating the data. You can see an example context in the environment in figure 6.1. What I speculate is wrong here is that most examples presented during training are rather trivial contexts without a chance of collision. This means that during most training steps the model is trained to simply interpolate linearly, or close to linearly. This effect is is strongly reinforced through the use of GPMP to generate data, which results in trajectories that are very close to the obstacles without a lot of variance.

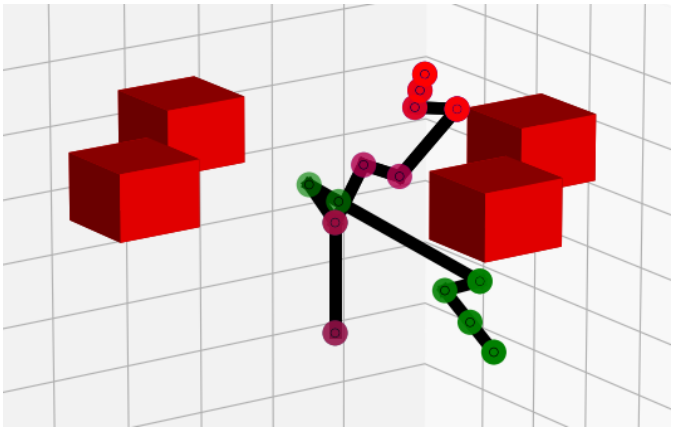
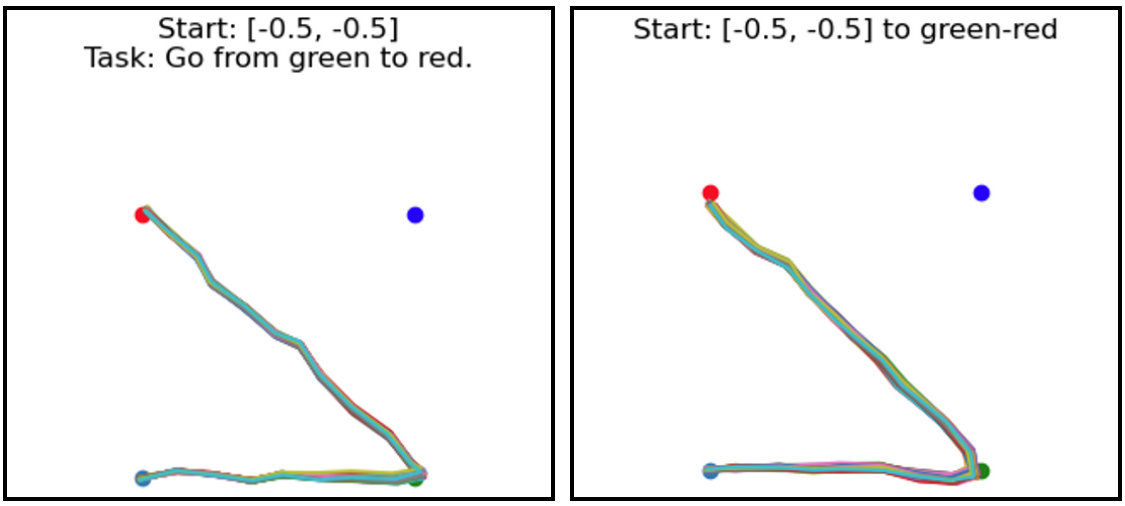


Figure 6.1: An example context of the Franka Panda environment. Green is the start and red is the goal.

It is my opinion that this can work really well if the training data is generated with RRT* in an environment that has way more samples that do not connect the start and goal linearly. I also think that because of the very high potential for large speedups in sampling time this is worth pursuing further, but since RRT* takes so long to generate samples in this environment it is not feasible for me to include this in my thesis.



(a) Example of language conditioned SBM samples. The language specifies the task, the positions of the points are provided as context.

(b) Example of image and context conditioned SBM samples. The image provides the positions of the colored points which are encoded through a CNN.

Figure 6.2: Two examples of other types of conditionings. During training the positions of the points were randomized for both of them.

6.2 Different Context Features

6.2.1 Target And Language Encoding

During the early exploration phases of this thesis I first looked into contexts to condition an SBM on. These were discontinued as I moved on to higher dimensional data. Two that worked rather well are visualised in figure 6.2. In figure 6.2a you can see some samples generated with a SBM conditioned on BERT language features. This allows generalising to similar sentences with the same meaning. The second one in figure 6.2b uses a CNN to encode the positions of the colored point, which are only provided through the image. The positions of the colored points and the start state were randomized during training for both of them.

