Learning and Inference for Adaptable Manipulation Planning

by

Thomas Power

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Robotics)
in The University of Michigan
2023

Doctoral Committee:

Associate Professor Dmitry Berenson, Chair
Associate Professor Ram Vasudevan
Assistant Professor Nima Fazeli
Professor Georg Martius

Thomas Power

tpower@umich.edu

ORCID iD: 0000-0002-2439-3262

ACKNOWLEDGEMENTS

I owe thanks to a host of people without whom this thesis would not have been possible. Firstly, I would like to thank my advisor Dmitry Berenson who gave me the opportunity to pursue this PhD, and who provided valuable feedback throughout my time in Michigan. I would also like to thank my committee Nima Fazeli, Georg Martius and Ram Vasudevan for their feedback. Thank you also to my collaborators Rana Soltani Zarrin, Soshi Iba, and Sergio Aguilera. I learned a great deal during my time at Honda Research Institute and have enjoyed working with you since.

I have had the pleasure of working with many great people at the Autonomous Robotic Manipulation Lab at Michigan. In particular, Peter Mitrano and Sheng Zhong, with whom I have shared the PhD journey these last 6 years. I am grateful for all the friends I have made along the way; those in Michigan who have made Ann Arbor home, and also those who have always given me somewhere I look forward to returning to. My friends have helped to ground me, reminding me that a world exists outside of robotics research. If you are reading this thesis you may benefit from the same occasional reminder.

Finally, I would like to thank my family, in particular my mother Tracey, who has always believed in me and guided me to where I am today. Lastly, I would like to thank Tuuli-Anna, who challenges me to always strive to be better but loves me regardless of the outcome. This has been particularly embodied in her proofreading of this thesis.

TABLE OF CONTENTS

LIST OF FIGURES

Figure

LIST OF TABLES

Table

LIST OF APPENDICES

Appendix

ABSTRACT

A central challenge for developing general-purpose robot assistants is the development of algorithms for robot manipulation that can perform a wide range of tasks across a diverse set of environments. In this thesis, I develop planning and trajectory optimization methods that can adapt to new and unforeseen systems. The key to these methods is the ability of robots to learn from experience and reason about related uncertainty. Using modern machine learning and approximate probabilistic inference techniques, the work I present in this thesis improves the ability of planning methods to do so.

Probabilistic inference is useful in two ways. First, by using a probabilistic framing, probabilities can be used as a way of expressing confidence in our current models. I develop a method that learns to predict the uncertainty of a given dynamics model with a small amount of data collected online and avoids areas where the model is uncertain. I also propose an approach that learns a generative model of control sequences to complete a given task. I demonstrate that we can detect and adapt this generative model to situations where the environment differs from the training environments.

Second, I incorporate probabilistic inference into the proposed methods by viewing planning itself as an inference problem. By framing planning as inference, we construct probability distributions over trajectories. This framework allows me to develop a method that views constrained trajectory optimization as inference, generating diverse sets of constraint-satisfying trajectories for completing manipulation tasks. This allows improved adaptation to online disturbances, since at any given time, there is a set of trajectories to select from. I demonstrate the effectiveness

of this method on several different tasks, including a 7DoF manipulator turning a wrench and a 16DoF multi-fingered hand turning a precision screwdriver.

The methods I present in this thesis contribute to the development of adaptable algorithms for robotic manipulation for the next generation of general-purpose robot assistants.

CHAPTER I

Introduction

To realize the transformative potential of robotics we need to develop autonomous systems with broad capabilities that can be reliably deployed across diverse domains such as manufacturing, logistics, healthcare, and people's homes. Fundamental to developing general robot agents is manipulation - how robots interact with and change the physical world. A central challenge is developing algorithms for robot manipulation that can perform a wide range of manipulation tasks across a diverse set of real-world environments.

Motion planning and trajectory optimization have been widely used to construct plans for completing non-trivial long-horizon manipulation tasks. These methods are popular because they provide general algorithmic frameworks that can be used to describe and complete a wide variety of different tasks, such as pick-and-place [107], opening doors [14] and rope manipulation [103].

While planning offers a very general way of framing different manipulation tasks, there are many practical challenges in applying these techniques broadly. Planning methods typically require knowledge of the environment, current system configuration, and system dynamics. These requirements may not be met in practice due to imperfect perception, noise, and modeling errors. Here, the generality of the planning framework belies practical difficulties. Although planning can solve an extremely broad range of tasks in principle, building perception systems and models that encompass all of these tasks becomes extremely difficult. My position in this

thesis is that even if perception and system identification techniques continue to improve, a deployed robot will always find itself in situations for which it has not been adequately prepared. This is due to the sheer variety of tasks and environments in which we expect robots to operate. In this thesis, I propose planning methods that have the ability to adapt to new and unforeseen environments and systems.

A common and effective strategy for improving the adaptability of planners is to re-plan trajectories during execution. Global sample-based motion planning methods [79, 69, 71, 14, 105, 58, 120] are very effective at generating feasible paths. However, they are typically too computationally intensive to perform online re-planning. While existing local trajectory optimization methods [101, 90, 17, 101, 56, 45, 171] are often much faster and aim to find a single locally-optimal trajectory, they may fail to find feasible solutions based on the initialization. This is particularly problematic when re-solving the optimization problem online under limited computation time when disturbances can lead to the previous solution becoming a poor initialization for the current optimization problem. There have been many methods that incorporate learning to accelerate planning and trajectory optimization [66, 178, 57, 128, 89, 81, 21], but none of these methods include strategies for when the planner is out-of-distribution of the learned components.

The methods presented in this thesis use modern machine learning and approximate probabilistic inference techniques to improve the ability of planning methods to adapt to new and unforeseen environments and systems. Given that the goal is to build adaptable planning systems, it is natural to leverage techniques that learn from experience. Probabilistic inference is useful in two ways. Firstly, by using a probabilistic framing, probabilities can be used to express confidence in the current models. This idea is used in both Chapter II, where the method avoids areas where there is low confidence in the models, and in Chapter III where we use it to determine if an environment is outside of the training distribution. The second way I will incorporate probabilistic inference into the proposed methods is by viewing planning itself as an inference problem. Framing planning as inference results in constructing probability distributions over trajectories. By computing probability distributions over trajectories, there is a set of trajectories to select from in the face of disturbances

or unseen situations. I use the ideas of framing planning as inference in Chapters III, IV, VI and V.

I will now give a summary of the structure of this thesis. Each chapter begins with an introduction to the relevant related work. In Chapter II I introduce a method for planning for manipulation with a highly miss-specified model from image observations, which relies on learning a Gaussian Process to estimate the model uncertainty. In Chapter III I present a method that learns a generative model of control sequences to complete a given task. I also present a method for adapting this generative model to Out-of-Distribution (OOD) environments. In Chapter IV I present a method that views constrained trajectory optimization as inference and generates diverse sets of constraint-satisfying trajectories for completing manipulation tasks. In Chapter V, I demonstrate the application of this method to challenging dexterous manipulation tasks with a multi-fingered hand. Finally, in Chapter VI, I propose learning a generative model of constraint-satisfying trajectories for manipulation planning which is used to initialize the trajectory optimization method introduced in Chapter IV.

This thesis makes the following contributions:

- A method that uses a simple miss-specified model and learns where this model is accurate for image-based control of a system with complex dynamics
- A method that learns a generative model of control sequences, and can adapt to OOD environments
- A method that generates diverse constraint-satisfying trajectories for manipulation, and the application of this method to challenging, highly constrained dexterous manipulation tasks
- A method for learning a generative model of constraint-satisfying trajectories, from which trajectories can be sampled and used to initialize a constrained trajectory optimization algorithm

CHAPTER II

Data-efficient Learning for Controlling Complex Systems with Simple Models

When manipulating a novel object with complex dynamics, a state representation is not always available, for example for deformable objects. Learning both a representation and dynamics from observations requires large amounts of data. We propose Learned Visual Similarity Predictive Control (LVSPC), a novel method for data-efficient learning to control systems with complex dynamics and high-dimensional state spaces from images. LVSPC leverages a given simple model approximation from which image observations can be generated. We use these images to train a perception model that estimates the simple model state from observations of the complex system online. We then use data from the complex system to fit the parameters of the simple model and learn where this model is inaccurate, also online. Finally, we use Model Predictive Control and bias the controller away from regions where the simple model is inaccurate and thus where the controller is less reliable. We evaluate LVSPC on two tasks; manipulating a tethered mass and a rope. We find that our method performs comparably to state-of-the-art reinforcement learning methods with an *order of magnitude less data*. LVSPC also completes the rope manipulation task on a real robot with 80% success rate after only 10 trials, despite using a perception system trained only on images from simulation. [1]

[1]The work in this chapter was published in [122].

2.1 Introduction

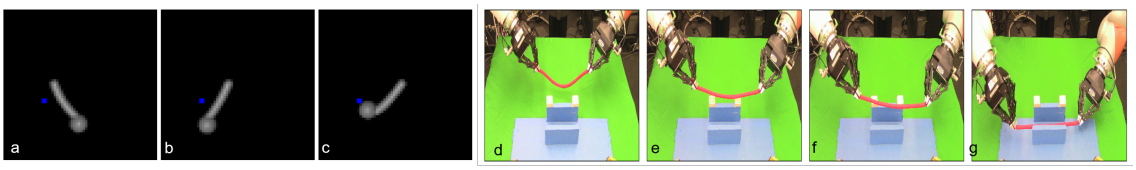


Figure 2.1: *(a-c)*: LVSPC controlling a tethered mass to a desired position (blue) from images by treating it as a cart-pole; *(d-g)*: LVSPC brings a rope to a target location in a narrow passage between two obstacles while avoiding protrusions by treating the rope as a rigid object. The robot starts with the rope slack but pulls it taut to keep the approximation more accurate, allowing it to complete the task.

While recent machine learning methods have been effective for many manipulation tasks, they rely on access to large datasets of the system being manipulated [170, 160, 1]. Yet in many scenarios we do not have time to gather extensive training data with an object before performing a task. Sim-to-real transfer has been used to fine-tune parameters on limited real-world data when the real object is similar to those used in simulation [59, 22], but these methods struggle if the objects are significantly different. We would like to use prior knowledge about the object to reduce the data required for learning, but the question of *how* to effectively use prior knowledge when encountering a *novel* object remains open.

This chapter addresses how to leverage dynamics models of simple systems when learning to control much more complex, but related, systems online. While it is possible to learn dynamics using only online data (e.g. [49]), we wish to use our knowledge of a simple model to make the learning much more data-efficient, and thus practical for real-world application. For example, consider a tethered mass being swung by a gripper (Figure 2.1). The dynamics of the system are complex and require a great deal of data to learn. However, if we treat the system as a cart with a rigid pendulum, we can predict the dynamics fairly accurately *for some subset of the state-action space*. We can exploit this subset to perform tasks such as bringing the mass to a target, even without a globally-accurate dynamics model. Simple models are often used in this way, for example in deformable object manipulation [104, 103]

and control for humanoids [127].

To use knowledge of the dynamics of the simple model to control the more complex true system, we must know which states of the complex system correspond to which states of the simple system. What makes this problem especially difficult is that, while we can design a useful state representation for the simple system offline, we do not know what state representation to use for the complex system, so we cannot explicitly define a correspondence between states.

Our key insight for overcoming this problem is that the simple system (and its state representation) is a good approximation of the complex system when it gives rise to similar image observations to the complex system. By using a metric for observation similarity that reasons about uncertainty we can build a controller for the complex system and also learn where our approximation is inaccurate (to avoid visiting those parts of the state space). By utilizing domain randomization during training, we enable a single simple system state to elicit a wide variety of image observations; i.e. shapes, colors, and obstacles can vary while still producing an image we consider to be *visually-similar*. We use online system identification to estimate the parameters of the simple model, however, deciding which class of simple model to use for a given task is not within the scope of this chapter. Here we made this decision manually but seek to automate selecting the class of simple model in future work.

This chapter makes the following contributions: 1) Learned Visual Similarity Predictive Control (LVSPC), a novel framework for learning how to perform manipulation tasks with a complex system given only a simple model and images from a small number of trials online; 2) Evaluation of LVSPC on manipulating a tethered mass (using a cart-pole as a simple model) and a rope (using a rigid body as a simple model) (See Fig. 2.1) in simulation, showing large improvements in data-efficiency over baselines (PlaNet [49] and CURL [86]). LVSPC also completes the rope manipulation task *on a real robot* with 80% success rate after only 10 trials.

LVSPC consists of two phases: 1) Offline, we train an ensemble Convolutional Neural Network (CNN) perception system on image observations of the simple system, outputting an estimate of the simple system's state. 2) Online, given image

observations of the complex system, we do system identification to estimate parameters of the simple system dynamics and learn a Gaussian Process (GP) that predicts where the simple model is accurate. We use the simple model and the GP to track the object via a Gaussian Process Unscented Kalman Filter (GPUKF) [76] and perform control via Model Predictive Path Integral Control (MPPI) [167], biasing the system away from inaccurate transitions.

2.2 Related Work

2.2.1 Dynamics from Images

Learning-based approaches using dynamics models for control with images observations have included learning dynamics models directly in image space [170, 1, 41]. Dynamics in image space are highly complex, and these methods require large amounts of data. Other methods learn dynamics in a lower-dimensional latent space [11, 160, 49, 50]. None of these methods incorporate prior knowledge. SE3-PoseNets [19] learn dynamics in pose-space from point cloud data. [174] use the positions of a set of ordered points as the representation of a rope and pre-trains a state estimator on ground truth in a simulator. Unlike LVSPC, neither of these methods use a given model approximation nor do they reason about model uncertainty.

2.2.2 Using simplified models

Simplified models have been widely explored in the legged robotics literature, in particular using spring-mass damper models [37, 127]. Simplified models have been used to generate trajectories for a lower-level controller to track with guarantees [78]. However, these guarantees require access to a high-fidelity model. Other work [102] has used a set of simple models and a selection mechanism to choose between them. [103] use a given simplified dynamics model and learns a classifier on whether a given transition is reliable. We use GP uncertainty to model transition reliability rather than a classifier. We also use image observations and perform tracking concurrently.

2.2.3 Incorporating model uncertainty

Previous work has shown that reasoning about model uncertainty can improve data efficiency [31, 29]. PILCO [31] uses a Gaussian Process dynamics model for model uncertainty and achieves high data efficiency on learning control policies. Gaussian Processes dynamics have also been used for the purpose of both avoiding uncertainty [36], or explicitly seeking it [13]. PETS [29] uses a probabilistic ensemble of neural networks to model uncertainty and is able to outperform PILCO on control tasks with high state dimension. These methods have only been demonstrated on tasks for which state is available, and not on image domains where parameterizing uncertainty can be difficult. LVSPC aims to combine modeling of uncertainty in the dynamics with strong priors to maintain high data efficiency when learning from images.

2.3 Problem Statement

We consider a nonlinear discrete-time system with state $x \in \mathcal{X}$ and controls $u \in \mathcal{U}$. The system has unknown true dynamics given by $x_{t+1} = f(x_t, u_t)$. We assume $\mathcal{X}$ may be arbitrarily high-dimensional and unobserved. Instead we may only have access to observations $o \in \mathcal{O}$ via an observation function at the current state $o_t = g(x_t)$.

We define a trial as a time-limited attempt to find a sequence of controls $\{u_1, ..., u_T\}$ such that the final state $x_T \in \mathcal{X}_{goal}$ where $\mathcal{X}_{goal}$ is the trial's goal region. We assume that we can fully observe when the system has reached the goal i.e. $o \in \mathcal{O}_{goal} \iff x \in \mathcal{X}_{goal}$. The goal in observation space is defined as $\mathcal{O}_{goal} = \{g(x) : x \in \mathcal{X}_{goal}\}$. We assume that data collection on the true system is expensive. The unknown dynamics and high-dimensional state make this problem intractable to solve with a small dataset. Instead we seek to model the system in a latent state of lower dimensionality $z \in \mathcal{Z}$ with *simple* dynamics $\hat{f}_\rho$ parameterized by ρ with input-dependent noise. The transition distribution, which we will denote as p_z for shorthand is given by

$$p(z_{t+1}|z_t, u_t) = \mathcal{N}(\hat{f}_\rho(z_t, u_t), Q(z_t, u_t)) \tag{2.1}$$

We assume that $\hat{f}_\rho$ is given and is differentiable with respect to (z, u, ρ). Q is an input-dependent uncertainty term. We also assume that the simple dynamics are Markovian. The simple system has the same observation space $\mathcal{O}$ and has a given observation function $o_t = \hat{g}(z_t)$. We assume that we can *a priori* specify some subset of the goal region in $\mathcal{Z}$ as $\mathcal{Z}_{goal}$, i.e that $\{\hat{g}(z) : z \in \mathcal{Z}_{goal}\} \subset \mathcal{O}_{goal}$. This could also be done by specifying $\mathcal{O}_{goal}$ directly (as is common in learning to control from images, e.g. [35]) and using this to infer z_{goal}. We then seek to design a feedback policy $u_t = \pi(z_t)$ such that $z_T \in \mathcal{Z}_{goal}$ for some time T. Our goal is to design π using $\hat{f}_\rho$ so that it achieves high success rate after a small number of trials.

2.4 Methods

Our approach to this problem requires input in the form of a simple model approximation that is believed to accurately represent the dynamics over some subset of the complex system ($\mathcal{X}$,$\mathcal{U}$). By using this simple model in simulation we can generate large amounts of data. The key to our approach is to leverage this data and our knowledge of the simple system. We then reduce the problem of unsupervised representation and dynamics learning to that of supervised learning of a perception system for the simple model representation (offline), and then learning when this representation and the dynamics are accurate (online).

Our full method is shown in Algorithm 1 and Figure 2.2. The overall procedure is to first generate a dataset of images with corresponding simple model configurations and then to train a perception system to estimate these configurations from images. Once this perception system is trained offline, we move to the online execution/learning phase, where we must manipulate the never-before-seen complex system.

The goal of the online execution is to reach a given goal region. However, because the perception system and the simple model dynamics can only account for *some* complex model states, we must try to avoid states where the perception/dynamics are inaccurate. To this end, we collect data as we attempt the task and use that data to train a GP that captures the error in the simple model predictions. This

error distribution is input into a Kalman Filter variant to better estimate the state and into a trajectory optimizer, which attempts to avoid regions of state space where the simple model predictions are inaccurate. The process of planning trajectories, executing one action, estimating the resulting state, and replanning a trajectory (Alg. 2) repeats until the goal (or a timeout) is reached.

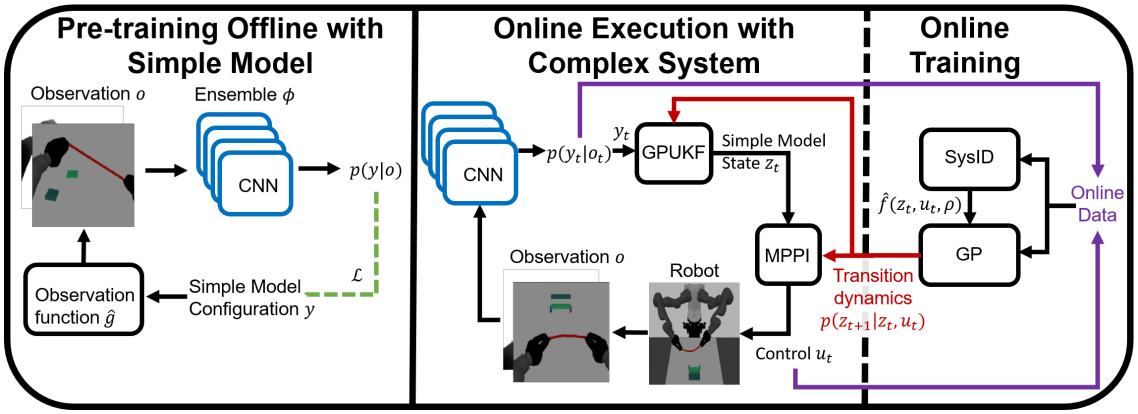


Figure 2.2: Method overview. *Left*: Training the CNN ensemble on image observations generated from the simple system offline. ϕ is a CNN ensemble with variance used as a measure of uncertainty; *Center*: Online execution using the simple model CNN with GPUKF filtering and MPPI for control; *Right*: Procedure for fitting parameterized simple model and GP from observations of the complex system. The transition probability (red) is trained to predict the future uncertainty of ϕ, allowing us to avoid avoid areas where ϕ is not confident.

2.4.1 Simple Model

The simple system state may contain elements which cannot be estimated from a single image, e.g. velocities. Thus we define the components of the simple state that can be noisily observed from a single image as latent observations y. We then have the non-linear discrete-time state space model with dynamics described in Eq. (2.1). In general there will be a non-linear mapping from z to y. In this chapter we consider only a linear mapping, which is sufficient for our models:

$$y_t = Cz_t + \epsilon, \tag{2.2}$$

Algorithm 1 LVSPC

Inputs: Simple model dynamics: $\hat{f}_\rho$; Simple model cost: c; Simple model renderer $\hat{g}$; Initial data size N; # Episodes K

Offline Training with simple system data

$\{y_i, o_i\}_{i=1}^N \leftarrow$ `CollectData`$(\hat{f}_\rho, \hat{g}, N)$;
$\phi \leftarrow$ `TrainStateEstimator`$(\{y_i, o_i\}_{i=1}^N)$;

Online Training with complex system data

$\mathcal{D} \leftarrow \emptyset$; $\rho, Q \leftarrow$ `Initialize`;
for k $\in \{1, ..., K\}$ **do**
  $p_z \leftarrow \mathcal{N}(\hat{f}_\rho(z_t, u_t), Q(z_t, u_t))$;
  $\mathcal{D} \leftarrow \mathcal{D} \cup$; `Rollout`$(p_z, c, \phi)$;
  $\rho \leftarrow$; `FitSimpleSystem`$(\mathcal{D}, \hat{f}_\rho)$;
  $Q \leftarrow$ `FitGP`$(\mathcal{D}, \hat{f}_\rho, Q, \rho)$;

For an n-dimensional simple model system ($z \in \mathbb{R}^n$) with m-dimensional ($m \leq n$) observations ($y \in \mathbb{R}^m$), $C = [I_{m \times m,}, 0_{m \times n-m}]$ selects the latent observations from z. For example, if z is the position and velocity of a point, then y is only the position, which is all that can be observed from a single image. In the case where $\epsilon \sim \mathcal{N}(0, R)$ for positive-definite R we can use noisy measurements y to estimate z by filtering using non-linear techniques such as the Unscented Kalman Filter (UKF) [159]. We will show how to use a GP to learn $Q(z_t, u_t)$ in Eq. (2.1) from data in Sec. 2.4.4.

2.4.2 Probabilistic CNN Ensemble for Perception

In order to use the simple model for the complex system, we need a perception system ϕ that maps images to simple model states (even if the image is generated from the complex system). We would also like a way to estimate how well a simple model state approximates the complex system at a given state, as this gives us an estimate of confidence in the simple system dynamics at this state. We use the uncertainty in the perception estimate as a proxy for correspondence between the

Algorithm 2 Rollout

Inputs: Transition distribution p_z; Simple model cost: c; CNN Ensemble ϕ

$\mathcal{D} \leftarrow \emptyset$; $\mu_1^z, \Sigma_1^z \leftarrow$ `Initialize`;
for t $\in \{1, ..., T\}$ **do**
  $\mu_t^y, \Sigma_t^y \leftarrow \phi(o_t)$;
  $y_t \sim \mathcal{N}(\mu_t^y, \Sigma_t^y)$;
  $\mu_t^z, \Sigma_t^z \leftarrow$`GPUKF`$(\mu_{t-1}^z, \Sigma_{t-1}^z, u_{t-1}, p_z, y_t)$;
  $u_t \leftarrow$ `MPPI`(μ_t^z, c, p_z);
  $\mathcal{D} \leftarrow \mathcal{D} \cup (\mu_t^y, \Sigma_t^y, u_t)$;
  `ExecuteAction(`u_t`)`;
  if `AtGoal` **then** break;
return $\mathcal{D}$

simple state and the unknown complex state. The perception output is

$$\mu_t^y, \Sigma_t^y = \phi(o_t) \tag{2.3}$$

$$y_t \sim p(y_t|o_t) = \mathcal{N}(\mu_t^y, \Sigma_t^y), \tag{2.4}$$

where the variance Σ_t^y estimates the uncertainty, and ϕ is the perception system. We assume an isotropic Gaussian in Eq. 2.3, thus Σ_t^y can be described by a vector $\sigma_t^y \in \mathbb{R}^m$. Ensembles have been empirically shown to give useful estimates of prediction uncertainty, which can be used to evaluate if a given input is out-of-distribution w.r.t the training data [82]. Thus using ensembles avoids manually defining a similarity between the complex system observations and observations generated from the simple system. Instead we can input observation o_t from the complex system into our perception system, and if it produces a high-certainty estimate of y_t (i.e. where $||\sigma_t^y||$ is small), this implies that y_t is a good approximation for the complex system at time t.

We parameterize ϕ as a CNN ensemble which is trained with data generated from the simple system. Each CNN in the ensemble is a probabilistic CNN which outputs the parameters of a Gaussian, these are then combined into one Gaussian estimate. We train the CNN via supervised learning on observations of the simple system which we collect from simulation, along with correspond simple system states. Importantly, we assume that we can generate observations from the simple system which are

similar to the complex system observations. To avoid requiring precise knowledge of the complex system before generating the simple model data, we generate a diverse training set of observations from the simple model. For example, we generate cart-poles with varying pendulum length for the tethered mass scenario. By generating diverse observations via domain randomization, our notion of visual similarity means that there is a simple system with some appearance and system parameters that looks similar to the complex system. See in Fig. 2.3 for examples.

Given an o_t of the complex system online, we sample y_t from the output of the ϕ and use this along with the learned GP transition distribution (Sec. 2.4.4) to track a Gaussian distribution over the simple model state ($p(z_t|u_{1:t-1}, y_{1:t}) = \mathcal{N}(\mu_t^z, \sigma_t^z)$) with a GPUKF [76]—an extension to the UKF for GP dynamics. When predicting $p(z_{t+1}|u_{1:t}, y_{1:t})$ in the GPUKF we use the posterior mean of the GP (Sec. 2.4.4) to perform the unscented transform, while the process noise is the posterior covariance of the GP, $Q(z_t, u_t)$, evaluated at (μ_t^z, u_t).

2.4.3 System Identification

The simple model dynamics may be parameterized by ρ (for example mass, length, etc.) and in order to use it, we must estimate the ρ which best approximates the complex system. One approach is using the Kalman filter to jointly estimate ρ and the latent state z, but we found that this was not numerically stable. Instead we use maximum-likelihood estimation on observed trajectories from the complex system.

Given an observed trajectory of the complex system consisting of $\{o_t, u_t\}_{t=1}^T$ we encode the observations into $\{\mu_t^y, \sigma_t^y, u_t\}_{t=1}^T$. Since our trajectory may contain transitions which the simple model cannot accurately predict, we split the trajectory into N trajectories of length $K < T$, and discard trajectories with average uncertainties above threshold α so we are left with high-certainty sub-trajectories. For each sub-trajectory we rollout the actions $u_{1:T}$ using Eq. (2.1) and (2.2) to get estimated observations $\hat{y}_{1:T}$ and perform gradient ascent on the parameters ρ and the trajectory initial states $\{z_1^i\}_{i=1}^N$ by maximizing the log likelihood of $\hat{y}_{1:T}$ in the distribution output by the CNN ensemble $\mathcal{N}(\mu_{1:T}^y, \sigma_{1:T}^y)$. The CNN weights are held constant. This

process optimizes ρ to match the observed dynamics for high-certainty transitions in $(\mathcal{Z},\mathcal{U})$.

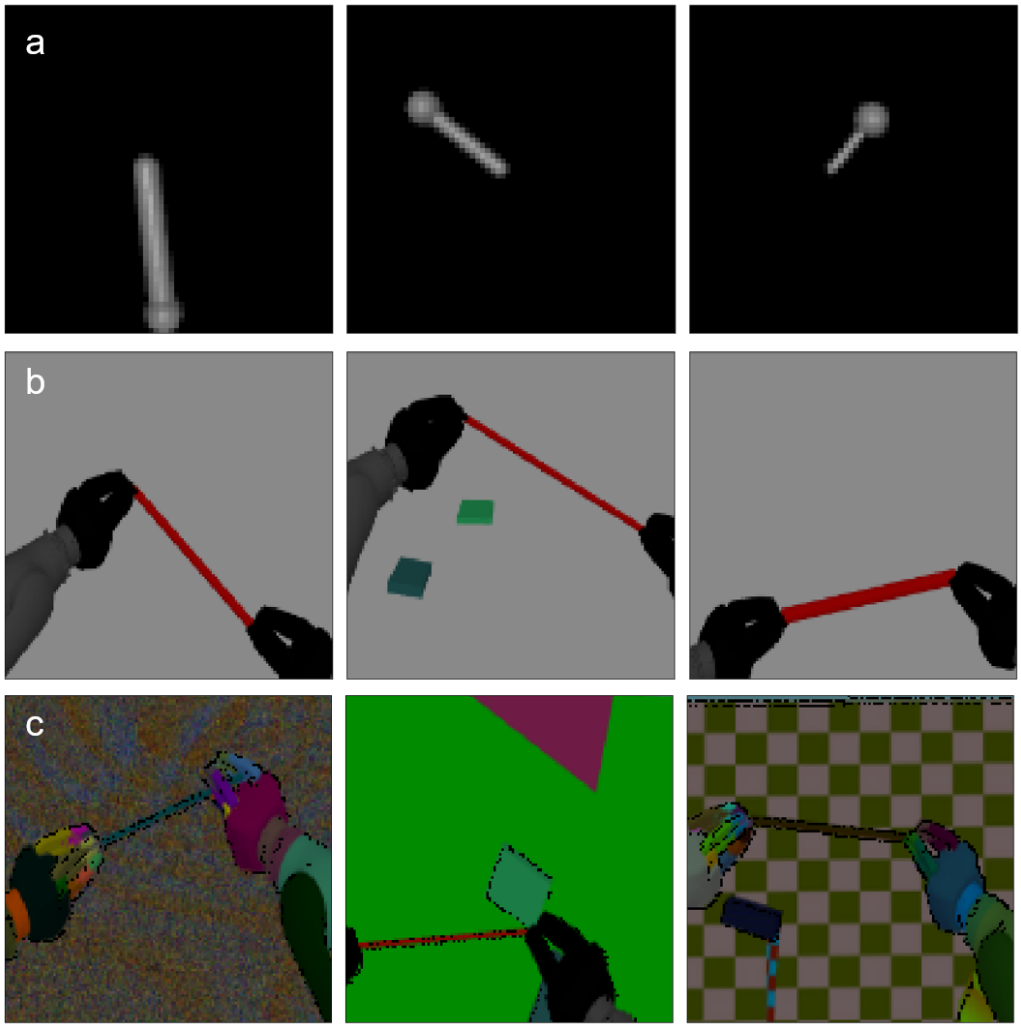


Figure 2.3: Examples of data generated from the simple system for training the CNN ensemble. (a) Tethered mass experiment, showing different geometries of the cart-pole. (b) Simulated rope manipulation experiment, showing different geometries of rigid link, and differing number and geometries of objects. (c) Real robot rope manipulation experiment. We randomize textures, lighting, obstacle configuration, camera pose, and rigid link geometry and add noise.

2.4.4 Predicting Future Uncertainty with GP Regression

From ϕ we have a confidence in our simple model approximation at a given y (the uncertainty σ^y). To keep the system in regimes where the approximation is accurate we also need to predict the future uncertainty conditioned on actions. Our uncertainty expresses uncertainty over the *validity* of the state as a description of the complex system, rather than the *value* of the state. Since we are using state uncertainty as a measure of confidence in the simple model approximation we model this uncertainty as state and action-dependent and use a GP with mean function $\hat{f}_\rho$ and kernel function $\mathcal{K}$ to model the transition distribution. The GP posterior is

$$p(z_{t+1}|z_t, u_t) = \mathcal{N}(\hat{f}_\rho(z_t, u_t, \rho) + \mu_f(z_t, u_t), Q(z_t, u_t)), \tag{2.5}$$

where μ_f and Q are typically found via conditioning on some training set. However in our case this is a Gaussian Process State Space Model (GPSSM) [43] with transition probability above and emission probability defined in Eq. (2.2). Training this GP is non-trivial as we do not have access to z directly. Instead we must jointly infer both the transition probability and z during training.

We use a Parametric Predictive GP (PPGP)[60] in order to train a GP with state-dependent aleatoric uncertainty via stochastic gradient descent. The uncertainty of the GP σ^z is used to predict the uncertainty of the CNN ensemble σ^y via Eq. (2.2). The PPGP is a sparse GP method which fits psuedo-inputs (ζ) and psuedo-ouputs ($\gamma \sim \mathcal{N}(m, S)$) such that conditioning the GP on (γ, ζ) approximates the true GP posterior. The GP parameters are thus (m, S, ζ) as well as the kernel hyper-parameters. The GP posterior contains an additional μ_f term compared with Eq. (2.1). This allows the GP posterior mean to deviate from that of the simple model, attempting to fit transitions which do not conform to the simple model dynamics. Since our representation is known to be insufficient to model the true dynamics of the system, we are conservative and do not allow the GP to fit such transitions by constraining $m = 0$ and thus $\mu_f = 0$. We compare to a variant of our method where we do not enforce $\mu_f = 0$ in our experiments.

We now describe how to train this GP using trajectories from the complex system

of the form $\{\mu_t^y, \sigma_t^y, u_t\}_{t=1}^T$. We would like Eq. (2.2, 2.5) and an initial $p(z_1)$ to be able to reproduce the trajectory and uncertainties from the CNN. The learning objective to be minimized is then

$$\mathcal{L} = \mathcal{KL}(p(y_{1:T}|o_{1:T})||p(y_{1:T}|u_{1:T})), \tag{2.6}$$

where $\mathcal{KL}$ is the Kullback–Leibler divergence, $p(y_{1:T})$ represents the joint distribution $p(y_1, ..., y_T)$, $p(y_{1:T}|o_{1:T})$ is the output of the CNN, and $p(y_{1:T}|u_{1:T})$ is the prediction from the dynamics and Eq. (2.2). The GP predicted uncertainty σ_t^z is used with Eq. (2.2) to predict a latent observation uncertainty $\hat{\sigma}_t^y$. This objective aims to make the predicted uncertainty $\hat{\sigma}_t^y$ and the observed uncertainties σ_t^y consistent, i.e. the GP will predict the future uncertainty.

$p(y_{1:T}|o_{1:T})$ is fixed (i.e. we are not retraining the CNN online). Given this, we can rewrite the objective in terms of expectations over $p(y_{1:T}|o_{1:T})$

$$\mathcal{L} = -\mathbb{E}_{p(y_{1:T}|o_{1:T})}\left[\log p(y_{1:T}|u_{1:T})\right] + \mathcal{H}\left[p(y_{1:T}|o_{1:T})\right], \tag{2.7}$$

where $\mathcal{H}$ is the entropy and this entropy term can be dropped as it only depends on the pre-trained CNN. We can then optimize by maximizing the conditional expectation in Eq. (2.7) of $y_{1:T}$. To do this we construct a variational lower bound on $p(y_{1:T}|u_{1:T})$. This lower bound is given by

$$\begin{aligned} ELBO = \sum_{t=1}^{T} \mathbb{E}_{q(z_t)}\left[\log p(y_t|z_t)\right] - \mathcal{KL}(q(z_1)||p(z_1)) - \\ \sum_{t=2}^{T} \mathbb{E}_{q(z_{t-1})}\left[\mathcal{KL}\left(q(z_t) \,||\, p(z_t|z_{t-1}, u_{t-1})\right)\right], \end{aligned} \tag{2.8}$$

where the prior on the initial state is $p(z_1) \sim \mathcal{N}(0, I)$ and $q(z_t) = p(z_t|y_{1:t}, u_{1:t-1})$ is the GPUKF filtering distribution [76]. The final objective to minimize is given by $\mathcal{L}_1$

$$\mathcal{L}_1 = -\mathbb{E}_{p(y_{1:T}|o_{1:T})}[ELBO] \geq \mathcal{L} \tag{2.9}$$

To evaluate this objective we use the reparameterization trick to sample from the CNN and estimate gradients for $\mathcal{L}_1$. After performing this training procedure we obtain the transition distribution p_z, which is used by the GPUKF to perform filtering and by the MPC to predict future uncertainty.

2.4.5 Model Predictive Control

For MPC we use MPPI [167] with a cost c for the given task. To encourage the controller to keep the system in the domain of the simple model we add a cost to penalize the predicted uncertainty. Thus the cost function has the form $c(z, \sigma^z, u)$ (examples are shown in the experiments). Note that typically in this setting the expected cost is computed, but as mentioned in the previous section, our uncertainty does not express uncertainty over the *value* of the state. When rolling out a predicted trajectory with the model we propagate the expectation through the dynamics and record the one-step uncertainty for each step resulting in a trajectory $(\mu_t^z, \sigma_t^z, u_t)_{t=1}^T$ with which to calculate the cost. If we do not penalize this uncertainty, it will be ignored, which is equivalent to assuming the simple model is always accurate (we compare to this method in our experiments). Also, because we manually design the simple model state representation, we can incorporate additional information, such as avoiding collision, into the cost, which would have to be learned for an unsupervised learned representation.

2.5 Results

We evaluate LVSPC on 1) manipulating a tethered mass, and 2) placing a rope in a narrow opening vs. baselines in the low-data regime. An episode is a time-limited attempt to reach the goal (terminating early when the goal is reached). See the accompanying video for example task executions.

2.5.1 Environments

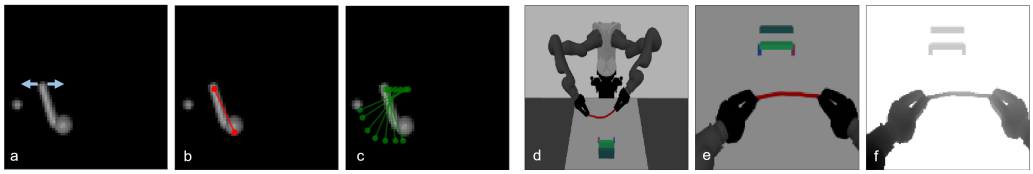


Figure 2.4: *(a)* Tethered mass input image (64x64 grayscale) with the target (left) and the single prismatic joint (blue); *(b)* output from CNN ensemble and GPUKF estimation (red); *(c)* planned trajectory from MPPI (green). Only the first action from this trajectory is executed before replanning; *(d)* The rope manipulation environment. The goal is to bring the centre of the rope to the centre of the narrow gap. The sides of the gap have protrusions which can catch the rope; *(e, f)* Example RGB and D observations from overhead Kinect.

Tethered Mass This task involves controlling a tethered mass by applying force to the base of the tether. The goal is to bring the mass to a target without the tether contacting the target (tether contact results in failure). We implement this system in MuJoCo [149]. There is a single actuated horizontal joint at the top of the tether (see Figure 2.4). Goals are randomly assigned at the start of each trial. This example demonstrates the applicability of LVSPC to highly-dynamic systems where velocity must be considered.

The simple system we choose here is the pendulum on a cart (i.e. a cart-pole); we choose this because we observed that when the tether is taut the system will behave like a pendulum. We use an analytical dynamics function for $\hat{f}_\rho$. We define $z = [p_{x_{cart}}, p_{x_{mass}}, p_{y_{mass}}, \dot{p}_{x_{cart}}, \dot{\theta}]$, where θ is the angle of the pendulum. We define the latent observations as $y = [p_{x_{cart}}, p_{x_{mass}}, p_{y_{mass}}]$ and thus $C = [I_{3\times3} \;\; 0_{3\times2}]$. The parameters ρ are [`mass_cart, mass_pole, angular_damping`].

Rope Manipulation This task consists of two KUKA iiwa 7-DOF arms holding the ends of a rope. The goal to bring the center of the rope to the center of a narrow gap between two obstacles. These obstacles have small protrusions on which the rope can become caught. We implement this environment in Gazebo with the ode45 back-end (Figure 2.4). The action space of the robot is $[\Delta p_L, \Delta p_R] \in \mathbb{R}^6$ where p_L, p_R

are the left and right end-effector positions, respectively. We use a Jacobian-based method for inverse kinematics so that transitions in the robot's configuration space are smooth. The observations consists of RGBD data from an overheard Kinect. The goal and obstacle configuration for the task remain fixed across trials, but the starting locations of the end-effectors vary. We choose this example because it mimics cable installation, which is necessary for manufacturing and repair applications, where there are often narrow gaps and protrusions.

The simple system we choose here is to treat the rope as if it is a rigid link. The simple dynamics are then specified by adding a constraint that the gripper distances remain fixed. This approximation will be accurate so long as the rope is kept taut for the duration of the task. We define $z = y = [p_L, p_R]$ and $C = [I_{6\times 6}]$. Since this model does not require dynamic parameters we forego the sysid step of our method.

2.5.2 Baselines

We compare LVSPC to two recent methods from the literature. The first method is PlaNet [49], a model-based reinforcement learning algorithm. PlaNet learns a low-dimensional state representation along with dynamics and cost functions. The second is CURL [86], which uses a contrastive loss to learn a representation in which to learn a policy and has shown state-of-the-art sample-efficiency. For each of these baselines we test them by training them directly on the task with the complex system. We also show results for when the baselines are pre-trained on the simple system and fine-tuned on the complex system to investigate if these methods can take advantage of the data from the simple system. Both baselines were originally proposed with RGB observations, and we extend them to use RGBD for the rope experiment.

We also test with three variants of LVSPC: 1) The full method which does both system identification and GP learning; 2) LVSPC without the GP, this is equivalent to only using the simple model for control, and assuming it will be sufficiently accurate for all transitions. We choose this variant to investigate whether learning and avoiding inaccurate areas of the simple model state space is helpful for task performance; and 3) LVSPC without constraining the GP posterior to be zero-mean, hence

attempting to learn a better approximation of the dynamics in the simple system state space, rather than only where the simple model is accurate.

2.5.3 Simple Model Data

Tethered Mass For pre-training the state estimator we generate 5000 trajectories of 20 time-steps from the cart-pole using random actions and render the cart-pole configurations to produce images. This corresponds to 100000 64×64 grayscale frames. For domain randomization, we vary the dimensions and parameters of the system (see Figure 2.3(a)).

Rope Manipulation For pre-training the state estimator we generate 800 trajectories of 50 time-steps length using random actions from the rigid body system and render the configuration. This corresponds to 80000 $128 \times 128 \times 4$ RGBD frames. For domain randomization, we vary the dimensions of the rigid link and the obstacles, as well as the obstacle locations (examples shown in Figure 2.3(b)).

2.5.4 Cost Functions

For both LVSPC and PlaNet we use an MPC horizon of 40 and sample 1000 trajectories per timestep. We do not have a cost on control. CURL and PlaNet use the true environmental cost i.e. $c_{env}(x_t)$, whereas LVSPC and variants use an equivalent cost based on the simple model state with an uncertainty penalty $c(z_t, \sigma_t^z)$. The environmental costs use the true state from the simulator to calculate the cost (because CURL and PlaNet have no knowledge of the simple model), whereas LVSPC uses the simple model state to approximate this cost, effectively giving CURL and PlaNet an advantage.

Tethered Mass The environmental cost consists of three parts; a euclidean distance to goal, a collision penalty for the tether and mass, and a penalty when the system goes out of view of the camera. The cost functions are $c(z_t, \sigma_t^z) = \delta_g \texttt{distToGoal}_{\mathcal{Z}} + \texttt{OffScreen}(z_t) + 10\texttt{checkCollision}(z_t) + \beta\sigma_t^z$ and $c_{env}(x_t) =$

$\delta_g \texttt{distToGoal}_{\mathcal{X}} + \texttt{OffScreen}(x_t) + 10\texttt{checkCollision}(x_t)$, where β is a parameter on how heavily to weigh uncertainty, and δ_g is 0 if the goal is reached before time t and 1 otherwise. To balance exploiting vs. exploring we increase β from 0 to 2.0 in the first 10 episodes. This cost is not memoryless; δ_g depends on the state for times $t' < t$. This is because we only wish to hit the target, we do not have to reach the target and stay there.