These are of course not fully explored experiments yet, but both have their merit for future work. Language is a natural way for humans to give instructions to a robot and if a robot can identify possible targets by itself this can be a big help too.

6.2.2 Obstacle Encoding

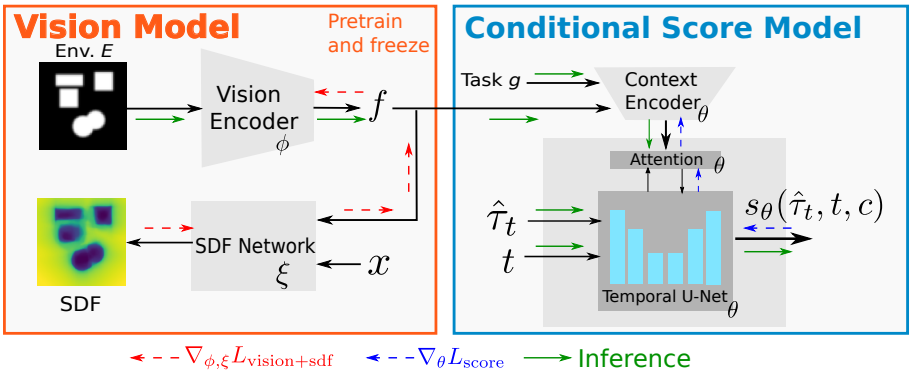


Figure 6.3: A vision encoder is learned using an SDF loss. The task and vision embeddings are concatenated as inputs to a context encoder and used to condition the SBM. The attention layer is optional. The architecture of the SBM is the same as before.

In subsection 6.2.1 we already saw that other encodings can be passed as a conditioning. Here the point is to encode the whole environment. Specifically the obstacles of the environment so the SBM can generalise to different settings.

Learning The Vision Encoder The overall architecture is depicted in figure 6.3. The **vision model** takes as input an image E (black-and-white occupancy map), and produces latent features $\boldsymbol{f}_{\boldsymbol{\phi}}(E)$. With access to the distribution of environments and ground-truth signed distance values $\text{sdf}_{\text{gt}}(E, \boldsymbol{x})$ for a point $\boldsymbol{x} \in \mathbb{R}^2$, we pretrain the vision encoder with parameters $\boldsymbol{\phi}$, by minimizing an SDF L_1-loss

$$\mathcal{L}_{\text{vision + sdf}}(\boldsymbol{\phi}, \boldsymbol{\xi}) = \mathbb{E}_{E\sim p(E), \boldsymbol{x}\sim p(\boldsymbol{x}|E)} \left[\left| \text{SDF}_{\text{gt}}(E, \boldsymbol{x}) - \hat{\text{SDF}}_{\boldsymbol{\xi}}(\boldsymbol{f}_{\boldsymbol{\phi}}(E), \boldsymbol{x}) \right| \right], \tag{6.1}$$

where $\hat{\text{sdf}}_{\boldsymbol{\xi}}$ is the learned SDF. The latent representation $\boldsymbol{f}$ could also be learned with a VAE, but this model would be larger (due to the decoder) and would need more care in designing it. Also it would not give access to the SDF, which could then be used to pre-rank the trajectories without explicit collision checking. In a real world scenario we are able to execute only one trajectory. A simple heuristic metric to select and rank trajectories – discard the ones where any point along them has a negative SDF value according to our model, which means collision with an obstacle. Afterwards, we select the trajectory that