Rope Manipulation The environmental cost is the distance to the goal, computed by considering the centre of the rope to be a floating point, discretizing the 3D environment into a 8-connected graph and solve for the shortest path to the goal for every point in the graph. We do not penalize contact for the baselines, as we found that they could exploit contact to help complete the task. The cost for LVSPC penalizes contact (because the simple model is rigid), where we do a collision-check for the rigid-link approximation. The cost functions are $c(z_t, \sigma_t^z) = \texttt{distToGoal}_{\mathcal{Z}} + \beta\sigma_t^z + 100\texttt{checkCollision}(z_t)$ and $c_{\text{env}}(x_t) = \texttt{distToGoal}_{\mathcal{X}}$.

To balance exploiting vs. exploring we increase β from 0 to 1000 in the first 10 episodes.

2.5.5 Network Architectures

Networks are implemented in PyTorch [115], and the GPs are implemented in GPytorch [44], which allows us to exploit parallelism on the GPU for GP inference when performing MPPI. Thus, for the rope manipulation experiment, an iteration of MPPI takes only 0.89s on average using an Intel i7-8700K CPU and an Nvidia 1080Ti GPU. For both experiments we use a CNN ensemble consisting of 10 networks. All convolutional filters have filter size 3×3 and stride 2 for downsampling, all layers other than the output layers use ReLU activations. We use the Adam optimizer with a learning rate of 10^{-3}, except when fine-tuning the pretrained CURL and PlaNet models where we use 10^{-4}.

For the GP dynamics model, we use 200 inducing points. We train an independent GP for each output dimension using the RBF kernel with automatic-relevance determination [110]. We use a learning rate of 10^{-2} to train the GP and perform

sysid. For each experiment CURL and PlaNet use encoders with the same architecture as our CNN. The transition and reward models for PlaNet are the same architecture as [49]. The actor-critic architecture for CURL is the same as in [86]. Both CURL and PlaNet are trained end-to-end.

Tethered Mass Each CNN consists first of 4 convolutional layers. There is then a fully connected layer with 2048 hidden units, followed by an output layer.

Rope manipulation Each CNN seperately processes depth and RGB, consisting of an RGB module and a depth module which are combined downstream. Each module consists first of 4 convolutional layers. There is then a fully-connected layer with 512 hidden units. After passing the RGB image through both the RGB module, and the depth image through the depth module, the output from each module is combined and passed through a final hidden layer of 1024 units, followed by an output layer.

2.5.6 Results

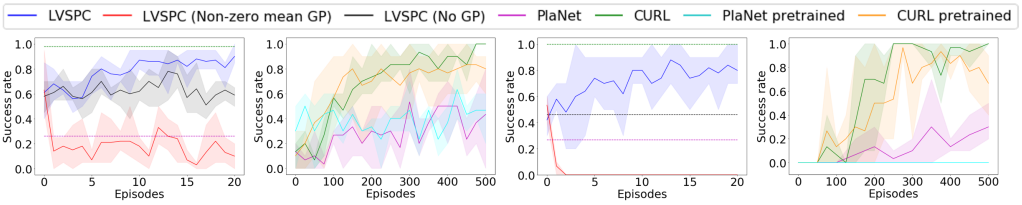


Figure 2.5: Average Success over 10 test tasks vs number of episodes for both experiments. Shaded region shows minimum and maximum success rate over 5 runs for LVSPC and ablations and 3 runs for the baselines for a total of 50 and 30 test tasks for LVSPC and the baselines, respectively. *a)* LVSPC and ablations for tethered mass, dotted lines show baseline performance after 500 episodes. *b)* Baselines for tethered mass. *c)* LVSPC and ablations for rope, dotted lines show baseline performance after 500 episodes. *d)* Baselines for rope.

Tethered Mass An example of the system tracking and MPC is demonstrated in Figure 2.4. Our statistical results are shown in Figure 2.5(a, b). PlaNet achieves it's

maximum performance at 200-300 episodes and has a success rate of approximately 26% with large variation. We see that CURL shows the highest asymptotic performance, with 97% after 400 episodes. Higher asymptotic performance is typical of model-free learning methods. Pre-training both PlaNet and CURL on data from the simple system results in improved initial performance, but lower final performance. In contrast, LVSPC achieves approximately 90% after 20 episodes, outperforming PlaNet and matching CURL's performance after 200 episodes, demonstrating 10x improved data efficiency. We also see that seeking to learn the dynamics in the simple state space with the GP results in substantially worse performance. This is likely because the simple state representation is insufficient to model the full complex dynamics.

Simulated rope manipulation Our statistical results are shown in Figure 2.5(c, d). PlaNet's performance after 500 episodes is approximately 30%, while CURL solves the task with almost 100% success rate after 250 episodes. Pre-training CURL on data from the simple system results in improved initial performance, but lower final performance, however pretraining PlaNet led to poor performance which it could not recover from, getting caught on the obstacles in every episode. Our full method achieves 80% success rate after 20 episodes, again equivalent to CURL's performance after 200 episodes (thus we have 10x better data-efficiency) and outperforming PlaNet's final performance. We see that naively treating the rope as a rigid object results in approximately 46% success and almost all failures result from the rope snagging on the protrusions on the side of the gap. As in the tethered mass experiment, attempting to fit the complex dynamics in the simple mode space is ineffective, causing frequent snagging on obstacles.

2.5.7 Rope Manipulation on a Real Robot

Our simulation experiments show that LVSPC is effective at transferring within the same simulation environment. To validate that we can still use LVSPC when the simple model and complex environments are very different, we perform the rope manipulation experiment described above on a real robot using a perception system

Episode	0	5	10	15	20
Success rate	0.3	0.7	0.8	0.78	0.82

Table 2.1: Results over 5 random seeds for real robot experiment

trained only in simulation. We use domain randomization to improve the transfer of the CNN ensemble to real data [146] (see Figure 2.3(c)). We observed better generalization when we randomized the pose of the camera and trained the CNN ensemble to produce an estimate in the camera frame instead of the world frame.

We perform the experiment on the real robot over 5 random seeds. For each seed, after every 5 episodes we record the success rate on 10 test episodes. The results are shown in Table 2.1. Using LVSPC we can complete this task with 80% success using only 10 episodes of data collected on the real robot. This experiment demonstrates that using LVSPC is promising for real-world tasks, as we only need data from simulation to train an effective perception system.

2.6 Conclusion

We have presented LVSPC for leveraging a given simple model approximation to improve data efficiency for control tasks on systems with complex dynamics from image observations. We demonstrated this method on two tasks, showing substantially improved performance in the low-data regime over recent reinforcement learning methods. We have also demonstrated that we can apply our framework to a real robot while only using simulated data for pre-training. We assumed that the user specifies a type of simple model, but choosing a simple model which can approximate the complex system is an open problem, made difficult by the requirement that it must be possible to complete the task while operating only in the regime where the simple model is accurate. In future work we intend to incorporate multiple simple models and create a way to decide which is most appropriate.

CHAPTER III

Learning a Generalizable Trajectory Sampling Distribution for Model Predictive Control

In this chapter we propose a sample-based Model Predictive Control (MPC) method for collision-free navigation that uses a normalizing flow as a sampling distribution, conditioned on the start, goal and environment. This representation allows us to learn a distribution that accounts for both the dynamics of the robot and complex obstacle geometries. We propose a way to incorporate this sampling distribution into two sampling-based MPC methods, MPPI and iCEM. However, when deploying these methods, the robot may encounter an out-of-distribution (OOD) environment. To generalize our method to OOD environments we also present an approach that performs *projection* on the representation of the environment. This projection changes the environment representation to be more in-distribution while also optimizing trajectory quality in the true environment. Our simulation results on a 2D double-integrator, a 12DoF quadrotor and a 7DoF kinematic manipulator suggest that using a learned sampling distribution with projection outperforms MPC baselines on both in-distribution and OOD environments, including OOD environments generated from real-world data. [1]

[1]The work in this chapter was published in [123] and [125].

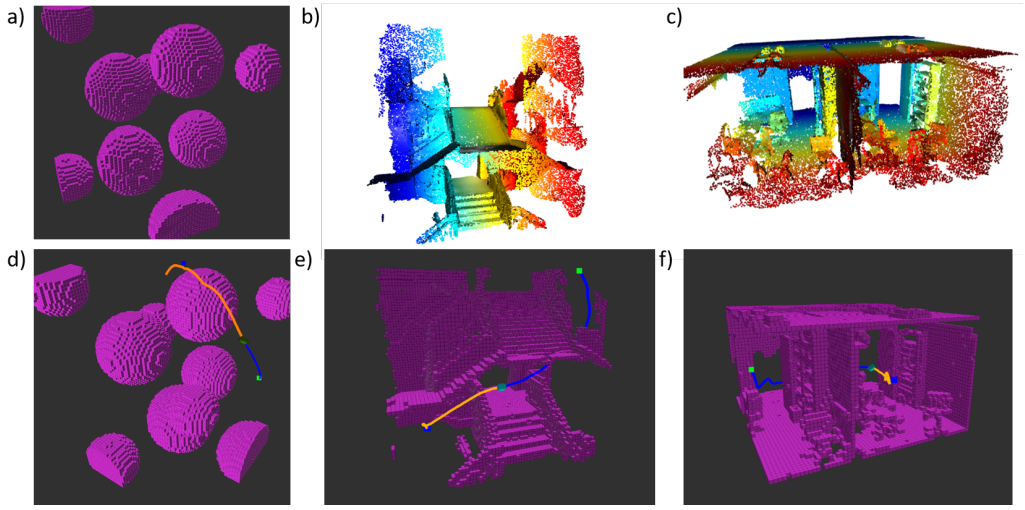


Figure 3.1: a) The training environment for our learned control sequence posterior. b,c) Point clouds of two real-world environments taken from the 2D-3D-S dataset [8]. d) One of our proposed methods, FlowMPPIProject, controlling a dynamic quadrotor in the training environment. e,f) FlowMPPIProject controlling a dynamic quadrotor to successfully traverse the two real-world environments. The executed trajectory is shown in blue, and the planned trajectory is shown in orange at an intermediate point in the execution.

3.1 Introduction

Model predictive control (MPC) methods have been widely used in robotics for applications such as autonomous driving [166, 167], bipedal locomotion [18] and manipulation of deformable objects [122]. For nonlinear systems, sampling-based approaches for MPC such as the Cross Entropy Method (CEM) and Model Predictive Path Integral Control (MPPI) [77, 167] have proven popular due to their ability to handle uncertainty, their minimal assumptions on the dynamics and cost function, and their parallelizable sampling. However, these methods struggle when randomly sampling low-cost control sequences is unlikely and can become stuck in local minima, for example when a robot must find a path through a cluttered environment. This problem arises because the sampling distributions used by these methods are not informed by the geometry of the environment.

Previous work has investigated the duality between control and inference [148,

144] and considered both planning and control as inference problems [9, 151, 131]. Several recent papers have considered the finite-horizon stochastic optimal control problem as Bayesian inference, and proposed methods of performing variational inference to approximate the distribution used to sample control sequences [84, 162, 112, 12]. To perform variational inference, we must specify a parameterized distribution that is tractable to optimize and sample while also being flexible enough to provide a good approximation of the true distribution over low-cost trajectories, which may exhibit strong environment dependencies and multimodalities. While more complex representations have been used to represent this distribution [84, 112], these distributions are initially uninformed and must be iteratively improved during deployment. In this paper, we present a method that uses a Normalizing Flow to represent this distribution and we learn the parameters for this model from data. The advantage of this approach is that it will learn to sample control sequences which are likely to be both goal-directed and collision-free (i.e. low-cost) for the given system. We also demonstrate how this learned distribution can be integrated with two sample-based MPC algorithms, iCEM [119] and MPPI [167].

However, as is common in machine learning, a learned model cannot be expected to produce reliable results when its input is radically different from the training data. Because the space of possible environments is very high-dimensional, we cannot hope to generate enough training data to cover the set of possible environments a robot could encounter. This problem compounds when we generate training data in simulation, but the method must be deployed in the real-world (i.e. the sim2real problem). Thus, when deploying this method, the robot may encounter an out-of-distribution (OOD) environment, i.e. one which is radically different from those used in training. In such cases, the learned distribution is unlikely to produce low-cost control sequences.

To generalize the learned distribution to OOD environments we propose performing a *projection* on the representation of the environment as part of the MPC process. This projection changes the environment representation to be more in-distribution while also optimizing trajectory quality in the true environment. In essence, this method "hallucinates" an environment that is more familiar to the normalizing flow

so that the flow produces reliable results. However, the key insight behind our projection method is that the "hallucinated" environment cannot be arbitrary, it should be constrained to preserve important features of the true environment for the MPC problem at hand. For example, consider a navigation problem for a 2D point robot. If the normalizing flow is trained only on environments consisting of disc-shaped obstacles, an environment with a corridor would be OOD and the flow would be unlikely to produce low-cost trajectories. However, if we morph the environment to approximate the corridor near the robot with disc-shaped obstacles (producing an in-distribution environment), the flow will then produce low-cost samples for MPC.

In this chapter we extend our previous conference paper [123] on this topic by incorporating the learned sampling distribution with another sampling-based MPC algorithm, iCEM [119]. We show that we can use the same learned sampling distribution with either iCEM or MPPI without retraining, which speaks to the generality of our method. In addition, we extend our method by learning a sampling distribution over a set of parameterized cost functions and show that we can achieve high performance across different parameter settings. Further, for the experiments where smooth control sequences are important, we have removed the addition of noise to the control sequence samples during training. This noise, while aiding exploration, resulted in noisy sampled control sequences. We also extend our methods to motion planning problems for manipulators and present new experiments on a 7DoF manipulator in several simulated and one real environment. Finally, we expand the discussion of related work and add a Discussion section that presents the limitations of our method.

Our simulation results on a 2D double-integrator, a 3D 12DoF underactuated quadrotor, and a 7DoF manipulator suggest that our flow-based MPC methods with projection outperform state-of-the-art MPC baselines on both in-distribution and OOD environments, including OOD environments generated from real-world data (Figure 3.1). In addition we validate our methods on a 7DoF manipulator in the real world.

3.2 Related Work

3.2.1 Planning & Control as Inference

The connection between control and inference was established many years ago, with Kalman first establishing the duality between inference in linear Gaussian systems and the Linear Quadratic Regulator (LQR) [65]. This duality was later extended further by Todorov to non-linear and deterministic systems subject to some restrictions, such as quadratic control cost and control-affine dynamics [148].

Early work by Attias framed planning for discrete state and action spaces as an inference problem over a a Hidden Markov Model and proposed a message-passing algorithm for planning [9]. Since then, many message-passing methods have been proposed for planning in continuous action spaces [151], for solving Stochastic Optimal Control (SOC) problems [151, 131, 164] and for policy learning [132]. For situations in which exact inference is intractable, such as for non-linear dynamics models, these methods perform approximate inference by using linearized Gaussian messages.

Another body of work has exploited the relationship between SOC problems and inference in diffusion processes [42], leading to the Path-Integral approach for control [67, 147, 68, 145, 144], which solves the SOC problem by performing approximate inference with Monte-Carlo expectations over trajectories. Model-predictive Path Integral (MPPI) [166, 167] control is one of these algorithms and has enjoyed widespread use for robot control problems. Another widely-used sampling-based MPC algorithm is the Cross Entropy Method (CEM) [77], with a recent variant, improved-CEM (iCEM) [119], showing state-of-the-art performance on several benchmarks. Recently, Watson and Peters proposed using a Gaussian Process to represent the sampling distribution of control sequences [163], resulting in much smoother sampled control sequences. However, the resulting sampling distribution is still Gaussian, and thus cannot capture multi-modal trajectories.

Recently, several approaches to control-as-inference have been developed that rely on Variational Inference (VI) to perform approximate inference [84, 162, 112, 12]. Variational inference techniques rely on approximating the distribution of interest

with a simpler, parameterized distribution. Inference is then performed by optimizing the parameters of this distribution (the *variational posterior*) to minimize the Kullback-Leibler divergence between the approximation and the desired distribution [16]. The choice of parameterized distribution is thus very important for the tractability of the approximate inference procedure and the quality of the approximation. VI methods often use an independent Gaussian posterior, known as the mean-field approximation [16]. Okada and Taniguchi represent the variational posterior as a Gaussian mixture [112], and show how this posterior can be used with both MPPI and CEM. Lambert et al. propose using a particle representation [84]. This method uses Stein Variational Gradient Descent (SVGD) [96], which performs gradient descent on the particles to maximize their posterior likelihood while also ensuring particle diversity, where diversity is determined by the choice of an appropriate kernel function. The authors use a Monte-Carlo estimate of the posterior gradient for non-differentiable costs and dynamics. This method has also been extended to handle parameter uncertainty [12]. These representations allow for greater flexibility in representing complex and multi-modal posteriors. We will similarly use a flexible class of distributions to represent the posterior, but will further make the posterior dependent on the start, goal, and environment. To our knowledge our approach is the first to amortize the cost of computing this posterior by learning a conditional control sequence posterior from a dataset.

3.2.2 Learning sampling distributions for planning

Our work is related to work on learning sampling distributions from data for motion planning. Previous work [178] has proposed learning a rejection sampling policy via Reinforcement Learning (RL). This policy is learned across multiple different environments, but does not take any environment information as input and thus its output is independent of the environment. Others have proposed learning a sampling distribution which is dependent on the environment, start and goal [57, 128]. These methods were restricted to geometric planning, but recent work [89] proposed an approach for kinodynamic planning which learns a generator and discriminator which

are used to sample states that are consistent with the dynamics. Recent work by Lai et al. [81] uses a diffeomorphism to learn the sampling distribution; a model that is similar to a normalizing flow. The model we propose will also learn to generate samples conditioned on the start, goal and environment, though in this work we are considering online MPC and not offline planning.

Loew et al. [98] uses probabilistic movement primitives (ProMPs) learned from data as the sampling distribution for sample-based trajectory optimization, however the representation of these ProMPs only allows for uni-modal distributions and the sampling distribution is not dependent on the environment.

Adaptive and learned importance samplers have been used for path integral controllers [66, 21]. These methods learn a feedback policy. Sampling control sequences then consists of sampling perturbations to the output of a feedback policy rather than open-loop controls, thus modifying the trajectory sampling distribution. These methods only consider a single control problem and the learned samplers do not generalize to different goals and environments.

Parallel work by Sacks and Boots has also proposed using a Normalizing Flow to learn the sampling distribution for MPC [137]. Their approach uses a bi-level optimization to learn the sampling distribution and demonstrates impressive results in the low-sample regime. However, the resulting sampling distribution is specific to a given MPC controller, and they do not train over multiple environments. In contrast, we demonstrate that our learned sampling distribution can be used with different sample-based MPC controllers without retraining, train over multiple environments, and adapt to novel environments.

3.3 Preliminaries

3.3.1 Variational Inference for Stochastic Optimal Control

We can reformulate SOC as an inference problem (as in [131, 150, 112, 84]). First, we introduce a binary 'optimality' random variable o for a trajectory such that

$$p(o = 1|\tau) \propto \exp\left(-c(\tau)\right). \tag{3.1}$$

We place a prior $p(U)$ on U, resulting in a prior on τ, $p(\tau) = p(X|U)p(U)$ and aim to find posterior distribution $p(\tau|o = 1) \propto p(o = 1|\tau)p(\tau)$. In general, this posterior is intractable, so we use variational inference to approximate it with a tractable distribution $q(\tau)$ which minimizes the KL-divergence $\mathcal{KL}(q(\tau)||p(\tau|o = 1))$ [16]. Since we define the trajectory by selecting the controls, the variational posterior factorizes as $p(X|U)q(U)$. Thus, we must compute an approximate posterior over control sequences. The quantity to be minimized is

$$\begin{aligned} \mathcal{KL}\left(q(\tau)||p(\tau|o = 1)\right) &= \int q(\tau) \log \frac{q(\tau)}{p(\tau|o = 1)} d\tau \\ &= \int q(X, U) \log \frac{p(X|U)q(U)p(o = 1)}{p(o = 1|X, U)p(X|U)p(U)} dX dU. \end{aligned} \tag{3.2}$$

Since $p(o = 1)$ on the numerator does not depend on U, when we minimize the above divergence it can be dropped. The result is minimizing the below quantity, the *variational free energy* $\mathcal{F}$

$$\mathcal{F} = \int q(X, U) \log \frac{q(U)}{p(o = 1|X, U)p(U)} dX dU \tag{3.3}$$

$$= -\mathbb{E}_{q(\tau)}[\log p(o|\tau) + \log p(U)] - \mathcal{H}(q(U)) \tag{3.4}$$

$$= \mathbb{E}_{q(\tau)}[\log \hat{J}(X, U)] - \mathcal{H}(q(U)), \tag{3.5}$$

where $\mathcal{H}(q(U)$ is the entropy of $q(U)$. For the last expressions we have used our formulation that the $p(o = 1|X, U) = \exp(-J(X, U))$ and we have incorporated the deviation from the prior into a modified cost function $\hat{J}$. For example, a zero-mean Gaussian prior on the controls can be equivalently expressed as a squared cost on the magnitude of the controls.

Intuitively, we can understand that the first term promotes low-cost trajectories, the second is a regularization on the control, and the entropy term prevents the variational posterior collapsing to a *maximum a posteriori* (MAP) solution.

3.3.2 Variational Inference with Normalizing flows

Normalizing flows are bijective transformations that can be used to transform a random variable from some base distribution (i.e. a Gaussian) to a more complex distribution [133, 33, 73]. Consider a random variable $z \in \mathbb{R}^d$ and with known pdf $p(z)$. Let us define a bijective function $f : \mathbb{R}^d \to \mathbb{R}^d$ and a random variable y such that $y = f(z)$ and $z = f^{-1}(y)$. According to the change of variable formula, we can define $p(y)$ in terms of $p(z)$ as follows:

$$p(x) = p(z) \left| \det \frac{\partial f}{\partial z} \right|^{-1} \tag{3.6}$$

$$\log p(y) = \log p(z) - \log \left| \det \frac{\partial f}{\partial z} \right|. \tag{3.7}$$

Normalizing flows can be used as a parameterization of the variational posterior [133]. By selecting a base PDF $p(z)$ and a family of parameterized functions f_θ, we specify a potentially complex set of possible densities $q_\theta(y)$. Suppose that we want to approximate some distribution $p(y)$ with some distribution $q_\theta(y)$. The variational objective is to minimize $\mathcal{KL}(q_\theta(y)||p(y))$. This is equivalent to

$$\begin{aligned} \mathcal{KL}\left(q_\theta(y)||p(y)\right) &= \int q_\theta(y) \log \frac{q_\theta(y)}{p(y)} dx \\ &= \mathbb{E}_{q_\theta(y)}[\log q_\theta(y) - \log p(y)] \\ &= \mathbb{E}_{p(z)} \left[\log p(z) - \log \left| \det \frac{\partial f_\theta}{\partial z} \right| - \log p(y) \right]. \end{aligned} \tag{3.8}$$

Thus we can optimize the parameters θ of the bijective transform f_θ to minimize the variational objective. We will use a normalizing flow to represent the control sequence posterior in our method.

3.4 Problem Statement

This chapter focuses on the problem of Finite-horizon Stochastic Optimal Control. We consider a discrete-time system with state $x \in \mathbb{R}^{d_x}$ and control $u \in \mathbb{R}^{d_u}$ and

known transition probability $p(x_{t+1}|x_t, u_t)$. We define finite horizon trajectories with horizon T as $\tau = (X, U)$, where $X = \{x_0, x_1, ...x_T\}$ and $U = \{u_0, u_1, ...u_{T-1}\}$.

Given an initial state x_0, a goal state x_G, and a signed-distance field (SDF) of the the environment E, our objective is to find U which minimizes the expected cost $E_{p(X|U)}[c(\tau)]$ for a given cost function c, where $p(X|U) = \prod_{t=0}^{T-1} p(x_{t+1}|x_t, u_t)$. Note that we will use c to mean both the cost on the total trajectory $c(\tau)$ and the cost of an individual state action pair $c(x, u)$. This chapter focuses on the problem of collision-free navigation, where c is parameterized by (x_G, E, ρ), where ρ is a set of parameters specifying the cost. In our experiments, ρ consists of parameters penalizing the magnitude of the controls, non-smooth controls, and where appropriate, velocities.

This problem is difficult to solve in the general case because the mapping from environments to collision-free U can be very complex and depends on the dynamics of the system. To aid in finding U, we assume access to a dataset $\mathcal{D} = \{E, x_0, x_G, \rho\}^N$, which will be used to train our method for a given system. We will evaluate our method in terms of its ability to reach the goal without colliding and the cost of the executed trajectory. Moreover, we wish to solve this problem very quickly (i.e. inside a control loop), which limits the amount of computation that can be used.

Our proposed architecture for learning an MPC sampling distribution is shown in Figure 3.2. In this section we first introduce how we represent and learn the control sequence posterior as a Normalizing Flow, and train over a dataset consisting of starts, goals, cost function parameters, and environments to produce a sampling distribution for control sequences. Next, we show how this sampling distribution can be used to improve two different sampling-based MPC methods, MPPI and iCEM. Finally, we describe an approach for adapting the learned sampling distribution to novel environments which are outside the training distribution.

3.4.1 Overview of Learning the Control Sequence Posterior

The control sequence posterior introduced in section 3.3.1 is specific to each MPC problem. Our approach is to use dataset $\mathcal{D}$ to learn a conditional control sequence posterior $q(U|x_0, x_G, \rho, E)$. We will use a Conditional Normalizing Flow (CNF) [168]

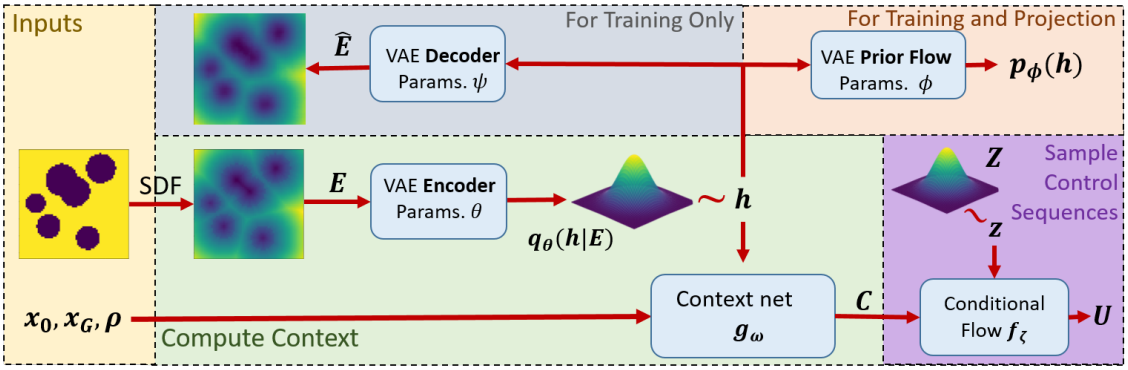


Figure 3.2: The architecture of our method for sampling control sequences. We take as input initial and goal states x_0, x_G, and the environment, converted to a signed distance field E. E is input into a VAE to produce a latent distribution $q_\theta(h|E)$, which we sample to get the environment embedding h. This h is used, along with x_0, x_G and ρ as input to the network g_ω to produce a context vector C. C, along with a sample from a Gaussian distribution Z, is input into the conditional normalizing flow f_ζ to produce a control sequence U. During training only, we use a decoder to reconstruct the SDF from h as part of the loss. We also use a normalizing flow prior for the VAE to compute an OOD score for a given h, which is necessary to perform projection.

to represent this conditional posterior. We use a CNF as this allows us tractably perform exact likelihood calculations and generate samples. The CNF takes the form of $q_\zeta(U|C)$, where C is the context vector which we compute as follows: First, we input E into the encoder of a Variational Autoencoder (VAE) [74] to produce a distribution over environment embeddings h. We then sample from this distribution to produce an h. A neural network g_ω then produces C from (x_0, x_G, ρ, h) (Figure 3.2). Essentially C is a representation of what is important about the start, goal, cost parameters, and environment for generating low-cost trajectories.

The above models are trained on the dataset $\mathcal{D}$, which consists of randomly sampled starts, goals and simulated environments. To train the system we iteratively generate samples from the control sequence posterior, weigh them by their cost, and perform a gradient step on the parameters of our models to maximize the likelihood of low-cost trajectories.

At inference time, we simply compute C and generate control sequence samples from $q_\zeta(U|C)$. Below we describe each component of the method to learn $q_\zeta(U|C)$ in detail.

3.4.2 Representing the start, goal and environment as C

As discussed, our dataset $\mathcal{D}$ consists of environments, starts and goals. The details of the dataset generation for each task can be found in section 3.5.1. Since the environment is a high dimensional SDF, we must first compress it to make it computationally tractable to train the control sequence posterior. To encode the environment, we use a VAE with environment embedding h. The VAE consists of an encoder ϕ, which is a Convolutional Neural Network (CNN) that outputs the parameters of a Gaussian. The decoder is a transposed CNN which produces the reconstructed SDF $\hat{E}$ from h. The decoder log-likelihood $p_\psi(E|h)$ is $||\hat{E} - E||_2$, where ψ are the parameters of the decoder CNN. [24] showed that learning a latent prior can improve VAE performance, so we parameterize the latent prior $p_\phi(h)$ as a

normalizing flow and learn the prior during training. The loss for the VAE is

$$\begin{aligned}\mathcal{L}_{VAE} &= \mathbb{E}_{\phi}\left[-\log p_{\psi}(E|h)\right] + \mathcal{KL}(\phi||p_{\phi}(h)) \\ &= \mathbb{E}_{\phi}\left[-\log p_{\psi}(E|h) + \log\phi - \log p_{\phi}(h))\right].\end{aligned} \tag{3.9}$$

We then use a Multilayer Perceptron (MLP) network g_{ω} to generate a context vector C to use in the normalizing flow, via $C = g_{\omega}(x_0, x_G, \rho, h)$, which has parameters ω.

3.4.3 Learning $q_{\zeta}(U|C)$

We use a conditional normalizing flow parameterized by ζ to define the conditional variational posterior, i.e. $q_{\zeta}(U|C)$ is defined by $U = f_{\zeta}(Z, C)$ for $Z \sim p(Z) = \mathcal{N}(0, I)$. The variational free energy (Equation (3.4)) then becomes

$$\begin{aligned}\mathcal{F} = &-\mathbb{E}_{q(\tau)}\left[\log p(o|\tau)\right] + \\ &\mathbb{E}_{p(Z)}\left[\log p(Z) - \log\left|\det\frac{\partial f_{\zeta}(Z, C)}{\partial Z}\right|\right].\end{aligned} \tag{3.10}$$

We can then optimize ζ to minimize the free energy.

By using a conditional normalizing flow, we are amortizing the cost of computing the posterior across environments. The conditional normalizing flow $U = f_{\zeta}(Z, C)$ is invertible with respect to Z, i.e. $Z = f^{-1}(U, C)$. For our conditional Normalizing Flow we use an architecture based on Real-NVP [33] architecture with conditional coupling layers [168], the structure is specified in section 3.5.3. Since U is a control sequence, the proposed normalizing flow is a joint distribution over the entire control sequence, meaning that the control sequence posterior is able to represent dependencies across time.

Minimizing eq. (3.10) via gradient descent requires the cost and dynamics to be differentiable. To avoid this, we estimate gradients, using the method in [112]: At each iteration, we sample R control sequences $U_{1..R}$ from $q_{\zeta}(U|C)$ and compute weights

$$w_i = \frac{q_{\zeta}(U_i|C)^{-\beta} p(o|\tau_i)^{\frac{1}{\alpha}}}{\frac{1}{R}\sum_{j=1}^{R} q_{\zeta}(U_j|C)^{-\beta} p(o|\tau_j)^{\frac{1}{\alpha}}}, \tag{3.11}$$

where $p(o|\tau) = \exp(-c(\tau))$. These weights represent a trade-off between low-cost and high entropy control sequences controlled by hyperparameters α and β. The weights and particles $\{U_{1..R}, w_{1..R}\}$ effectively approximate a posterior which is closer to the optimal $q(U|C)$. At each iteration of training, we take one gradient step to maximize the likelihood of $U_{1..R}$ weighted by $w_{1..R}$, then resample a new set $U_{1..R}$. The flow training loss for this iteration is

$$\mathcal{L}_{flow} = -\sum_{i=1}^{R} w_i \log q_\zeta(U_i|C). \tag{3.12}$$

This process is equivalent to performing mirror descent on the variational free energy, see [112] for a full derivation. In practice, when sampling $U_{1..R}$ from $q_\zeta(U|C)$ we can optionally add a Gaussian perturbation to the samples, decaying the magnitude of the perturbation during training. While this means we are no longer performing gradient descent on $\mathcal{F}$ exactly, we found that this empirically improved exploration during training. Doing this however, results in less smooth trajectories. We use both options in our method; for experiments in which smoothness is particularly important, we do not include this noise. To train the parameters of our system we perform the following optimization via stochastic gradient descent:

$$\min_{\theta,\phi,\psi,\omega,\zeta} \mathcal{L}_{flow} + a\mathcal{L}_{VAE}, \tag{3.13}$$

for scalar $a \geq 0$. We use a combined loss and train end-to-end so that h is explicitly trained to be used to condition the control sequence posterior. We then continue training the control sequence posterior with a fixed VAE with the optimization

$$\min_{\omega,\zeta} \mathcal{L}_{flow}. \tag{3.14}$$

3.4.4 Using the control sequence posterior for sample-based MPC

In this section we introduce two approaches for using the learned control sequence posterior with sample-based MPC controllers, FlowMPPI and FlowiCEM, based on

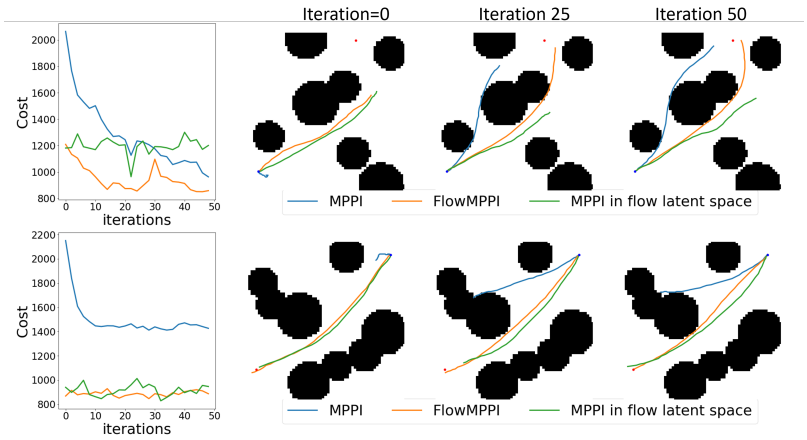


Figure 3.3: Two examples in which we run 50 iterations of MPPI, FlowMPPI and MPPI in the latent space of the flow in a 2D navigation task with double-integrator dynamics. Top: Initial samples from the Flow are goal directed, but do not yet fully reach the goal, in contrast the initial samples for MPPI perform poorly. We see that while MPPI can improve with successive iterations, running MPPI in the latent space of the flow fails to improve the trajectory. In contrast FlowMPPI starts with a better initialization and is able to improve faster than MPPI. Bottom: Here the initial samples form the control sequence posterior have already reached the goal, and so no improvement is necessary. In contrast, since MPPI is only performing local improvements to the control sequence it becomes stuck in a local minima.

MPPI [167] and iCEM [119] respectively. Given a C computed from (x_0, x_G, ρ, E), the control sequence posterior $q_\zeta(U|C)$ can be used as a sampling distribution. For each of these methods, we use the same $q_\zeta(U|C)$ learned via the procedure outlined in the previous section.

3.4.4.1 FlowMPPI

MPPI iteratively perturbs a nominal control sequence with Gaussian noise and performs a weighted sum of the perturbations to find a new control sequence. It is thus a method for local optimization, and the connection between MPPI and mirror descent was noted in [158]. As a local optimization method it is susceptible to becoming stuck in local minima given improper initialization. By sampling Gaussian perturbations the algorithm is uninformed about the perturbation direction expected

to lower cost.

In contrast, $q_\zeta(U|C)$ is able to directly sample collision-free goal-directed trajectories and produces highly informed samples. One way in which we might think to use $q_\zeta(U|C)$ in an MPC framework is to run MPPI in the latent space of the flow. This is appealing because $q_\zeta(U|C)$ generates low-cost control sequence. This means we would be searching directly in the space of low-cost control sequences and we would not waste many samples exploring high-cost control sequences. Unfortunately this method does not work well in practice, as visualized in (Figure 3.3). We note that when performing MPPI in the latent space Z of $q_\zeta(U|C)$, while the initial samples are usually relatively low-cost we fail to locally improve the control sequence with successive iterations.

One challenge of performing MPPI in the latent space of the flow is that small changes in Z often lead to large differences in the resulting control sequence. Another additional challenge is that by performing MPPI in Z we can only ever generate control sequences that are produced by $q_\zeta(U|C)$. While in principle, given a highly expressive model that is trained to minimize eq. (3.4), all low-cost control sequences should have high density under $q_\zeta(U|C)$, there will inevitably be an approximation gap, i.e. $\mathcal{KL}\left(q_\zeta(U|C)||p(\tau|o=1)\right) > 0$. This approximation gap is likely to increase in OOD situations.

To avoid this, our first proposed MPC algorithm, FlowMPPI, uses samples from $q_\zeta(U|C)$ while also allowing the control sequences to improve further beyond what can be sampled from $q_\zeta(U|C)$. FlowMPPI combines sampling in the latent space Z, and sampling perturbations to trajectories to get the advantages of both. For a given sampling budget K, we generate half of the samples from perturbing the nominal trajectory as in MPPI, and the other half from sampling from the control sequence posterior. These samples will be combined as in standard MPPI.

The algorithm for a single step of FlowMPPI is shown in Algorithm (3). For a step of FlowMPPI we first perform a shift operation on the previous nominal control sequence, replacing the final control in the sequence with Gaussian noise, shown on lines 3-5. We generate half of the samples and compute their respective costs via the standard MPPI approach using a Gaussian perturbation to the nominal control

sequence, shown lines 6-13. We generate the samples from the flow by first sampling from a standard Normal distribution and then mapping these samples through the control sequence posterior flow to generate control sequences. We then evaluate the costs of these sequences, including a perturbation cost on the distance of the sampled control sequence from the nominal. This cost is given in the algorithm by the cost $S_{nominal}$. This cost can be computed either in the flow latent space with $S^Z_{nominal} = \lambda \epsilon_Z^T (f_\psi^{-1}(U, C) - \epsilon_Z)$. This mirrors the similar cost in the original MPPI algorithm. However, this requires querying the inverse of the control sequence normalizing flow. An alternative is $S^U_{nominal} = \lambda ||U_k - U)||^2_{\Sigma^{-1}}$. We make use of both of these in our methods. Finally, we compute the new nominal control sequence via a weighted sum of the sampled control sequences, where the weights are determined by the exponentiated negative costs.

3.4.4.2 FlowiCEM

CEM [77] is an iterative sample-based MPC algorithm which uses a Gaussian sampling distribution. It samples control sequences, selecting the N_{elites} *elites* with the lowest cost, and refitting the Gaussian sampling distribution to those elites. iCEM [119] is a recent method that builds on CEM; where CEM uses a Gaussian as the sampling distribution, iCEM uses colored Gaussian noise. iCEM also maintains low-cost control sequences between iterations, rather than discarding sampled control sequences after an iteration is complete.

As with FlowMPPI, to incorporate $q_\zeta(U|C)$ into iCEM, we do not perform iCEM exclusively in the latent space Z (for the reasons discussed in Section 3.4.4.1). Similar to MPPI, we found that iCEM is very good at locally optimizing a control sequence, and that performing iCEM in Z results in a failure to improve further on the initial samples. We take a similar approach as with FlowMPPI, proposing an algorithm, FlowiCEM, that uses some samples from $q_\zeta(U|C)$ while still allowing further improvement.

In FlowiCEM, we add samples from the control sequence posterior by adding to the initial population at the beginning of every time-step. These samples can

Algorithm 3 A single step of FlowMPPI, this will run every timestep.

Inputs: Cost function J, previous nominal trajectory U, Context vector $C = g_\omega(x_0, x_G, \rho, h)$, control sequence posterior flow f_ζ, MPPI hyperparameters (λ, Σ), Horizon T, Samples K

function FLOWMPPISTEP
  ▷ Perform shift operation on nominal U
  for t $\in \{1, ..., T-1\}$ **do**
    $U_{t-1} \leftarrow U_t$
  $U_{T-1} \sim \mathcal{N}(0, \Sigma)$
  ▷ Generate samples by perturbing nominal U
  for k $\in \{1, ..., \frac{K}{2}\}$ **do**
    $\epsilon_U \sim \mathcal{N}(0, \Sigma)$
    $U_k \leftarrow U + \epsilon_U$
    $\tau_k \sim p(\tau | U_k)$ ▷ Sample trajectory
    $S_k \leftarrow J(\tau_k) + \lambda U_k \Sigma^{-1} \epsilon_U$ ▷ Compute cost
  ▷ Generate samples from control sequence posterior
  for k $\in \{\frac{K}{2} + 1, ..., K\}$ **do**
    $\epsilon_Z \sim \mathcal{N}(0, I)$
    $U_k \leftarrow f_\zeta(\epsilon_Z, C)$
    $\tau_k \sim p(\tau | U_k)$ ▷ Sample trajectory
    $S_k \leftarrow J(\tau_k) + S_{nominal}$ ▷ Compute cost
  ▷ Compute new nominal U
  $\beta \leftarrow \min_k S_k$
  $\eta = \sum_{k=1}^{K} \exp(-\frac{1}{\lambda}(S_k - \beta))$
  for k $\in \{1, ..., K\}$ **do**
    $w_k \leftarrow \frac{1}{\eta} \exp(-\frac{1}{\lambda}(S_k - \beta))$
  $U \leftarrow \sum_{k=1}^{K} w_k U_k$
  return U

be thought of as an initial set of 'elites', i.e. low-cost control sequences. Further iterations proceed as normal with iCEM.

A single step of the FlowiCEM algorithm is shown in Algorithm 4. For a single step of FlowiCEM, we first shift the control sequence mean, replacing the final mean Gaussian with noise. If this is not the first step, we also perform the shift operation on the elites kept from the previous step. We sample a set of elites from the control-sequence posterior in lines 8-9. We then proceed for several iterations of the sample-based optimization. First we sample control sequences from a colored noise distribution. If this is the first iteration, we introduce the elites sampled from the flow into the population. For all iterations, we introduce the elites kept from previous iterations into the population. We then compute the costs, and use the best N_{elites} elites to update the parameters of the Gaussian used to sample control sequences. We then select the best $N_{kept-elites} < N_{elites}$ elites to be maintained for the next iteration and proceed with the next iterations. Once we have finished all iterations we return the lowest-cost elite.

3.4.5 Generalizing to OOD Environments

A novel environment can be OOD for the control sequence posterior and result in poor performance. We present an approach where we *project* the OOD environment embedding h in-distribution in order to produce low-cost trajectories when it is used as part of the input to f_ζ. The intuition behind this approach is that our goal is to sample low-cost trajectories in the current environment. Given that f_ζ will have been trained over a diverse set of environments, if we can find an in-distribution environment that would elicit similar low-cost trajectories, then we can use this environment as a proxy for the actual environment when sampling from the flow. Thus we avoid the problem of samples from the control sequence posterior being unreliable when the input is OOD.