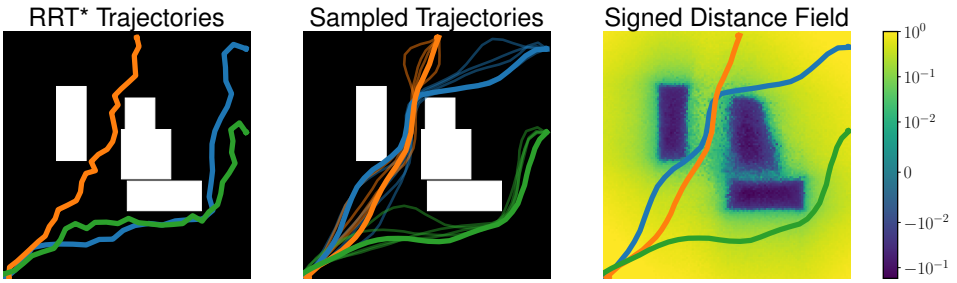


Figure 6.4: This figure shows a sample from the validation set of the RECTANGLES AND CIRCLES environments. The environment is 64×64 image, where white is where an obstacle is present. The left figure shows trajectories generated with RRT* for different goal positions (top right area). Thinner trajectories in the center are sampled with the conditioned SBM. Thicker trajectories are the ones considered after ranking. The right plot shows the learned SDF and the selected trajectories.

has the shortest path. Even though this does not guarantee the optimal trajectory is found, it can be a good first heuristic prior for motion optimization methods. You can see some preliminary results of the generated SDF and trajectories sampled with the conditional SBM with a never seen environment in Figure 6.4.

6.3 Progressive Distillation

Salimans and Ho [47] proposes a method to distill two steps of a SBM into just one in a process called Progressive Distillation. After an initial SBM is trained normally, they use this model as a teacher for further student models to be trained. Here the student model, initialised with the parameters of the teacher model is trained on the integrated output of the teacher model after N time-steps. This student model then learns to do the same thing the teacher does, just with fewer steps.

This can be repeated in multiple distillation steps with the student becoming the teacher and so forth.

The promise of using this technique is of course faster sampling times, which is especially useful in realtime applications.

6.4 Minor Improvements And Extensions

Collision Check After Splining As mentioned earlier, the trajectories are splined before they are used in training. This is necessary and even useful since RRT generated trajectories do not have a uniform horizon, and using a B-spline makes the training data smoother. The training data would be cleaner and should result in better samples if this splining would be done before the collision checking. This way the trajectories would be truly collision free, instead of getting artifacts like corner cutting of obstacles.

Extension To SE-03 This one is self explanatory. Including rotation in the data makes generation more tricky, since it increases dimensions. Since SBM can handle high dimensions this could be worth investigating.

	Sampling Time (s)	Collision Free (%) ↑	Collision Intensity (%)	Distance	Cosine Sim
RRT connect	1.07 ±0.91	100.00 ±0.00	0.00 ±0.00	10.87 ±3.29	0.89 ±0.11
StochGPMP	39.35 ±0.04	24.46 ±23.37	19.44 ±9.42	7.19 ±1.07	0.04 ±0.01
SBM	0.32 ±0.12	36.56 ±47.90	18.99 ±17.96	2.14 ±0.53	0.01 ±0.00
SBM Prior + StochGPMP	3.28 ±0.01	36.16 ±42.64	18.82 ±16.24	5.38 ±1.42	0.01 ±0.00

Table 6.1: Preliminary statistics for the 7 dof Franka Panda Environment.

Bibliography

[1] Hagai Attias. Planning by probabilistic inference. In Christopher M. Bishop and Brendan J. Frey, editors, *Proceedings of the Ninth International Workshop on Artificial Intelligence and Statistics*, volume R4 of *Proceedings of Machine Learning Research*, pages 9–16. PMLR, 03–06 Jan 2003. URL https://proceedings.mlr.press/r4/attias03a.html. Reissued by PMLR on 01 April 2021.

[2] Mayur J. Bency, Ahmed Hussain Qureshi, and Michael C. Yip. Neural path planning: Fixed time, near-optimal path generation via oracle imitation. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2019, Macau, SAR, China, November 3-8, 2019*, pages 3965–3972. IEEE, 2019. doi: 10.1109/IROS40897.2019.8968089. URL https://doi.org/10.1109/IROS40897.2019.8968089.

[3] Constantinos Chamzas, Anshumali Shrivastava, and Lydia E. Kavraki. Using local experiences for global motion planning. In *International Conference on Robotics and Automation, ICRA 2019, Montreal, QC, Canada, May 20-24, 2019*, pages 8606–8612. IEEE, 2019. doi: 10.1109/ICRA.2019.8794317. URL https://doi.org/10.1109/ICRA.2019.8794317.