In order to do this projection, we first need to quantify how far out-of-distribution a given environment is. Once we have such an OOD score, we will find a proxy environment embedding $\hat{h}$ by optimizing the score, while also regularizing to encourage

Algorithm 4 A single step of FlowiCEM, this will run every timestep

Inputs: Cost function J, previous mean control sequence μ, Context vector $C = g_\omega(x_0, x_G, \rho, h)$, control sequence posterior flow f_ζ, iCEM hyperparameters $(\alpha, \gamma, \sigma^2, M, N)$, Horizon T, Samples K

function FLOWICEMSTEP
  ▷ Perform shift operation on nominal mean
  $\mu \leftarrow$ shifted μ
  if $U_{kept-elites} \neq \{\}$ **then**
    $U_{kept-elites} \leftarrow$ shifted $U_{kept-elites}$
  $\mu_{T-1} \sim \mathcal{N}(0, \Sigma)$
  ▷ Generate initial elites from flow
  $\epsilon_Z \sim \mathcal{N}(0, I)$
  $U_{flow} \leftarrow f_\zeta(\epsilon_Z, C)$
  for i $\in \{1, ..., \text{iters}\}$ **do**
    ▷ Generate samples from colored noise
    $U \leftarrow$ SAMPLECOLOREDNOISE(μ, σ^2)
    ▷ Add samples from flow to population
    if i $== 0$ **then**
      $U \leftarrow U \cup U_{flow}$
    ▷ Add kept-elites to population
    $U \leftarrow U \cup U_{kept-elites}$
    $\tau \sim p(\tau|U)$ ▷ Sample trajectories
    costs $\leftarrow J(\tau)$ ▷ Evaluate costs
    $U_{elites} \leftarrow N_{elites}$ lowest-cost trajectories
    $\mu \leftarrow (1-m)\text{mean}(U_{elites}) + m\mu$
    $\sigma \leftarrow (1-m)\text{std}(U_{elites}) + m\sigma$
    $U_{kept-elites} \leftarrow$ Best $N_{kept-elites}$ elites
  return lowest-cost elite from $U_{kept-elites}$

low-cost trajectories. For the OOD score, we use the VAE we have discussed in section 3.4.2. VAEs and other deep latent variable models have been used to detect OOD data in prior work [38, 169, 109], however these methods are typically based on evaluating the likelihood of an input, in our case $p(E)$. For VAEs this requires reconstruction. We would like to avoid using reconstruction in our OOD score for two reasons. First, reconstruction, particularly of a 3D SDF, adds additional computation cost and we would like to evaluate the OOD score in an online control loop. Second, optimizing an OOD score based on reconstruction would drive us to find an environment embedding proxy which is able to approximately reconstruct the *entire* environment. This makes the problem more difficult than is necessary, as we do not need $\hat{h}$ to accurately represent the entire environment, only to elicit low-cost trajectories from the control sequence posterior.

To determine how close h is to being in-distribution, we use the OOD score

$$\mathcal{L}_{OOD}(h) = -\log p_\phi(h), \tag{3.15}$$

where $p_\phi(h)$ is the learned flow prior for the VAE. The intuition for using this as an OOD score is that this term is minimized for the dataset in $\mathcal{L}_{VAE}$, so we should expect it to be lower for in-distribution data. Previous work on OOD detection using Normalizing Flows [75] found that using the likelihood of a Normalizing Flow as an OOD score is more effective for image data when using a feature representation of the input which contains higher-level semantic information compared with using the raw pixel values. The authors hypothesize that the failure to successfully distinguish between in-distribution and OOD data when using raw pixel values is the over-reliance on local pixel correlations. We train the environment embedding h end-to-end both for reconstruction and to be used for generating collision-free control sequences. For this reason we hypothesize that our latent embedding h contains higher level information on the structure of the environment, and hence the learned flow prior likelihood is a more effective OOD score. Further motivating our approach, using a learned prior was shown to improve density estimation over a Gaussian prior [24]. Likewise, we found the learned prior yielded much better OOD detection than

using a Gaussian prior, which is the standard VAE prior (see Figure 3.4).

We can perform gradient descent on $\mathcal{L}_{OOD}$ to find $\hat{h}$, thus *projecting* the environment to be in-distribution. Note that without regularization this process will converge to a nearby maximum likelihood solution, which may lose key features of the current environment. Since our aim is to sample low-cost trajectories from the control sequence posterior, we use $\mathcal{L}_{flow}$ as a regularizer for this gradient descent. Our intuition here is that in order to generate low-cost trajectories in the true environment, the projected environment embedding should preserve important features of the environment relevant for that particular planning query. The new environment embedding is then given by

$$\hat{h} = \arg\min_{h} \; b\mathcal{L}_{OOD} + \mathcal{L}_{flow}, \tag{3.16}$$

for scalar $b > 0$. We project h to $\hat{h}$ by minimizing the equation 3.16 gradient descent. This step is incorporated into our proposed MPC methods and we call the resulting methods FlowMPPIProject and FlowiCEMProject. This version of our method will perform M steps of gradient descent on the above combined loss at initialization, followed by a single step at each iteration of the MPC. The algorithm for projection is shown in algorithm 5. The algorithm SamplePertU is shown in the appendix.

Algorithm 5 Projection

Inputs: N iterations, K samples, $\theta, \phi, \omega, \zeta$ parameters, control perturbation covariance Σ_ϵ, learning rate η, loss hyperparameters (α, β, b)

$h^1 \leftarrow \phi$
for n $\in \{1, ..., N\}$ **do**
  Compute $\log p_\phi(h^n)$ via eq. (3.7)
  $C \leftarrow g_\omega(x_0, x_G, h^n)$
  $\{U_k, q_\zeta(U_k|C)\}_{k=1}^K \leftarrow \textsc{SamplePertU}(C, \Sigma_\epsilon, K)$
  $\mathcal{L} \leftarrow -p_\phi(h^n)$
  for k $\in \{1, ..., K\}$ **do**
    $w_k \leftarrow$ from $(\{U_i, \log q_\zeta(U_i|C)\}_{i=1}^K, \alpha, \beta\})$ via (3.11)
    $\mathcal{L} \leftarrow \mathcal{L} - w_k \cdot \log q_\zeta(U_k|C)$
  $h^{n+1} \leftarrow h^n - \eta \frac{\partial \mathcal{L}}{\partial h}$

3.5 Results

In this section, we will evaluate our proposed approaches FlowMPPI & Flow-iCEM with and without projection on three simulated systems; a 2D point robot, a 3D 12DoF quadrotor and a 7DoF manipulator. For each system, we will train the flow on a dataset of starts, goals and environments and evaluate the performance on environments drawn from the same distribution. In addition, for each system we will test on novel environments that are radically different from those used for training and evaluate the generalization of our approach and the ability of our projection approach to adapt to these OOD environments. For the 12DoF quadrotor system and the 7DoF manipulator, we additionally evaluate our method in simulation on environments generated from real-world data. Our goal is to evaluate if the control sequence posterior, trained on simulated environments, can adapt to real-world environments.

For our novel environments, we select environments which are difficult for sampling-based MPC techniques. We will use the terms "in-distribution" and "out-of distri-

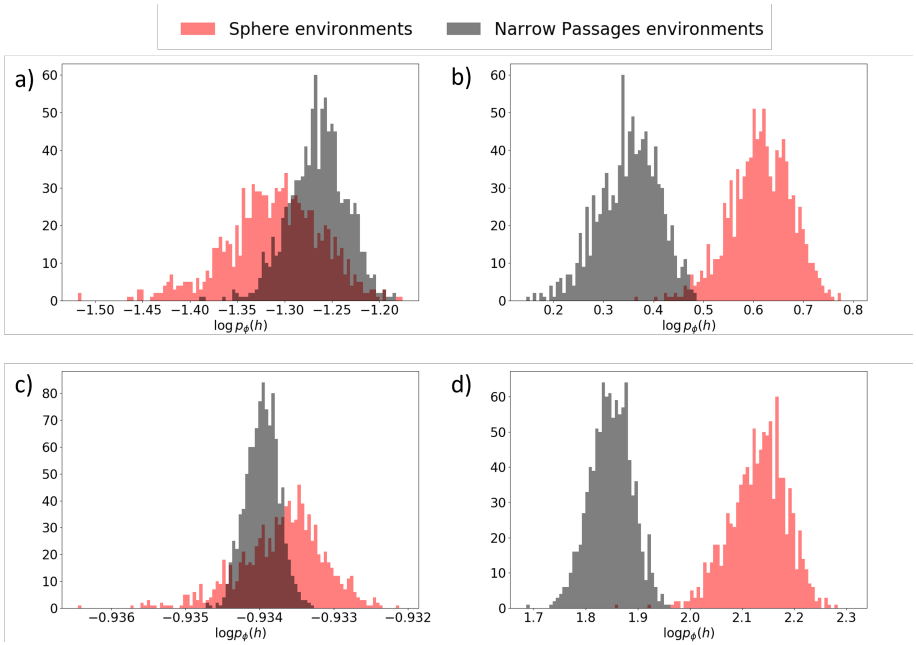


Figure 3.4: Comparison of our OOD scores with using a VAE with a standard Gaussian prior for in-distribution (red) and out-of-distribution (grey) simulated environments. a) planar navigation using a Gaussian prior, b) planar navigation using a Normalizing flow prior, c) 12DoF quadrotor using a Gaussian prior, d) 12DoF quadrotor using a Normalizing flow prior, These scores are computed by sampling h from ϕ and evaluating $\log p_\phi(h)$. The score is normalized by the dimensionality of h. We see that our method, shown in (b) and (d), achieves a clear separation between in-distribution and out-of-distribution environments in both cases.

bution" for environments for the rest of this section, but note that these terms are relative to the set of environments which we use to train our method. Being out-of-distribution has no bearing on the non-learning based baselines. The performance of non-learning sampling-based MPC algorithms depends only on the given environment, not its relation to other environments.

We evaluate our proposed algorithms on the resulting attained costs, success rates, and, for the 12DoF quadrotor and 2DoF double integrator, smoothness of the resulting controls. To compute smoothness we use

$$\mathcal{L}_{smooth}(U) = \sum_{t=1} ||u_t - u_{t-1}||^2. \tag{3.17}$$

3.5.1 Systems & Environments

In this section, we will introduce the systems and the environments we use for evaluation. For all systems and environments, a task is considered a failure if there is a collision or if the system does not reach the goal region within a timeout of 100 timesteps. The cost function for all systems is given by $J(\tau) = 100 d_G(x_T) + \sum_{t=1}^{T-1} 10 d_G(x_t) + \sum_{t=1}^{T} 10000 D(x_t) + \rho_v ||v_t||_2^2$, where T is the MPC horizon, d_G is a distance to goal function, and D is an indicator function which is 1 if x_t is in collision and 0 otherwise, and v is the velocity. The exception is that the 7DoF manipulator does not have a velocity so that term is omitted. For all of our experiments, the control horizon $T = 40$.

We use a Gaussian prior over controls, assuming each control dimension is independent from one another. To encourage smoothness, we make control dimensions correlated across time, by computing the covariance for the ith control dimension u^i as $\Sigma^i_{t,t'} = \sigma^2 \exp -\frac{||u^i_t - u^i_{t'}||^2}{l}$. Here σ controls the magnitude of the controls and l controls the smoothness. The prior is then given by $p(U) = \mathcal{N}(0, \Sigma)$. Combining the cost and the control prior yields the total cost $\hat{J}(\tau) = J(\tau) + \log p(U)$, with control cost becoming the weighted l2-norm of the controls weighted by Σ. For all of our experiments, the dynamics are deterministic. The parameterization of the total cost function is $\rho = [\rho_v, \sigma^2, l]$. Further details of the generation of training data can be

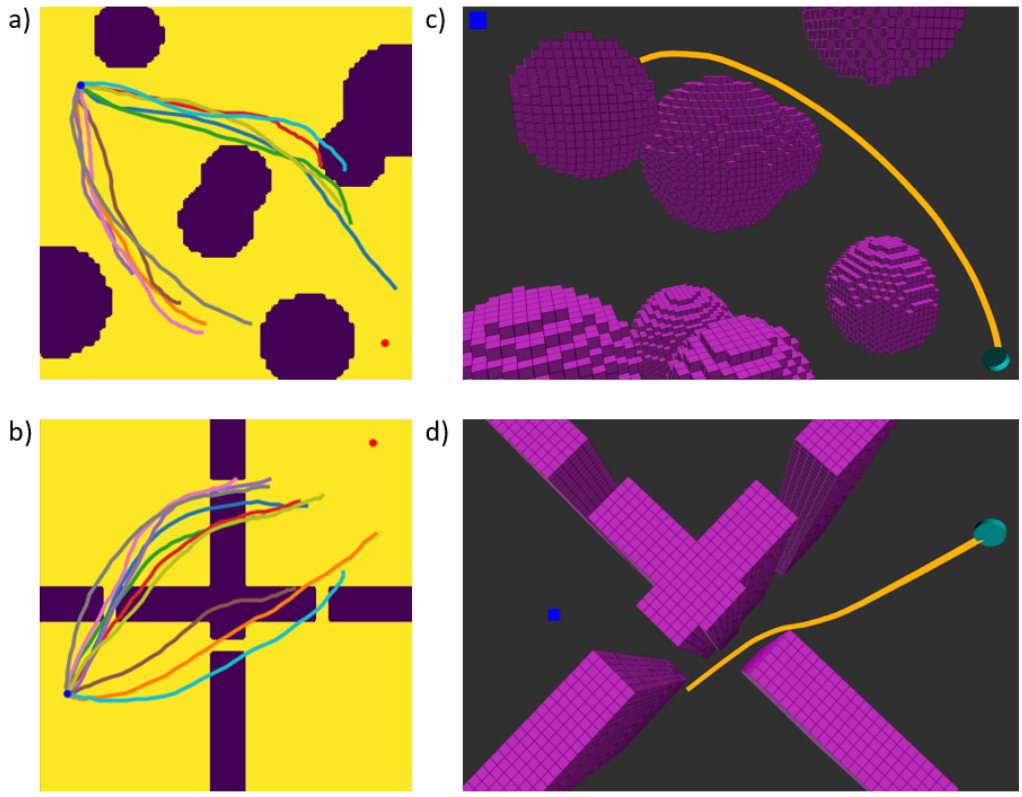


Figure 3.5: Examples of our 'in-distribution' environments (top) and 'out-of-distribution' environments (bottom). a) The sphere environment for the planar navigation task, showing sampled trajectories from the flow. b) The narrow passages environment for planar navigation, we see that the samples from the flow are goal orientated and generally toward the passages, but most are generally not collision free. c) The sphere environment for the 12DoF quadrotor. d) The narrow passages environment for the 12DoF quadrotor

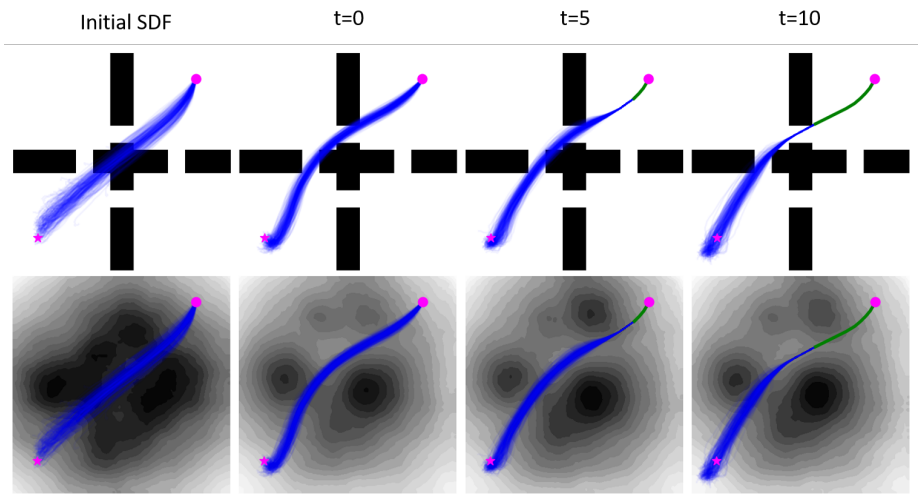


Figure 3.6: The projection process visualized for the planar navigation task. We visualize the projected environment embedding using the VAE decoder on the bottom row. Note that decoding h is only used for training the VAE and visualization, it is not necessary for projection. The top shows the environment and sampled trajectories from $q_\zeta(U|C)$. The bottom shows the same samples overlaid on a reconstruction of projected environment embedding $\hat{h}$. On the left, the initial SDF is very poor, and sampled control sequences result in trajectories passing directly through the obstacle. As the task progresses, the iterative projection results in an SDF that resembles the training environment more. The environment embedding encodes obstacles that result in a trajectory that traverses the narrow passage. Notice however, that regions that are not relevant for this planning task, such as the lower wall, do not need to accurately represent the environment.
found in appendix A.1.2.

3.5.1.1 Planar Navigation

The robot in the planar navigation task is a point robot with double-integrator dynamics. The goal is to perform navigation in an environment cluttered with obstacles. The state and control dimensionality are 4 and 2, respectively. The environment is represented as a 64×64 SDF. Examples of the training and evaluation environments are shown in Figure 3.5 (a & b). The training environments consist disc-shaped obstacles, where the size, location and number of obstacles is randomized. The out-of-distribution environment consists of four rooms, with narrow passages randomly

generated between them. The location of the passages is randomized for each OOD environment. The distance-to-goal function is $d_G(x) = ||x - x_G||_2$. The goal region for this task is given by $\mathcal{X}_G = \{x : ||x - x_G||_2 < 0.1\}$. We consider cost parameters in the range $\rho_v \in [0.01, 1]$, $\sigma^2 \in [1, 10]$ and $l \in [0.02, 2]$. During evaluation, we evaluate with three different settings of ρ. The first is $[0.1, 4, 0.2]$ which moderately penalizes the velocity, control magnitude, and controller smoothness. The next is $[0.01, 8, 0.02]$ which represents a more aggressive cost that has lower penalty on velocity, control magnitude, and smoothness. The next is a more conservative cost function defined by $\rho = [1, 1, 2]$, with a stronger penalty on velocity, control magnitude and smoothness. For each cost function we fix a single OOD environment and perform 100 control trials. For FlowMPPI, we use $S^U_{nominal}$ to compute the cost to the nominal trajectory, avoiding an additional query to the control sequence posterior flow. The dynamics for this system are shown in appendix A.1.3.

3.5.1.2 3D 12DoF Quadrotor

This system is a 3D 12DoF underactuated quadrotor with the shape of a short cylinder. It has state and control dimensionality of 12 and 4, respectively. As with the planar navigation task, the goal is to perform navigation in a cluttered environment. Examples of the training and evaluation environments are shown in Figure 3.5 (c & d). The training environment consists of spherical obstacles of random size, location, and number, and the out-of-distribution environment of four rooms separated by randomly generated narrow passages. The environment is represented as a $64 \times 64 \times 64$ SDF. The goal region is specified as a 3D position p_G. The distance-to-goal function is $d_G(x) = ||Ax - p_G||_2 + \rho_v||Bx||_2$ where A selects the position components from the state x, and B selects the angular velocity components. The goal region is $\mathcal{X}_G = \{x : d_G(x) < 0.3\}$. During training, we consider cost parameters in the range $\rho_v \in [0.01, 1]$, $\sigma^2 \in [4, 40]$ and $l \in [0.02, 2]$. During evaluation, we evaluate on three different settings of ρ. The first is $[0.1, 25, 0.02]$. The next is $[0.1, 25, 0.02]$, which encourages smooth behavior without strongly penalizing the magnitude of the controls. The next is a more conservative cost function defined

by $\rho = [1, 12, 0.2]$, which penalizes velocities and control magnitude more strongly. For each cost function, we fix a single OOD environment and perform 50 control trials. We also tested in two simulation environments generated from real-world data (shown in 3.1). For FlowMPPI, we use $S^U_{nominal}$ to compute the cost to the nominal trajectory, avoiding an additional query to the control sequence posterior flow. The dynamics for this system are shown in appendix A.1.4.

3.5.1.3 7DoF Manipulator

This system is a kinematic 7DoF manipulator shown in different environments in Figure 3.7. The state and control dimensions are both 7. The goal is to reach a target end-effector position in the presence of obstacles. The training environment consists of spherical obstacles, shown in Figure 3.7(a). The number and size of the spherical obstacles is randomized during training. The simulated novel environments are shown in Figure 3.7(b-d). We additionally evaluate on one environment generated from real-world data, shown in Figure 3.7(d,e). The environment is represented as a $64 \times 64 \times 64$ SDF. The goal region is specified as a 3D position p_G. To generate the SDF we generated pointclouds from several different views with a KinectV2. We used motion capture to determine the camera frame and aggregated the point clouds together. The distance-to-goal function is $d_G(x) = |\texttt{ForwardKinematics}(x) - p_G||_2$. The goal region is $\mathcal{X}_G = \{x : d_G(x) < 0.1\}$. For this task we keep the cost parameters ρ constant with $\sigma^2 = 4$. We do not include the smooth prior, effectively taking $l \to \infty$. Since the 7DoF manipulator task is quasi-static, we do not include the velocity penalty for this system. To perform fast batched collision checking on the GPU using the environment SDF we approximate the robot geometry as a set of spheres. For FlowMPPI, we use $S^Z_{nominal}$ to compute the cost to the nominal trajectory, as this task is kinematic, computation time is less important.

3.5.2 OOD Score and Projection

To confirm the efficacy of our OOD score, we computed this score for the training and OOD environments for each system above. Figure 3.4 shows that this score is

clearly able to distinguish in-distribution environment embeddings from OOD ones.

3.5.3 Network Architectures

For both the control sequence posterior flow f_ζ and the VAE prior $p_\phi(h)$ we use an architecture based on Real-NVP [33]. For the VAE prior $p_\phi(h)$ we use a flow depth of 4, while for the control sampling flow f_ζ we use a flow depth of 12 for the 12DoF quadrotor and 2DoF double integrator, and 20 for the 7DoF manipulator. For the control sampling flow we use the conditional coupling layers from [168]. For the VAE encoder we use four CNN layers with a kernel of 3 and a stride of 2, followed by a fully connected layer. For the VAE decoder we used a fully connected layer followed by four transposed CNN layers. For the 3D case we use 3D convolutions. The dimensionality of both h and C was 256 for all tasks. g_ω was defined as an MLP with a single hidden layer of size 256. For nonlinear activations, we used ReLU throughout. We implement all networks using PyTorch [115].

3.5.4 Training & Data

For training, we use 10000 randomly generated environments for planar navigation task, and 20000 for the 3D 12DoF quadrotor and 7DoF manipulator tasks. At each epoch, for each environment, we randomly select one of 100 start and goal pairs, and also randomly sample cost parameters ρ. We train the control sequence posterior flow f_ζ, the VAE parameters (θ, ϕ, ψ) and the context MLP g_ω end-to-end using Adam. After 100 epochs, we freeze the VAE and do not continue training with $\mathcal{L}_{\mathcal{VAE}}$. This is primarily because the VAE converges quickly and training proceeds more quickly without reconstruction. When training the VAE we divide the loss by the total dimensionality of the SDF and use $a = 5$. For the double integrator and quadrotor tasks, we train the control sequence posterior without perturbing the samples with noise. These are second-order systems and thus it is important to minimize the controller jerk. For the 7DoF manipulator planning, we train with the noise. We found that without using the noise for the 7DoF manipulator the resulting control sequence posterior was not able to generate diverse enough samples for successful

use as an MPC sampling distribution.

A full list of training hyperparameters can be found in appendix A.1.

3.5.5 Baselines

For our baselines, we use several state-of-the-art sampling-based MPC methods: MPPI [167], Stein Variational MPC (SVMPC) [84] and improved CEM (iCEM) [119]. MPPI uses a Gaussian distribution as the sampling distribution, iCEM uses colored noise, and SVMPC uses a mixture of Gaussians. For each baseline, we tune the hyperparameters to get the best performance based on the training environments, and maintain these hyperparameters when switching to the out-of-distribution environments. We evaluate each method with a sampling budget of 512. This means that for methods that require multiple iterations per timestep, the sampling budget is distributed across the iterations. A more detailed list of the hyperparameters for each controller can be found in appendix A.1.5.

Table 3.1: Comparison of methods for the Planar Navigation Tasks. We evaluate on both in-distribution environments and OOD environments across different cost function parameters ρ.

	In-Distribution Environment $\rho = [0.1, 25, 0.2]^T$			Out-of Distribution Environments $\rho = [0.1, 25, 0.2]^T$			$\rho = [0.1, 25, 2]^T$			$\rho = [1, 12, 0.2]^T$		
Controller	Success	Cost	$\mathcal{L}_{smooth}$	Success	Cost	$\mathcal{L}_{smooth}$	Success	Cost	$\mathcal{L}_{smooth}$	Success	Cost	$\mathcal{L}_{smooth}$
MPPI	0.82	1829	15.4	0.24	2791	16.7	0.16	2842	18.2	0.15	3103	7.86
SVMPC	0.82	1824	6.60	0.08	3014	6.36	0.08	2991	7.62	0.09	3058	2.55
iCEM	0.87	1427	**1.20**	0.37	2175	**0.873**	0.41	2150	**1.13**	0.42	2142	**0.662**
FlowMPPI	0.97	1084	17.0	0.85	1529	20.8	0.92	1399	27.8	0.75	1867	7.24
FlowMPPIProject	0.96	1097	16.7	**0.95**	**1328**	22.4	**0.95**	**1274**	26.8	**0.8**	1783	7.34
FlowiCEM	0.97	1038	4.26	0.77	1678	3.64	0.77	1647	6.10	0.72	1801	1.27
FlowiCEMProject	**0.98**	**1008**	4.22	0.77	1633	3.71	0.77	1583	7.49	0.74	1777	1.59

Table 3.2: Comparison of methods for the 12DoF quadrotor task. We evaluate on both in-distribution environments and OOD environments across different cost function parameters ρ.

	In-Distribution Environment $\rho = [0.1, 25, 0.2]^T$			Out-of Distribution Environments $\rho = [0.1, 25, 0.2]^T$			$\rho = [0.1, 25, 2]^T$			$\rho = [1, 12, 0.2]^T$		
Controller	Success	Cost	$\mathcal{L}_{smooth}$	Success	Cost	$\mathcal{L}_{smooth}$	Success	Cost	$\mathcal{L}_{smooth}$	Success	Cost	$\mathcal{L}_{smooth}$
MPPI	0.50	2999	14.2	0.06	4307	53.0	0.08	4079	46.3	0.0	4047	72.0
SVMPC	0.32	3580	89.0	0.00	4768	17.7	0.00	4671	16.7	0.00	4400	16.9
iCEM	0.60	2775	**1.55**	0.14	4069	**1.64**	0.24	3784	**1.66**	0.00	4397	**1.25**
FlowMPPI	0.85	2138	67.3	0.36	3077	60.8	0.80	2876	52.3	0.78	2504	58.2
FlowMPPIProject	0.96	**1933**	67.7	**0.9**	**2569**	58.2	**0.98**	**2560**	49.5	**0.84**	**2385**	54.0
FlowiCEM	0.68	2556	106	0.62	3474	67.4	0.62	3470	45.0	0.38	3026	68.8
FlowiCEMProject	**0.98**	2225	108	0.72	3272	101.7	0.86	3047	67.7	0.68	2654	84.4

Table 3.3: Computational times

System	MPPI	SVMPC	iCEM	FlowMPPI	FlowMPPIProject	FlowiCEM	FlowiCEMProject
Planar Navigation	0.0078	0.052	0.032	0.029	0.075	0.059	0.116
12DoF Quadrotor	0.084	0.124	0.087	0.076	0.136	0.134	0.195

Table 3.4: Comparison of methods for the 3D 12DoF quadrotor navigation task with two environments generated from real-world data. The rooms environment is shown in figure 3.5 (b) and the stairway environment is shown in figure 3.5 (a). We evaluate on 100 randomly sampled starts and goals in each environment.

	Rooms Environment $\rho = [0.1, 25, 0.2]^T$			**Stairway Environment** $\rho = [0.1, 25, 0.2]^T$		
Method	Cost	Success	$\mathcal{L}_{smooth}$	Cost	Success	$\mathcal{L}_{smooth}$
MPPI	0.34	2576	12.8	0.1	3194	13.6
SVMPC	0.00	3621	14.8	0.02	4255	12.8
iCEM	0.24	2749	**1.30**	0.12	2577	**1.32**
FlowMPPI	**0.94**	1643	69.6	0.44	2260	70.3
FlowMPPIProject	**0.94**	**1589**	67.2	**0.58**	**2045**	73.4
FlowiCEM	0.66	2386	102	0.38	2646	87.5
FlowiCEMProject	0.68	2068	100	0.54	2435	105.7

3.5.6 Results

The results comparing our MPC methods to baselines are shown in Tables 3.1, 3.2 3.4 and 3.5. In addition, box plots showing the distribution of costs are shown in Figures 3.9 and 3.10, and the computational times for all methods are shown in Table 3.3. For the planar navigation case, we see that all our proposed methods perform similarly for the in-distribution environment, as expected. All methods based on iCEM perform well for this task, with iCEM achieving the lowest cost of all baselines. In addition, iCEM reliably achieves the smoothest controls. For the out-of-distribution environments, all of our proposed flow variants reach the goal significantly more often. For example, the success rate for FlowMPPIProject is 0.95 for $\rho = [0.1, 25, 0.2]$ compared with the next closest baseline, iCEM, which attains a success rate of 0.85. While all of our proposed methods demonstrate strong performance in cost and success rate, they achieve lower control smoothness than their corresponding baseline method. For example, for the in-distribution environment, FlowiCEM results in a smoothness of 4.26 vs 1.20 for iCEM while improving the cost from 1427 to 1008. The flow-based methods show stronger control action, trading

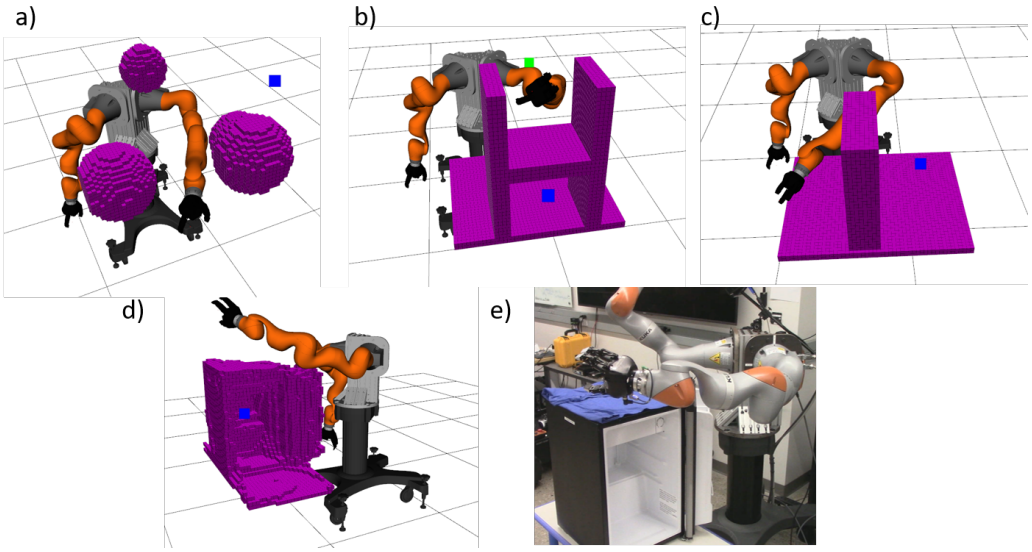


Figure 3.7: We evaluate our approach on control of a kinematic 7DoF manipulator on four environments in simulation (a-d). Tasks consist of a) Navigating around spherical obstacles b) Reaching into a shelf c) Going from one side of a wall to another d) Reaching inside a fridge e) Real world setup for the reaching into a fridge task. The voxel grid in d) was generated from the fridge in e) using multiple views of a Kinect v2.

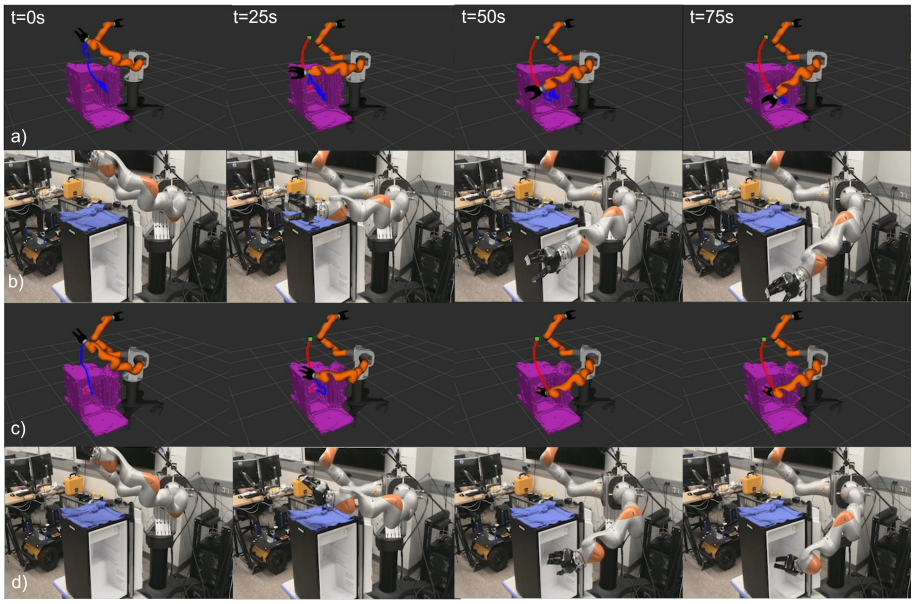


Figure 3.8: a,b) iCEM baseline performing a task where the goal is to navigate to to the inside of the fridge. The baseline fails to successfully navigate to the goal. c,d) One of our proposed methods, FlowiCEMProject, successfully navigating to the inside of the fridge.

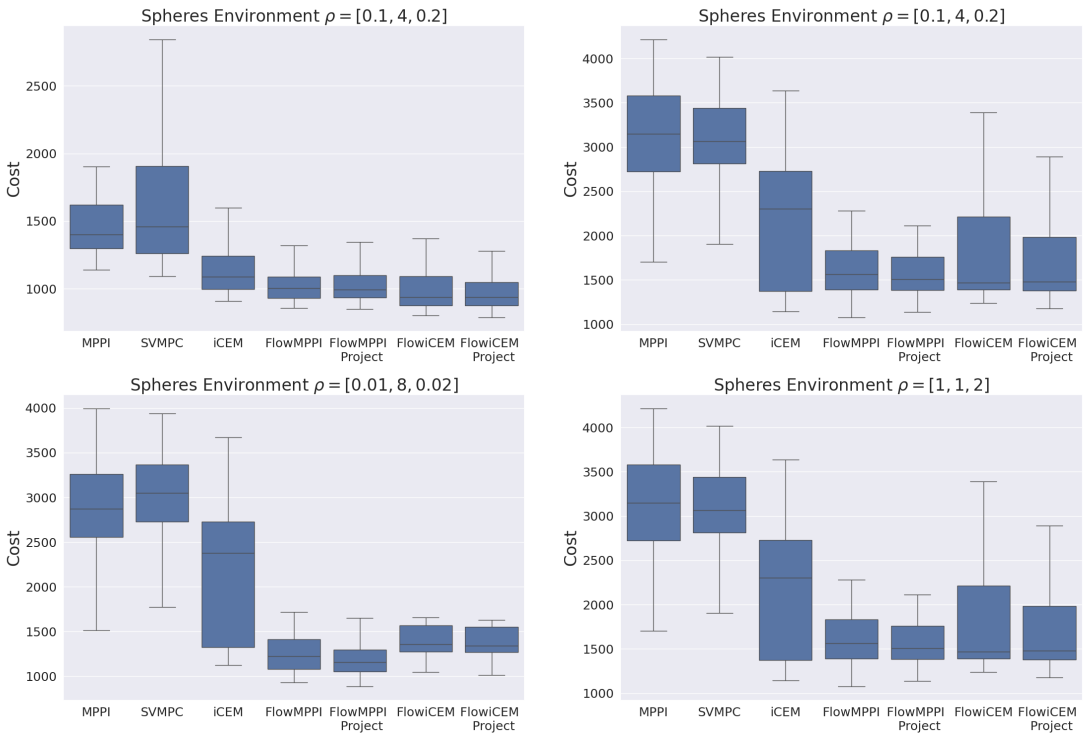


Figure 3.9: Box plot of the costs for the double integrator experiments. We evaluate on 100 trials for the training environment consisting of randomly generated disc obstacles. In addition we evaluate for 100 trials with three different cost parameterizations in three different environments consisting of 4 walls with narrow passages between them.

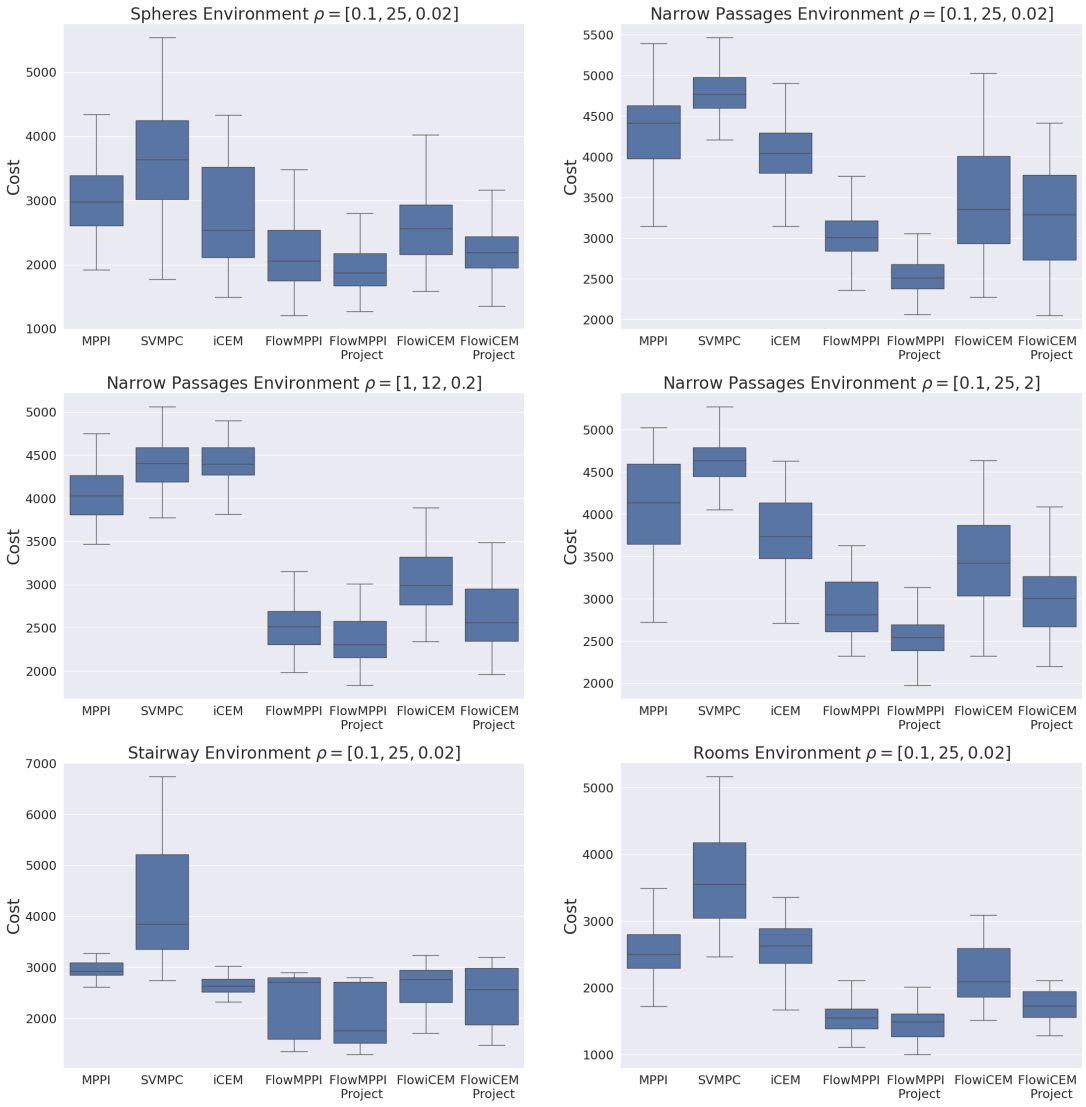


Figure 3.10: Box plot of the costs for the 12DoF quadrotor experiments. We evaluate on 50 trials for the training environment consisting of randomly generated disc obstacles. In addition, we evaluate 50 trials with three different cost parameterizations in three different environments consisting of 4 walls with narrow passages between them.

off smoothness for rapidly moving to the goal. Since the overall cost for these methods is lower, this suggests that this trade-off is desirable according to our given cost functions. The projection process for the planar navigation is shown in Figure 3.6). We observed during this experiment that when iCEM is able to generate a trajectory that reaches the goal region, they are able to locally optimize this trajectory better than FlowMPPI variants, while FlowMPPI is better able to generate sub-optimal trajectories to the goal region.

For the quadrotor system, FlowMPPIProject outperforms all other methods in both cost and success rate across all environments and cost parameterizations bar the training environment. and sampling budgets. For the cost parameterization $\rho = [0.1, 25, 0.2]$ evaluated on the Narrow Passages environment, FlowMPPIProject attains a 90% success rate compared to 14% by iCEM and 6% by MPPI for OOD environments. When we increase the smoothness parameter, FlowMPPI attains 98% vs 24% for iCEM and 8% for MPPI. Increasing the smoothness parameter in the cost does lead to a corresponding improvement in the $\mathcal{L}_{smooth}$ for all the methods other than iCEM. When evaluating on the more conservative cost $\rho = [0.1, 25, 0.2]$, all baselines fail with 0% success rate, while FlowMPPIProject attains 68%. The dynamics of the quadrotor task make it much more difficult, particularly because stabilizing around the goal is non-trivial. We found that the baselines struggled to find trajectories that both reached and stabilized to the goal and thus were more susceptible to becoming stuck in local minima.

Table 3.4 shows the results when evaluating our method in simulation in two environments generated from real-world data. FlowMPPIProject outperforms all other methods in cost & success rate, despite only being trained on simulated environments consisting of large spherical obstacles. For the challenging stairway environment, FlowMPPIProject achieves 58% success, while the next closest baseline, iCEM, has 44% success. FlowMPPI and FlowiCEM achieve only 44% and 54% success rate, respectively, for this task, rising to 78% and 53% when performing online projection, highlighting the importance of projection for real-world environments.

The results for the 7DoF manipulator experiment are shown in table 3.5. For this experiment, we use a fixed sampling budget of 512 samples for all methods.

On the in-distribution environment methods perform similarly, with FlowMPPI and FlowiCEM marginally improving on MPPI and iCEM, respectively, and projection resulting in further improvement. For the OOD environments, either FlowMPPIProject or FlowiCEMProject perform best in both success rate and average cost. For the Fridge environment, which was generated from real world data, FlowiCEMProject achieved 97% success rate compared with 89% for iCEM and 44% for SVMPC. Figure 3.1 shows FlowiCEMProject running on a 7DoF manipulator in real hardware. These results suggest that our learned flow-based posterior does indeed improve the performance of sampling-based MPC methods for a variety of tasks. It is especially encouraging that our methods succeed despite the testing environments being very different (i.e. OOD) with respect to the training environments, which demonstrates the generalization afforded by our OOD-projection approach.

Table 3.5: Results for attempting task 100 times for each environment in simulation. The environments are shown in Figure 3.7. The fridge environment is generated from real-world data from the fridge shown in Figure 3.7 (e)

.

	In-Distribution		Out-of Distribution					
	Spheres Environment		**Shelf Environment**		**Wall Environment**		**Fridge Environment**	
Method	Success	Cost	Success	Cost	Success	Cost	Success	Cost
MPPI	0.83	836	0.24	1900	0.12	1938	0.16	1944
SVMPC	0.82	737	0.08	2132	0.42	1628	0.44	1946
iCEM	0.85	694	0.66	1302	0.36	1768	0.89	898
FlowMPPI	0.85	698	0.65	1355	0.62	1280	0.74	1080
FlowiCEM	0.86	628	0.62	1339	0.44	1573	0.94	850
FlowMPPIProject	**0.87**	**582**	**0.75**	**1127**	0.64	1178	0.83	819
FlowiCEMProject	0.86	612	0.66	1268	**0.7**	**1109**	**0.97**	**798**

3.6 Discussion

While our results demonstrate the efficacy of using a flow-based posterior and OOD-project for MPC problems, our approach has several limitations. First, our experiments only consider navigation tasks where the objective is to reach a configuration while avoiding collision. This means that the cost functions are relatively easily parameterized with a start, goal, and SDF of the environment. In order to generalize our method to a wider range of tasks, such as robotic manipulation, we must be able to design a flexible task parameterization that we can use as input to the control sequence posterior. This is a topic we intend to explore in future work.