[4] Constantinos Chamzas, Zachary K. Kingston, Carlos Quintero-Peña, Anshumali Shrivastava, and Lydia E. Kavraki. Learning sampling distributions using local 3d workspace decompositions for motion planning in high dimensions. In *IEEE International Conference on Robotics and Automation, ICRA 2021, Xi'an, China, May 30 - June 5, 2021*, pages 1283–1289. IEEE, 2021. doi: 10.1109/ICRA48506.2021.9561104. URL https://doi.org/10.1109/ICRA48506.2021.9561104.

[5] Ricky T. Q. Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud. Neural ordinary differential equations. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018. URL https://proceedings.neurips.cc/paper/2018/file/69386f6bb1dfed68692a24c8686939b9-Paper.pdf.

[6] Richard Cheng, Krishna Shankar, and Joel W. Burdick. Learning an optimal sampling distribution for efficient motion planning. In *IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2020, Las Vegas, NV, USA, October 24, 2020 - January 24, 2021*, pages 7485–7492. IEEE, 2020. doi: 10.1109/IROS45743.2020.9341245. URL https://doi.org/10.1109/IROS45743.2020.9341245.

[7] Congyue Deng, Or Litany, Yueqi Duan, Adrien Poulenard, Andrea Tagliasacchi, and Leonidas J. Guibas. Vector neurons: A general framework for so(3)-equivariant networks. In *2021 IEEE/CVF International Conference on Computer Vision, ICCV 2021, Montreal, QC, Canada, October 10-17, 2021*, pages 12180–12189. IEEE, 2021. doi: 10.1109/ICCV48922.2021.01198. URL https://doi.org/10.1109/ICCV48922.2021.01198.

[8] Pete Florence, Corey Lynch, Andy Zeng, Oscar Ramirez, Ayzaan Wahid, Laura Downs, Adrian Wong, Johnny Lee, Igor Mordatch, and Jonathan Tompson. Implicit behavioral cloning. *Conference on Robot Learning (CoRL)*, 2021.

[9] Jonathan D. Gammell, Siddhartha S. Srinivasa, and Timothy D. Barfoot. 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, Chicago, IL, USA, September 14-18, 2014*, pages 2997–3004. IEEE, 2014. doi: 10.1109/IROS.2014.6942976. URL https://doi.org/10.1109/IROS.2014.6942976.

[10] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 27. Curran Associates, Inc., 2014. URL https://proceedings.neurips.cc/paper/2014/file/5ca3e9b122f61f8f06494c97b1afccf3-Paper.pdf.

[11] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, NIPS'20, Red Hook, NY, USA, 2020. Curran Associates Inc. ISBN 9781713829546.

[12] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997. doi: 10.1162/neco.1997.9.8.1735.

[13] Aapo Hyvärinen. Estimation of non-normalized statistical models by score matching. *Journal of Machine Learning Research*, 6(24):695–709, 2005. URL http://jmlr.org/papers/v6/hyvarinen05a.html.

[14] Jeffrey Ichnowski, Yahav Avigal, Vishal Satish, and Ken Goldberg. Deep learning can accelerate grasp-optimized motion planning. *Sci. Robotics*, 5(48):7710, 2020. doi: 10.1126/scirobotics.abd7710. URL https://doi.org/10.1126/scirobotics.abd7710.

[15] Michael Janner, Yilun Du, Joshua Tenenbaum, and Sergey Levine. Planning with diffusion for flexible behavior synthesis. In *International Conference on Machine Learning*, 2022.

[16] Jacob J. Johnson, Uday S. Kalra, Ankit Bhatia, Linjun Li, Ahmed H. Qureshi, and Michael C. Yip. Motion planning transformers: A motion planning framework for mobile robots, 2021. URL https://arxiv.org/abs/2106.02791.

[17] M. Kalakrishnan, S. Chitta, E. Theodorou, P. Pastor, and S. Schaal. Stomp: Stochastic trajectory optimization for motion planning. In *IEEE International Conference on Robotics and Automation (ICRA)*, Shanghai, China, May 9-13, 2011. URL http://www-clmc.usc.edu/publications/K/kalakrishnan-ICRA2011.pdf. clmc.

[18] M. Kalakrishnan, S. Chitta, E. Theodorou, P. Pastor, and S. Schaal. Stomp: Stochastic trajectory optimization for motion planning. In *IEEE International Conference on Robotics and Automation (ICRA)*, Shanghai, China, May 9-13, 2011. URL http://www-clmc.usc.edu/publications/K/kalakrishnan-ICRA2011.pdf. clmc.