Second, while our experiments demonstrate that our OOD-projection approach enables our method to generalize to novel environments, the limits to this generalization are unknown. Given a novel environment, we do not have a way of predicting how well our projection method is likely to work without attempting the task in that environment. Our overall projection seeks to find an environment embedding that has a high likelihood according to the training distribution while minimizing the cost of sampled trajectories. This inevitably leads to some loss of information. In particular, the regularization term minimizing the cost of sampled trajectories can only encourage the preservation of environment details local to the sampled trajectories, so we can only expect a local approximation of the environment. In addition, finding an environment embedding with a high likelihood under the training distribution means we are unable to represent environment geometries that differ significantly from those seen during training.

Third, we have only considered the case where the environment is static. A simple method of applying to a dynamic environment could be incorporated by updating the SDF online and planning as if the scene were static. However, our projection method currently updates the environment embedding with a single gradient step per timestep to reduce the computation time, using the previous environment embedding as the initialization. If the environment SDF is reset every time-step then one gradient step may no longer be sufficient to adapt to novel environments. One interesting potential avenue for future work is to incorporate knowledge of the envi-

ronment dynamics during planning, by predicting the future environment SDFs as in [40]. By projecting these future environment SDFs in-distribution we may avoid the issue of having the environment representation reset, as we can warm-start from the projected future SDF. However, in our current approach the SDF is encoded once at the beginning of the task, this method would require encoding both current and predicted SDFs at every time-step, which would increase the computational cost.

Fourth, we note from Table 3.3 that incorporating projection requires significant computation time, and thus the current implementation cannot be used for real-time control. For both the 12DoF quadrotor and the 2DoF double integrator, the total computational time is larger than the simulation time-step for both FlowiCEMProject and FlowMPPIProject. Also note that the computation time is likewise longer than the simulation time-step for several of the baseline methods, including all methods for the 12DoF quadrotor. Our method and all baselines were implemented in Python, and implementing these methods in C++ may enable real-time performance on these systems in future work. The learned components of our system could be deployed in C++ using LibTorch [115].

Fifth, training an effective control-sequence posterior requires tuning system-dependent hyper-parameters. While some hyper-parameters can be automatically selected (see Appendix A.1.1 for details), tuning parameters α and β is necessary when considering a new system. These parameters control the trade-off between diversity and low-cost control-sequence samples, and this trade-off is sensitive to the scale of the objective, which is often system-dependent.

Finally, we assume that an accurate model of the dynamics is known. Since we are using the learned control-sequence posterior in the context of model-based control, we believe assuming access to a dynamics model is reasonable. However, if the dynamics model is inaccurate, the control sequence posterior will have been learned using data from an inaccurate model. MPC is often robust to model errors, but it is unclear how the performance will be affected by using the inaccurate learned control sequence posterior.

3.7 Conclusion

In this chapter, we have presented a framework for using a Conditional Normalizing Flow to learn a control sequence sampling distribution for MPC based on the formulation of MPC as Variational Inference. The control sequence posterior samples control sequences which result in low-cost trajectories that avoid collision. We have shown how this control sequence posterior can be used in two different sampling-based MPC methods, FlowMPPI and FlowiCEM. We have also proposed a method for adapting this control sequence posterior to OOD environments by *projecting* the representation of the environment to be in-distribution, essentially "hallucinating" an in-distribution environment which elicits low-cost trajectories from the control sequence posterior. We have demonstrated that incorporating our learned sampling distribution into MPC algorithms offers large improvements over baselines in difficult environments and that by performing the environment projection we can successfully transfer a control sequence posterior learned with simulated environments to environments generated from real-world data.

CHAPTER IV

Constrained Stein Variational Trajectory Optimization

In this chapter we present Constrained Stein Variational Trajectory Optimization (CSVTO), an algorithm for performing trajectory optimization with constraints on a set of trajectories in parallel. We frame constrained trajectory optimization as a novel form of constrained functional minimization over trajectory distributions, which avoids treating the constraints as a penalty in the objective and allows us to generate diverse sets of constraint-satisfying trajectories. Our method uses Stein Variational Gradient Descent (SVGD) to find a set of particles that approximates a distribution over low-cost trajectories while obeying constraints. CSVTO is applicable to problems with differentiable equality and inequality constraints and includes a novel particle re-sampling step to escape local minima. By explicitly generating diverse sets of trajectories, CSVTO is better able to avoid poor local minima and is more robust to initialization. We demonstrate that CSVTO outperforms baselines in challenging highly-constrained tasks, such as a 7DoF wrench manipulation task, where CSVTO outperforms all baselines both in success and constraint satisfaction.

4.1 Introduction

Trajectory optimization and optimal control are powerful tools for synthesizing complex robot behavior using appropriate cost functions and constraints [17, 139, 56,

118, 18]. Constraint satisfaction is important for safety-critical applications, such as autonomous driving, where constraints determine which trajectories are safe. Constraints can also provide effective descriptions of desired behavior. For instance, consider a robot sanding a table. This problem can be defined with an equality constraint specifying that the end-effector must move along the surface of the table as well as constraints on the minimum and maximum force applied to the table. For many tasks, including manipulation tasks like the one above, satisfying these constraints can be very difficult as constraint-satisfying trajectories may lie on implicitly defined lower-dimensional manifolds. Such constraints present difficulties for sample-based methods since the feasible set has zero measure and thus it is difficult to sample. It is also difficult for gradient-based methods since even for trajectories that start feasible, if the constraint is highly nonlinear then updates based on a first-order approximation of the constraint will lead to solutions leaving the constraint manifold. In addition, many useful tasks entail constrained optimization problems that are non-convex and exhibit multiple local minima.

Global sample-based motion planning methods such as Rapidly-Exploring-Random-Trees (RRT) [79]. Probabilistic Roadmaps (PRM) [71] effectively solve difficult planning problems, however, they do not find paths that minimize a given cost function. To minimize a given cost function, algorithms such as RRT* and PRM* [70] have been proposed to find asymptotically globally optimal paths. Alternatively, a common approach is to use the path returned from a sample-based motion planner to initialize a trajectory optimization problem [87]. Sample-based methods have additionally been applied to constrained planning problems [14, 105, 58, 120], and kinodynamic problems [91, 165]. While effective for solving problems exhibiting local minima, when applied to kinodynamic or constrained problems these global methods are typically computationally expensive.

One of the key advantages of trajectory optimization techniques over global search methods, such as sampling-based motion planning, is computation speed. Faster computation speed enables online re-planning to adapt to disturbances. For example, consider again the robot sanding the table, but now in the proximity of a human. The human may move in an unexpected way which necessitates an update to the

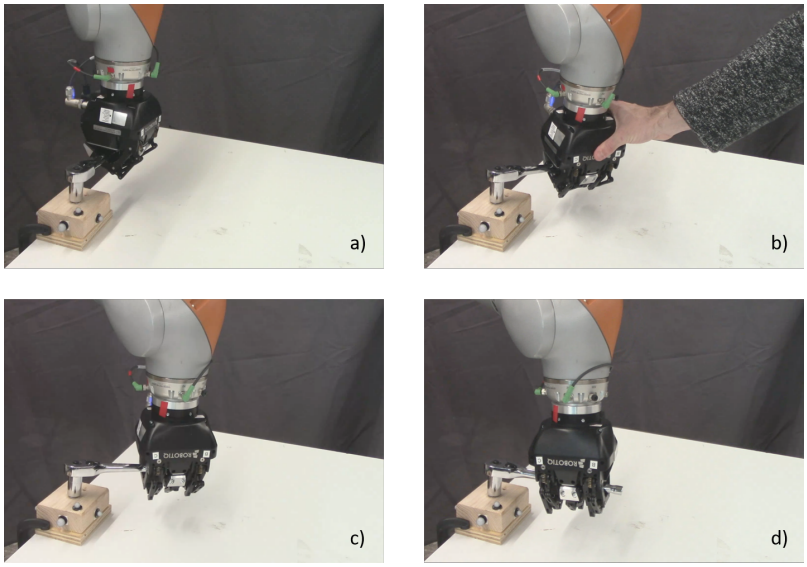


Figure 4.1: We use CSVTO to turn a wrench in the real world with online replanning; b) A human disturbs the robot, changing the grasp position of the wrench; c) The robot readjusts the grasp position; d) The robot achieves the desired wrench angle.

planned trajectory. However, even if the cost function is well-suited to the task, the performance of many trajectory optimization methods is still highly dependent on the initialization. Poor initialization may lead to the solver converging to a poor local minimum. For example, for a robot minimizing a distance to goal cost subject to collision constraints, this may mean a trajectory that avoids obstacles but makes little or no progress toward the goal. In the worst case, the solver may not find a feasible solution, in which case the robot may collide with an obstacle. A dependence on initialization is particularly problematic when re-solving the optimization problem online under limited computation time when disturbances can lead to the previous solution becoming a poor initialization for the current optimization problem. In the sanding example mentioned previously, the human may move to block the robot's path, and performing a local optimization starting from the previous trajectory may not return a feasible solution.

In this article, we formulate the constrained trajectory optimization problem as a Bayesian inference problem. This view has advantages as it aims to find a distribution over trajectories rather than a single trajectory alone. As noted by Lambert

et. al. [83], commonly used Variational Inference approaches [16] lead to minimizing entropy-regularized objectives [83] which can improve exploration of the search space and give greater robustness to initialization. Previous methods taking the inference view of trajectory optimization have only been able to incorporate constraints via penalties in the cost [176, 106, 83, 84]. A drawback of penalty methods is that selecting the relative weights of the penalties is challenging due to possible conflicts with the objective. We compare against baselines that incorporate constraints via penalties and show that, for non-trivial constraints, this results in poor constraint satisfaction. An alternative method for enforcing constraints in trajectory optimization is via barrier functions [51, 111]. While effective, they are only applicable to inequality constraints and have not yet been applied in the context of trajectory optimization as an inference problem.

We propose Constrained Stein Variational Trajectory Optimization (CSVTO), an algorithm that performs constrained trajectory optimization on a set of trajectories in parallel. Our method builds on Orthogonal-Space Stein Variational Gradient Descent (O-SVGD), a recent non-parametric variational inference method for domains with a single equality constraint [179]. We present a constrained SVGD algorithm for trajectory optimization with differentiable equality and inequality constraints, generating a diverse set of approximately constraint-satisfying trajectories. The trajectories are *approximately* constraint-satisfying because we do not run the algorithm until convergence to avoid excessive computation times. We additionally incorporate a novel re-sampling step that re-samples and perturbs particles in the tangent space of the constraints to escape local minima. Our contributions are as follows:

- We frame constrained trajectory optimization as a novel form of constrained functional minimization over trajectory distributions, which avoids treating the constraints as a penalty in the objective.
- We present a constrained SVGD algorithm for trajectory optimization, which is applicable to problems with differentiable equality and inequality constraints.
- We propose a novel particle re-sampling step for re-sampling and perturbing

trajectory particles in the tangent space of the constraints to escape local minima.

- We evaluate our method on three complex constrained problems, including a 12DoF underactuated quadrotor and two highly constrained 7DoF manipulation tasks.

Our experimental results demonstrate that CSVTO outperforms baselines in challenging, highly constrained tasks, such as a 7DoF wrench manipulation task where our method achieves 20/20 success compared with 12/20 for Interior Point OPTimizer (IPOPT) [157] and 19/20 for Stein Variational Model Predictive Control (SVMPC) [84], CSVTO also achieves the lowest constraint violation of all baselines. In addition, CSVTO outperforms baselines in a 12DoF quadrotor task with a dynamic obstacle that necessitates online adaption of the planned trajectory.

The chapter is organized as follows. In Section 4.2 we discuss related work. In Section 4.3 we will discuss the trajectory optimization problem, followed by an overview of the variational inference approach to trajectory optimization in Section 4.4. In Section 4.5 we introduce our novel formulation of trajectory optimization as a constrained functional minimization over trajectory distributions. We will then give some additional background information on SVGD in Section 4.6 which is necessary to develop our algorithm. In Section 4.7 we introduce CSVTO. In Section 4.8 we evaluate our method on a 12DoF quadrotor task and two highly constrained tasks with a 7DoF manipulator. We additionally deployed CSVTO to turn a wrench in the real world (Figure 4.1).

4.2 Related Work

4.2.1 Trajectory optimization

Previous work on local trajectory optimization techniques includes direct methods [6, 118], where the explicit optimization problem is transcribed and solved using nonlinear solvers such as IPOPT [157] or Sparse Nonlinear OPTimizer (SNOPT) [46]. Methods in this class include Sequential Convex Programming (SCP) methods such

as TrajOpt [139] and Guaranteed Sequential Trajectory Optimization (GuSTO) [17]. In contrast, indirect methods aim instead to solve the local optimality conditions of the trajectory and early examples include Differential Dynamic Programming (DDP) [101] and iterative Linear Quadratic Regulator (iLQR) [90], however, neither of these methods can handle constraints. Later work incorporated constraint satisfaction with these indirect methods [45, 171, 56]. Direct methods are typically easier to initialize but less accurate [156]. However all of these methods only aim to find a single locally-optimal trajectory, and the performance is dependent on the initialization. In contrast, our approach optimizes a diverse set of trajectories in parallel. This makes our approach easier to initialize as well as more robust to disturbances when re-planning online. Our approach is related to the direct methods, in that we use an iterative algorithm that aims to minimize an objective. However, our method is based on viewing the trajectory optimization problem as a Bayesian inference problem.

4.2.2 Sample-based Motion Planning

Many global search methods have been developed in the sampling-based motion planning literature, yielding motion planners for constrained domains. These can be broadly categorized as projection methods, whereby sampled configurations are projected to the constraint [14, 105], and continuation methods, which use a local approximation of the constraint manifold at feasible configurations to sample new configurations [58, 120]. Our method of trajectory optimization is similar to continuation methods, as our iterative algorithm projects update steps to the tangent space of the constraint. While these global motion planners can be highly effective, they are typically too computationally intensive to be run online.

4.2.3 Planning & Control as Inference

Prior work framing trajectory optimization as Bayesian inference has used Gaussian approximations to yield fast, gradient-based algorithms [106, 107, 151, 131, 164, 176]. Ha et al. presented a probabilistic approach for trajectory optimization with

constraints, using Laplace approximations around local minima found by solving a non-linear program (NLP) [48]. This approach uses a Gaussian approximation with a degenerate covariance with variance only in the tangent space of the constraints. Samples from this distribution will generally deviate from the constraint manifold for non-linear constraints, in contrast, our approach directly optimizes for diverse constraint-satisfying samples. Sample-based techniques such as Model Predictive Path Intregral (MPPI) control [167] and Cross-Entropy Method (CEM) [77] have strong connections to the inference formulation of Stochastic Optimal Control (SOC) [162], but these methods again use Gaussian sampling distributions. Several recent works have focused on improving the performance of these algorithms, often by modifying the sampling distribution. Watson and Peters recently proposed using a Gaussian Process as a sampling distribution [163], and Pinneri et al. proposed using colored noise [119], both of which lead to smoother sampled trajectories. Bhardwaj et al. [15] has also demonstrated improvements to MPPI with a focus on robot manipulation. However, in all of these prior works, the sampling distribution is uni-modal. Uni-modal sampling distributions can be problematic in complex environments due to their lack of flexibility which hinders exploration of the search space. Recent work has proposed learning non-Gaussian sampling distributions with flexible model classes [123, 136].

Another class of methods has used Stein Variational Gradient Descent (SVGD) [96] for Model Predictive Control (MPC) [84, 12] and trajectory optimization [83]. By using particle approximations these methods can generate multi-modal trajectory distributions. SVGD has also been used to improve Probabilistic Roadmaps (PRMs) [85], and for planning to goal sets [116]. Our method is also based on SVGD.

However, to date, control-as-inference-based methods have been unable to handle highly constrained domains. Recently Constrained Covariance Steering MPPI (CCSMPPI) [10] was proposed which can satisfy chance inequality constraints, but is restricted to linear systems. Our method uses SVGD to generate diverse sets of *constraint-satisfying* trajectories which can satisfy both inequality and equality constraints. Another method closely related to ours is Stochastic Multimodal Trajectory Optimization (SMTO) [114], this method treats the trajectory optimization

problem as a density estimation problem and alternates between sampling and performing a gradient-based optimization to generate multiple low-cost trajectories that satisfy the constraints. SMTO uses Covariant Hamiltonian Optimization for Motion Planning (CHOMP) [180] to perform the gradient-based optimization sequentially for each sampled trajectory. Our contribution is complementary to SMTO; SMTO could substitute CHOMP with our method, CSVTO, in the gradient-optimization step. This would have the advantage of performing the gradient-based optimization in parallel and encouraging diversity among trajectories.

4.2.4 Gradient Flows for constrained optimization

Our method is closely related to methods using gradient flows for constrained optimization. Gradient flows are an optimization method that re-frames optimization as the solution to an ordinary differential equation (ODE); gradient flows can be thought of as continuous-time versions of gradient descent algorithms. Yamashita proposed a gradient flow method for equality-constrained problems [173]. The most common method of extending this to problems with inequality constrained is via the introduction of slack variables to convert inequality constraints to equality constraints [138, 140, 64]. Our method, CSVTO, also uses slack variables to transform inequality constraints into equality constraints. Recently, Feppon et. al. [39] proposed a method that instead solves a Quadratic Program (QP) subproblem to identify active inequality constraints which are treated as equality constraints in the gradient flow. Jongen and Stein applied constrained gradient flows to global optimization, by proposing a gradient flow algorithm that iterates between searching for local minima and local maxima [64].

SVGD has been interpreted as a gradient flow [95], and similar ideas to those developed in the gradient flows for constrained optimization literature were recently explored in O-SVGD [179]. O-SVGD performs SVGD in domains with a single equality constraint. We extend and modify O-SVGD to domains with multiple equality and inequality constraints.

4.3 Trajectory Optimization

Trajectory optimization is commonly modeled as an Optimal Control Problem (OCP). We consider a discrete-time system with state $\mathbf{x} \in \mathbb{R}^{d_x}$ and control $\mathbf{u} \in \mathbb{R}^{d_u}$, where d_x and d_u are the dimensionality of the state and control, respectively, and dynamics $\mathbf{x}_t = f(\mathbf{x}_{t-1}, \mathbf{u}_{t-1})$. We define finite horizon trajectories with horizon T as $\tau = (\mathbf{X}, \mathbf{U})$, where $\mathbf{X} = \{\mathbf{x}_1, ...\mathbf{x}_T\}$ and $\mathbf{U} = \{\mathbf{u}_0, ...\mathbf{u}_{T-1}\}$. Given an initial state $\mathbf{x}_0$, the aim when solving an OCP is to find a trajectory τ that minimizes a given cost function C subject to equality and inequality constraints:

$$
\begin{aligned}
\min_{\tau} \quad & C(\tau) \\
\text{s.t.} \quad & \\
& h(\tau) = 0 \\
& g(\tau) \leq 0 \\
& \forall t \in \{1, \ldots, T\} \\
& f(\mathbf{x}_{t-1}, \mathbf{u}_{t-1}) = \mathbf{x}_t \\
& \mathbf{u}_{\min} \leq \mathbf{u}_{t-1} \leq \mathbf{u}_{\max} \\
& \mathbf{x}_{\min} \leq \mathbf{x}_t \leq \mathbf{x}_{\max}.
\end{aligned}
\tag{4.1}
$$

Here we have separated general inequality constraints g from simple bounds constraints, as well as the dynamics constraints from other equality constraints h. We additionally assume that C is non-negative and once differentiable and that f, g, h are all twice differentiable[1]. Problem (4.1) will be non-convex in general, therefore it is likely it will have multiple local minima. The quality of solutions for most methods for solving this optimal control problem depends heavily on the initialization; often a poor initialization can lead to infeasibility.

[1]We can also accommodate constraints that are only once-differentiable via an approximation (see Section 4.7.1.1

4.4 Variational Inference for Trajectory Optimization

In this section, we will demonstrate how unconstrained trajectory optimization can be framed as an inference problem, as in [131, 150, 112, 84]. This framing results in estimating a distribution over low-cost trajectories, rather than a single optimal trajectory. By using this framing we can leverage approximate inference tools for trajectory optimization, in particular, Variational Inference [16]. In this section, we will show how this framing leads to an entropy-regularized objective [83] which aims to find a distribution over low-cost trajectories while maximizing entropy. By using an entropy-regularized objective we aim to have improved exploration of the search space and greater robustness to initialization.

To reframe trajectory optimization as probabilistic inference, we first introduce an auxiliary binary random variable o for a trajectory such that

$$p(o = 1|\tau) = \exp\left(-\gamma C(\tau)\right), \tag{4.2}$$

which defines a valid probability distribution over o provided both γ and C are non-negative. We can trivially see that the trajectory that maximizes the likelihood of $p(o = 1|\tau)$ is the trajectory that minimizes the cost. Introducing this binary variable allows us to express the cost as a likelihood function, which we will use in the Bayesian inference formulation of trajectory optimization. Using this likelihood to perform inference gives us a principled way of computing a distribution over trajectories, where lower-cost trajectories have a higher likelihood. The term γ controls how peaked the likelihood function is around local maxima, or minima of C, which in turn controls the dispersion of the resulting trajectory distribution after performing inference.