[19] L.E. Kavraki, P. Svestka, J.-C. Latombe, and M.H. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation*, 12(4):566–580, 1996. doi: 10.1109/70.508439.

[20] Gwanghyun Kim, Taesung Kwon, and Jong Chul Ye. Diffusionclip: Text-guided diffusion models for robust image manipulation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2426–2435, June 2022.

[21] Diederik P. Kingma and Max Welling. Auto-encoding variational bayes. In Yoshua Bengio and Yann LeCun, editors, *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*, 2014. URL http://arxiv.org/abs/1312.6114.

[22] Dorothea Koert, Guilherme Maeda, Rudolf Lioutikov, Gerhard Neumann, and Jan Peters. Demonstration based trajectory optimization for generalizable robot motions. In *2016 IEEE-RAS 16th International Conference on Humanoid Robots (Humanoids)*, pages 515–522, 2016. doi: 10.1109/HUMANOIDS.2016.7803324.

[23] J.J. Kuffner and S.M. LaValle. Rrt-connect: An efficient approach to single-query path planning. In *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No.00CH37065)*, volume 2, pages 995–1001 vol.2, 2000. doi: 10.1109/ROBOT.2000.844730.

[24] Jean-Claude Latombe. *Robot Motion Planning*. Kluwer Academic Publishers, USA, 1991. ISBN 0792391292.

[25] S. M. LaValle. *Planning Algorithms*. Cambridge University Press, Cambridge, U.K., 2006. Available at http://planning.cs.uiuc.edu/.

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

[27] Sergey Levine. Reinforcement learning and control as probabilistic inference: Tutorial and review, 2018. URL `https://arxiv.org/abs/1805.00909`.

[28] Mustafa Mukadam, Xinyan Yan, and Byron Boots. Gaussian process motion planning. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 9–15, 2016. doi: 10.1109/ICRA.2016.7487091.

[29] Mustafa Mukadam, Jing Dong, Xinyan Yan, Frank Dellaert, and Byron Boots. Continuous-time Gaussian process motion planning via probabilistic inference. volume 37, pages 1319–1340, 2018.

[30] Joseph Ortiz, Alexander Clegg, Jing Dong, Edgar Sucar, David Novotny, Michael Zollhoefer, and Mustafa Mukadam. isdf: Real-time neural signed distance fields for robot perception. In *Robotics: Science and Systems*, 2022.

[31] Joaquim Ortiz-Haro, Jung-Su Ha, Danny Driess, and Marc Toussaint. Structured deep generative models for sampling on constraint manifolds in sequential manipulation. In Aleksandra Faust, David Hsu, and Gerhard Neumann, editors, *Proceedings of the 5th Conference on Robot Learning*, volume 164 of *Proceedings of Machine Learning Research*, pages 213–223. PMLR, 08–11 Nov 2022. URL `https://proceedings.mlr.press/v164/ortiz-haro22a.html`.

[32] Alexandros Paraschos, Christian Daniel, Jan Peters, and Gerhard Neumann. Using probabilistic movement primitives in robotics. *Autonomous Robots*, 42(3):529–551, 2018.

[33] Jeong Joon Park, Peter Florence, Julian Straub, Richard Newcombe, and Steven Lovegrove. Deepsdf: Learning continuous signed distance functions for shape representation. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[34] Jan Peters, Katharina Mulling, and Yasemin Altun. Relative entropy policy search. In *Conference on Artificial Intelligence (AAAI)*, 2010.

[35] Jan Peters, Katharina Mülling, and Yasemin Altün. Relative entropy policy search. In *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence*, AAAI'10, page 1607–1612. AAAI Press, 2010.

[36] Thomas Power and Dmitry Berenson. Variational inference mpc using normalizing flows and out-of-distribution projection. *Robotics: Science and Systems 2022*. doi: 10.15607/RSS.2022.XVIII.027. URL https://par.nsf.gov/biblio/10348466.

[37] Charles Ruizhongtai Qi, Hao Su, Kaichun Mo, and Leonidas J. Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*, pages 77–85. IEEE Computer Society, 2017. doi: 10.1109/CVPR.2017.16. URL https://doi.org/10.1109/CVPR.2017.16.

[38] Ahmed Hussain Qureshi and Michael C. Yip. Deeply informed neural sampling for robot motion planning. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2018, Madrid, Spain, October 1-5, 2018*, pages 6582–6588. IEEE, 2018. doi: 10.1109/IROS.2018.8593772. URL https://doi.org/10.1109/IROS.2018.8593772.

[39] Ahmed Hussain Qureshi, Anthony Simeonov, Mayur J. Bency, and Michael C. Yip. Motion planning networks. In *International Conference on Robotics and Automation, ICRA 2019, Montreal, QC, Canada, May 20-24, 2019*, pages 2118–2124. IEEE, 2019. doi: 10.1109/ICRA.2019.8793889. URL https://doi.org/10.1109/ICRA.2019.8793889.

[40] Ahmed Hussain Qureshi, Jiangeng Dong, Austin Choe, and Michael C. Yip. Neural manipulation planning on constraint manifolds. *IEEE Robotics Autom. Lett.*, 5(4):6089–6096, 2020. doi: 10.1109/LRA.2020.3010220. URL https://doi.org/10.1109/LRA.2020.3010220.

[41] Ahmed Hussain Qureshi, Yinglong Miao, Anthony Simeonov, and Michael C. Yip. Motion planning networks: Bridging the gap between learning-based and classical motion planners. *IEEE Trans. Robotics*, 37(1):48–66, 2021. doi: 10.1109/TRO.2020.3006716. URL https://doi.org/10.1109/TRO.2020.3006716.

[42] Aditya Ramesh, Prafulla Dhariwal, Alex Nichol, Casey Chu, and Mark Chen. Hierarchical text-conditional image generation with clip latents, 2022. URL https://arxiv.org/abs/2204.06125.

[43] Muhammad Asif Rana, Mustafa Mukadam, Seyed Reza Ahmadzadeh, Sonia Chernova, and Byron Boots. Towards robust skill generalization: Unifying learning from demonstration and motion planning. In *1st Annual Conference on Robot Learning, CoRL 2017, Mountain View, California, USA, November 13-15, 2017, Proceedings*, volume 78 of *Proceedings of Machine Learning Research*, pages 109–118. PMLR, 2017. URL http://proceedings.mlr.press/v78/rana17a.html.

[44] Nathan Ratliff, Matt Zucker, J. Andrew Bagnell, and Siddhartha Srinivasa. Chomp: Gradient optimization techniques for efficient motion planning. In *2009 IEEE International Conference on Robotics and Automation*, pages 489–494, 2009. doi: 10.1109/ROBOT.2009.5152817.

[45] Danilo Jimenez Rezende and Shakir Mohamed. Variational inference with normalizing flows. In Francis R. Bach and David M. Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, volume 37 of *JMLR Workshop and Conference Proceedings*, pages 1530–1538. JMLR.org, 2015. URL http://proceedings.mlr.press/v37/rezende15.html.

[46] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models, 2021.

[47] Tim Salimans and Jonathan Ho. Progressive distillation for fast sampling of diffusion models. *CoRR*, abs/2202.00512, 2022. URL https://arxiv.org/abs/2202.00512.

[48] John Schulman, Jonathan Ho, Alex X. Lee, Ibrahim Awwal, Henry Bradlow, and Pieter Abbeel. Finding locally optimal, collision-free trajectories with sequential convex optimization. In Paul Newman, Dieter Fox, and David Hsu, editors, *Robotics: Science and Systems IX, Technische Universität Berlin, Berlin, Germany, June 24 - June 28, 2013*, 2013. doi: 10.15607/RSS.2013.IX.031. URL http://www.roboticsproceedings.org/rss09/p31.html.

[49] Anthony Simeonov, Yilun Du, Andrea Tagliasacchi, Joshua B. Tenenbaum, Alberto Rodriguez, Pulkit Agrawal, and Vincent Sitzmann. Neural descriptor fields: Se(3)-equivariant object representations for manipulation. In *2022 International Conference on Robotics and Automation, ICRA 2022, Philadelphia, PA, USA, May 23-27, 2022*, pages 6394–6400. IEEE, 2022. doi: 10.1109/ICRA46639.2022.9812146. URL https://doi.org/10.1109/ICRA46639.2022.9812146.

[50] Yang Song and Stefano Ermon. Generative modeling by estimating gradients of the data distribution. In Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 11895–11907, 2019. URL https://proceedings.neurips.cc/paper/2019/hash/3001ef257407d5a371a96dcd947c7d93-Abstract.html.

[51] Yang Song, Jascha Sohl-Dickstein, Diederik P Kingma, Abhishek Kumar, Stefano Ermon, and Ben Poole. Score-based generative modeling through stochastic differential equations. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=PxTIG12RRHS.

[52] Robin Strudel, Ricardo Garcia Pinel, Justin Carpentier, Jean-Paul Laumond, Ivan Laptev, and Cordelia Schmid. Learning obstacle representations for neural motion planning. In Jens Kober, Fabio Ramos, and Claire J. Tomlin, editors, *4th Conference on Robot Learning, CoRL 2020, 16-18 November 2020, Virtual Event / Cambridge, MA, USA*, volume 155 of *Proceedings of Machine Learning Research*, pages 355–364. PMLR, 2020. URL https://proceedings.mlr.press/v155/strudel21a.html.

[53] Marc Toussaint. Robot trajectory optimization using approximate inference. In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, page 1049–1056, New York, NY, USA, 2009. Association for Computing Machinery. ISBN 9781605585161. doi: 10.1145/1553374.1553508. URL https://doi.org/10.1145/1553374.1553508.

[54] J. Urain, A.T. Le, A. Lambert, G. Chalvatzaki, B. Boots, and J. Peters. Learning implicit priors for motion optimization. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2022. URL https://www.ias.informatik.tu-darmstadt.de/uploads/Team/AnThaiLe/iros2022_ebmtrajopt.pdf.

[55] Julen Urain, Niklas Funk, Jan Peters, and Georgia Chalvatzaki. Se(3)-diffusionfields: Learning smooth cost functions for joint grasp and motion optimization through diffusion, 2022. URL https://arxiv.org/abs/2209.03855.

[56] Pascal Vincent. A connection between score matching and denoising autoencoders. *Neural Computation*, 23(7):1661–1674, 2011. doi: 10.1162/NECO_a_00142.

[57] Jiankun Wang, Wenzheng Chi, Chenming Li, Chaoqun Wang, and Max Q.-H. Meng. Neural rrt*: Learning-based optimal path planning. *IEEE Transactions on Automation Science and Engineering*, 17(4):1748–1758, 2020. doi: 10.1109/TASE.2020.2976560.

[58] Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E. Sarma, Michael M. Bronstein, and Justin M. Solomon. Dynamic graph CNN for learning on point clouds. *ACM Trans. Graph.*, 38(5):146:1–146:12, 2019. doi: 10.1145/3326362. URL https://doi.org/10.1145/3326362.

[59] J. Watson and J. Peters. Inferring smooth control: Monte carlo posterior policy iteration with gaussian processes. In *Conference on Robot Learning*, 2022. URL https://www.ias.informatik.tu-darmstadt.de/uploads/Team/JoeWatson/watson22corl.pdf.

[60] Joe Watson, Hany Abdulsamad, and Jan Peters. Stochastic optimal control as approximate input inference. In Leslie Pack Kaelbling, Danica Kragic, and Komei Sugiura, editors, *3rd Annual Conference on Robot Learning, CoRL 2019, Osaka, Japan, October 30 - November 1, 2019, Proceedings*, volume 100 of *Proceedings of Machine Learning Research*, pages 697–716. PMLR, 2019. URL http://proceedings.mlr.press/v100/watson20a.html.

[61] Max Welling and Yee Whye Teh. Bayesian learning via stochastic gradient langevin dynamics. In *Proceedings of the 28th International Conference on International Conference on Machine Learning*, ICML'11, page 681–688, Madison, WI, USA, 2011. Omnipress. ISBN 9781450306195.

[62] Clark Zhang, Jinwook Huh, and Daniel D. Lee. Learning implicit sampling distributions for motion planning. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2018, Madrid, Spain, October 1-5, 2018*, pages 3654–3661. IEEE, 2018. doi: 10.1109/IROS.2018.8594028. URL https://doi.org/10.1109/IROS.2018.8594028.

[63] Matt Zucker, Nathan Ratliff, Anca D. Dragan, Mihail Pivtoraiko, Matthew Klingensmith, Christopher M. Dellin, J. Andrew Bagnell, and Siddhartha S. Srinivasa. Chomp: Covariant hamiltonian optimization for motion planning. *The International Journal of Robotics Research*, 32(9-10):1164–1193, 2013. doi: 10.1177/0278364913488805. URL https://doi.org/10.1177/0278364913488805.