We aim to find the posterior distribution over trajectories, conditioned on the value of auxiliary variable o. This is given by Bayes theorem as

$$p(\tau|o = 1) = \frac{p(o = 1|\tau)p(\tau)}{p(o = 1)}, \tag{4.3}$$

where $p(\tau) = p(\mathbf{X}, \mathbf{U})$ is a prior on trajectories. For deterministic dynamics, this prior is determined by placing a prior on controls $\mathbf{U}$. This prior is a design choice and can be used to regularize the controls. For instance, a squared control cost can be equivalently expressed as a Gaussian prior. Alternatively, this prior could be learned from a dataset of low-cost trajectories [154]. The trajectory prior is

$$p(\tau) = p(\mathbf{U}) \prod_{t=1}^{T} \delta(\mathbf{x}_t - \hat{\mathbf{x}}_t), \tag{4.4}$$

where $\hat{\mathbf{x}}_t = f(\mathbf{x}_{t-1}, \mathbf{u}_{t-1})$, and δ is the Dirac delta function. This inference problem can be performed exactly for the case of linear dynamics and quadratic costs [9, 164]. However, in general, this problem is intractable and approximate inference techniques must be used. We use variational inference to approximate $p(\tau|o = 1)$ with distribution $q(\tau)$ which minimizes the Kullback–Leibler divergence $\mathcal{KL}(q(\tau)||p(\tau|o = 1))$ [16]. The quantity to be minimized is

$$\begin{aligned} \mathcal{KL}\left(q(\tau)||p(\tau|o=1)\right) &= \int q(\tau) \log \frac{q(\tau)}{p(\tau|o=1)} d\tau \\ &= \int q(\tau) \log \frac{q(\tau)p(o=1)}{p(o=1|\tau)p(\tau)} d\tau. \end{aligned} \tag{4.5}$$

The $p(o = 1)$ term in the numerator does not depend on τ so can be dropped from the minimization. This results in the *variational free energy* $\mathcal{F}$:

$$\mathcal{F}(q) = \int q(\tau) \log \frac{q(\tau)}{p(o=1|\tau)p(\tau)} d\tau \tag{4.6}$$

$$= -\mathbb{E}_{q(\tau)}[\log p(o=1|\tau) + \log p(\tau)] - \mathcal{H}(q(\tau)) \tag{4.7}$$

$$= \mathbb{E}_{q(\tau)}[\gamma C(\tau)] - \mathbb{E}_{q(\tau)}[\log p(\tau)] - \mathcal{H}(q(\tau)), \tag{4.8}$$

where $\mathcal{H}(q(\tau))$ is the entropy of $q(\tau)$. Intuitively, we can understand that the first term promotes low-cost trajectories, the second is a regularization on the trajectory, and the entropy term prevents the variational posterior from collapsing to a *maximum a posteriori* (MAP) solution. We may choose to provide regularization on the controls

as part of C, in which case the prior term is absorbed into the cost term.

4.5 Problem Statement

In this article, we frame the constrained optimal control problem introduced in Section 4.3 as a probabilistic inference problem, using ideas developed in Section 4.4.

It is first instructive to consider the dynamics constraint, which is incorporated into the prior in equation (4.4) via the Dirac delta function. In this case, the term $\mathbb{E}_{q(\tau)}[-\log p(\tau)]$ is infinite for any τ which does not obey the dynamics constraint. We can convert this unconstrained optimization problem with infinite cost to the following constrained optimization problem on the space of probability distributions:

$$\begin{aligned} \min_{q} \quad & \tilde{\mathcal{F}}(q) \\ \text{s.t.} \quad & \\ & \forall t \in \{1, \dots, T\} \\ & P_q(f(\mathbf{x}_{t-1}, \mathbf{u}_{t-1}) = \mathbf{x}_t) = 1, \end{aligned} \tag{4.9}$$

where $\tilde{\mathcal{F}}(q)$ is the free energy from equation (4.8) with the infinite cost term $\sum_{t=1}^{T} \log \delta(\mathbf{x}_t - f(\mathbf{x}_{t-1}, \mathbf{u}_{t-1}))$ dropped from $\log p(\tau)$, and $P_q(A)$ is the probability of event A under

probability measure $q(\tau)$. Applying this process to other constraints we have

$$\begin{aligned}
&\min_{q} \quad \tilde{\mathcal{F}}(q) \\
&\text{s.t.} \\
&\quad P_q(h(\tau) = 0) = 1 \\
&\quad P_q(g(\tau) \leq 0) = 1 \\
&\quad \forall t \in \{1, \ldots, T\} \\
&\quad P_q(f(\mathbf{x}_{t-1}, \mathbf{u}_{t-1}) = \mathbf{x}_t) = 1 \\
&\quad P_q(\mathbf{u}_{\min} \leq \mathbf{u}_{t-1} \leq \mathbf{u}_{\max}) = 1 \\
&\quad P_q(\mathbf{x}_{\min} \leq \mathbf{x}_t \leq \mathbf{x}_{\max}) = 1.
\end{aligned} \tag{4.10}$$

Our goal is to solve the above optimization problem. However, for any practical algorithm, we cannot guarantee exact constraint satisfaction, both due to the potential non-convexity of the constraint functions and due to limited computation time. Computation time is especially limited in an online planning scenario. Therefore we will evaluate our method according to both the cost of the resulting trajectories and the amount of constraint violation when optimizing within a fixed number of iterations.

4.6 Stein Variational Gradient Descent

We develop an algorithm to solve the constrained variational inference objective in (4.10) based on Stein Variational Gradient Descent (SVGD) [96]. In this section we will give an overview of SVGD which forms the foundation of our method. SVGD is a variational inference technique that uses a non-parametric representation of the variational posterior. In our algorithm, we use SVGD to approximate the distribution $p(\tau|o = 1)$ with particles, where each particle is a trajectory. Consider the variational inference problem

$$q^*(\mathbf{x}) = \arg\min_{q(\mathbf{x})} \quad \mathcal{KL}\left(q(\mathbf{x})||p(\mathbf{x})\right), \tag{4.11}$$

where $\mathbf{x} \in \mathbb{R}^d$ and p and q are two probability density functions supported on $\mathbb{R}^d$. SVGD uses a particle representation of $q(\mathbf{x}) = \frac{1}{N}\sum_{i=1}^{N}\delta(\mathbf{x} - \mathbf{x}^i)$, and iteratively updates these particles in order to minimize $\mathcal{KL}\left(q(\mathbf{x})||p(\mathbf{x})\right)$. SVGD updates the particle set with the update equation

$$\mathbf{x}_{k+1}^i = \mathbf{x}_k^i + \epsilon\phi^*(\mathbf{x}_k^i), \tag{4.12}$$

where $\epsilon > 0$ is a step-size parameter. The update ϕ^* is computed using a differentiable positive definite kernel function $\mathcal{K}$ via

$$\phi^*(\mathbf{x}_k^i) = \frac{1}{N}\sum_{j=1}^{N}\mathcal{K}(\mathbf{x}_k^i, \mathbf{x}_k^j)\nabla_{\mathbf{x}_k^j}\log p(\mathbf{x}_k^j) + \nabla_{\mathbf{x}_k^j}\mathcal{K}(\mathbf{x}_k^i, \mathbf{x}_k^j). \tag{4.13}$$

The first term of this objective maximizes the log probability $p(\mathbf{x})$ for the particles, with particles sharing gradients according to their similarity defined by $\mathcal{K}$. The second term is a repulsive term that acts to push particles away from one another and prevents the particle set from collapsing to a local MAP solution.

We will now give further details on the derivation of the SVGD algorithm and demonstrate that it does indeed minimize $\mathcal{KL}\left(q(\mathbf{x})||p(\mathbf{x})\right)$. We will use the developments in this section to show that the fixed points of our algorithm satisfy first-order optimality conditions in section 4.7.1.3. SVGD is based on the *Kernelized Stein Discrepancy* (KSD) [97], which is a measure of the discrepancy between two distributions p and q. The KSD is computed as the result of the following constrained functional maximization

$$\mathbb{S}(p, q) = \max_{\phi \in \mathcal{H}^d}\left\{\mathbb{E}_{\mathbf{x}\sim q}[\mathcal{A}_p\phi(\mathbf{x})] \text{ s.t. } ||\phi||_{\mathcal{H}^d} \leq 1\right\}, \tag{4.14}$$

where $\phi : \mathbb{R}^d \to \mathbb{R}^d$ is a function in a vector-valued Reproducing Kernel Hilbert Space (RKHS) $\mathcal{H}^d$ with a scalar kernel $\mathcal{K} : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$. $\mathcal{A}_p$ is the Stein operator

$$\mathcal{A}_p\phi(\mathbf{x}) = \nabla_{\mathbf{x}}\log p(\mathbf{x})^T\phi(\mathbf{x}) + \nabla_{\mathbf{x}} \cdot \phi(\mathbf{x}), \tag{4.15}$$

where $\nabla_{\mathbf{x}} \cdot \phi(\mathbf{x}) = \sum_{k=1}^{d} \partial_{x_k} \phi_k(\mathbf{x})$. It was established in [97] that $\mathbb{S}(q, p) = 0 \iff p = q$ for a strictly positive-definite kernel $\mathcal{K}$. To minimize the KL divergence, SVGD considers the incremental transform $\mathbf{x}_\epsilon = \mathbf{x} + \epsilon\phi(\mathbf{x})$, where $\mathbf{x} \sim q(\mathbf{x})$ and ϵ is a scalar step-size parameter. The resulting distribution after applying the transform is $q_{[\epsilon\phi]}$. SVGD uses the following result:

$$\nabla_\epsilon \mathcal{K}\mathcal{L}(q_{[\epsilon\phi]} || p(\mathbf{x}))|_{\epsilon=0} = -\mathbb{E}_{\mathbf{x}\sim q}[\mathcal{A}_p \phi(\mathbf{x})], \tag{4.16}$$

which relates the Stein operator and the derivative of the KL divergence w.r.t the perturbation ϵ. We would like to select ϕ that maximally decreases the KL divergence. By considering $\phi \in \{\phi \in \mathcal{H}^d \,;\, ||\phi||_{\mathcal{H}^d} \leq 1\}$, the optimal ϕ is the solution to the following constrained functional maximization:

$$\phi^* = \arg\max_{\phi \in \mathcal{H}^d} \{-\nabla_\epsilon \mathcal{K}\mathcal{L}(q_{[\epsilon\phi]} || p(\mathbf{x}))|_{\epsilon=0}, \text{s.t.} ||\phi||_{\mathcal{H}^d} \leq 1\}. \tag{4.17}$$

This maximization has a closed-form solution, derived by Liu et al. in Theorem 3.8 of [97]. Note that we have used a slightly different definition of the Stein operator than that used by Liu et al., with $\mathcal{A}_p$ as defined in equation (4.15) as equal to the trace of the Stein operator defined in [97]. The closed-form solution is given by

$$\begin{aligned} \phi^*(\cdot) &= \mathbb{E}_{\mathbf{x}\sim q}[\mathcal{A}_p \mathcal{K}(\cdot, \mathbf{x})] && (4.18) \\ &= \mathbb{E}_{\mathbf{x}\sim q}[\mathcal{K}(\cdot, \mathbf{x}) \nabla_{\mathbf{x}} \log p(\mathbf{x}) + \nabla_{\mathbf{x}} \mathcal{K}(\cdot, \mathbf{x})], && (4.19) \end{aligned}$$

and the resulting gradient of the KL divergence is

$$\nabla_\epsilon \mathcal{K}\mathcal{L}(q_{[\epsilon\phi^*]} || p(\mathbf{x}))|_{\epsilon=0} = -\mathbb{S}(p, q). \tag{4.20}$$

This implies that for a suitably-chosen kernel $\mathcal{K}$, if the gradient of the KL divergence is zero then the KSD is also zero, which means that $p = q$. We finally arrive at the update rule given in equation (4.13) as the approximation of the closed-form solution in equation (4.19) with a finite set of particles.

4.6.1 Orthogonal-Space Stein Variational Gradient Descent

Recently Zhang et. al. proposed O-SVGD, a method for performing SVGD with a single equality constraint [179], though they do not consider the problem of trajectory optimization. In this section, we give an overview of O-SVGD, but we give an alternative derivation to that given in [179] based on vector-valued RKHS and matrix-valued kernels [161]. This alternative derivation will allow us to analyze our algorithm (Section 4.7.1.3). The problem [179] aims to solve is

$$\min_q \mathcal{KL}(q(\mathbf{x})||p(\mathbf{x})) \quad \text{s.t.} \quad P_q(h(\mathbf{x}) = 0) = 1, \tag{4.21}$$

where h represents a single equality constraint. For particles $\mathbf{x}$ that are on the manifold induced by $h(\mathbf{x}) = 0$, we would like them to remain on the manifold after applying the Stein update in equation (4.12). To do this, we replace the function $\phi(\mathbf{x})$ with $P(\mathbf{x})\phi(\mathbf{x})$. Where $P(\mathbf{x})$ projects the updates to be in the tangent space of the constraint and is given by

$$P(\mathbf{x}) = I - \frac{\nabla h(\mathbf{x}) \nabla h(\mathbf{x})^T}{||\nabla(h(\mathbf{x})||^2}. \tag{4.22}$$

We can develop an SVGD algorithm that updates particles on the constraint manifold by considering the set of functions $\{P(\mathbf{x})\phi(\mathbf{x}),\ \phi(\mathbf{x}) \in \mathcal{H}^d\}$. By applying Lemma 2 from [161] we establish that this set of functions is an RKHS $\mathcal{H}_\perp^d$ with matrix-valued kernel $\mathcal{K}_\perp$ given by

$$\mathcal{K}_\perp(\mathbf{x}^i, \mathbf{x}^j) = P(\mathbf{x}^i)\mathcal{K}(\mathbf{x}^i, \mathbf{x}^j)P(\mathbf{x}^j)^T \tag{4.23}$$

$$= \mathcal{K}(\mathbf{x}^i, \mathbf{x}^j)P(\mathbf{x}^i)P(\mathbf{x}^j), \tag{4.24}$$

where we have used the fact that $\mathcal{K}$ is a scalar function and that $P(x)$ is symmetric to rearrange. Running SVGD with kernel $\mathcal{K}_\perp$ will therefore solve the constrained minimization problem (4.17), maximally reducing the KL divergence while only considering updates that lie in the tangent space of the constraint. Zhang et. al. [179] also add a term to equation (4.12) that drives particles to the manifold induced by

the constraint

$$\phi_C = -\frac{\psi(h(\mathbf{x}))\nabla h(\mathbf{x})}{||\nabla h(x)||^2}, \tag{4.25}$$

where ψ is an increasing odd function.

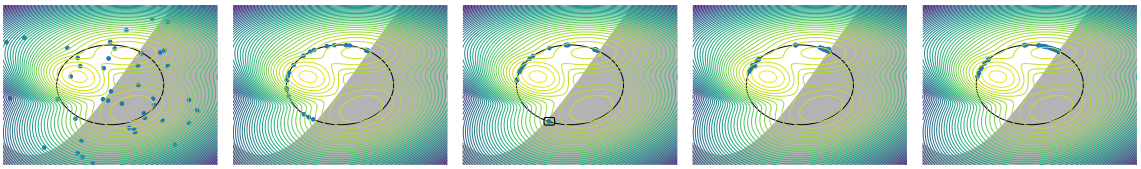
4.7 Methods

Our proposed trajectory optimization algorithm uses SVGD to perform constrained optimization on a set of trajectories in parallel. The result is a diverse set of low-cost constraint-satisfying trajectories. The full algorithm is shown in Algorithm 6. First, we will introduce the main component of our proposed algorithm, which decomposes the Stein update into a step tangential to the constraint boundary, and a step toward constraint satisfaction. We will then provide an analysis of the algorithm which relates it to problem (4.10). Finally, we will discuss strategies for improving performance which include separating the bounds constraints, an annealing strategy for increasing particle diversity, and re-sampling particles during the optimization. Figure 4.2 demonstrates CSVTO being applied to a 2D toy problem.

4.7.1 Constrained Stein Trajectory Optimization

Solving the constrained variational inference problem in (4.10) is very difficult, since it requires finding a distribution that may exhibit multi-modality and has constrained support. To address this, we use a non-parametric representation of the distribution $q(\tau)$. We use SVGD where each particle is a trajectory, and iteratively update the particle set while enforcing the constraints on each particle. To do this we extend O-SVGD to multiple equality and inequality constraints and use it to generate constraint-satisfying trajectories.

First, we relate using SVGD for unconstrained trajectory optimization to the minimization of the unconstrained variational free energy $\mathcal{F}(q)$ from (4.7). Consider the iterative transform $\tau_\epsilon = \tau + \epsilon\phi^*(\tau)$, where ϕ^* is the solution to (4.17) with posterior log likelihood $\log p(\tau|o = 1)$, $\tau \sim q(\tau)$ and $\tau_\epsilon \sim q_{[\epsilon\phi^*]}(\tau)$. We can recast



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

Figure 4.2: CSVTO visualized for a 2D problem. The posterior is a mixture of 3 Gaussians, with the log posterior peaks visualized. There is an equality constraint that the particles must lie on the circle. There is also an inequality constraint that the particles must lie outside the shaded region. a) The initial particles are randomly generated and are not necessarily feasible. b) Due to the annealing discussed in section 4.7.1.4, early on in the optimization the particles are constraint-satisfying and diverse. c) The particles move towards the relative peaks of the objective, however, the circled particle has become stuck in a poor local minimum due to the constraints, where the gradient of the log posterior is directed towards an infeasible peak. Since the particle is isolated it is not sufficiently affected by the repulsive gradient term that would help escape the local minimum. d) The re-sampling step from section 4.7.1.8 re-samples the particles, applying noise in the tangent space of the constraints. This eliminates the particle at the poor local minimum. e) The set of particles converges around the local minimum of the objectives while satisfying the constraints.

(4.17) for trajectories in terms of the free energy $\mathcal{F}(q)$

$$\phi^*(\tau) = \arg\max_{\phi \in \mathcal{H}^d} \{-\nabla_\epsilon \mathcal{F}(q_{[\epsilon\phi]})|_{\epsilon=0}, \text{s.t.} ||\phi||_{\mathcal{H}^d} \leq 1\}. \tag{4.26}$$

Thus the update $\tau + \epsilon\phi^*$ ensures we maximally decrease the variational free energy. If $\phi^*(\tau) = 0$ then $q(\tau)$ is at a local minimum of $\mathcal{F}(q)$. We will now modify the Stein update to account for constraints.

4.7.1.1 Equality constraints

We propose a modified Stein update rule for the i-th particle, in which we decompose the update into two components:

$$\tau_{k+1}^i = \tau_k^i + \alpha_J \phi_\perp(\tau_k^i) + \alpha_C \phi_C(\tau_k^i), \tag{4.27}$$

where $\phi_\perp$ is an update that is tangential to the constraint boundary, ϕ_C acts in the direction which decreases constraint violation, α_J and α_C are scalar step size parameters, and k is the iteration. We replace the O-SVGD ϕ_C from equation (4.25) with a Gauss-Newton step to minimize $h(\tau)^T h(\tau)$

$$\phi_C(\tau) = \nabla h(\tau)^T (\nabla h(\tau) \nabla h(\tau)^T)^{-1} \, h(\tau). \tag{4.28}$$

This uses approximate second-order curvature information for fast convergence. We then compute the *projection* matrix $P(\tau)$, which projects vectors onto the tangent space of the constraints as

$$P(\tau) = I - \nabla h(\tau)^T (\nabla h(\tau) \nabla h(\tau)^T)^{-1} \nabla h(\tau). \tag{4.29}$$

Inverting $\nabla h(\tau) \nabla h(\tau)^T$ is only possible if $\nabla h(\tau)$ is full rank. When $\nabla h(\tau)$ is not full rank, we compute the pseudo-inverse of $\nabla h(\tau) \nabla h(\tau)^T$ via the singular-value decomposition, discarding singular vectors corresponding to singular values that are

zero. Once we have $P(\tau)$, we use it to define the tangent space kernel, as in [179]:

$$\mathcal{K}_\perp(\tau^i, \tau^j) = \mathcal{K}(\tau^i, \tau^j)P(\tau^i)P(\tau^j). \tag{4.30}$$

We then use this kernel for the SVGD update to produce an update that is in the tangent space of the constraint:

$$\begin{aligned}\phi^*_\perp(\tau^i) =& \frac{1}{N}\sum_{j=1}^{N} \mathcal{K}_\perp(\tau^i, \tau^j)\nabla_{\tau^j} \log p(\tau^j|o=1) \\ &+ \nabla_{\tau^j}\mathcal{K}_\perp(\tau^i, \tau^j).\end{aligned} \tag{4.31}$$

Since $\mathcal{K}_\perp$ is a matrix-valued function, the last term is calculated (as in [161]) as

$$[\nabla_{\tau^j}\mathcal{K}_\perp(\tau^i, \tau^j)]_l = \sum_m \nabla_{[\tau^j]_m}[\mathcal{K}_\perp(\tau^i, \tau^j)]_{l,m}, \tag{4.32}$$

where the notation $[x]_l$ indicates the lth element of x. Equation (4.31) has several interesting features. First, two trajectory particles τ^i and τ^j are considered close if they are close according to the original kernel $\mathcal{K}$. In addition, expanding the first term to $\mathcal{K}(\tau^i, \tau^j)P(\tau^i)P(\tau^j)\nabla_{\tau^j} \log p(\tau^j|o=1)$, we see that if $P(\tau^i) = P(\tau^j)$ this reduces to $\mathcal{K}(\tau^i, \tau^j)P(\tau^j)\nabla_{\tau^j} \log p(\tau^j|o=1)$. For $P(\tau^i) \neq P(\tau^j)$, the magnitude of this term will always be reduced. Intuitively this means that particles will share gradients if particles are close and the tangent space of the constraint is similar. In addition, all updates will be in the tangent space of the constraint.

Repulsive term in the tangent space The derivative $\nabla_{[\tau^j]_m}[\mathcal{K}_\perp(\tau^i, \tau^j)]_{l,m}$ can be expanded to

$$\begin{aligned}\nabla_{[\tau^j]_m}[\mathcal{K}_\perp(\tau^i, \tau^j)]_{l,m} &= \nabla_{[\tau^j]_m}[\mathcal{K}(\tau^i, \tau^j)P(\tau^i)P(\tau^j)]_{l,m} \\ &= [P(\tau^i)P(\tau^j)]_{l,m}\nabla_{[\tau^j]_m}\mathcal{K}(\tau^i, \tau^j) + \\ &\quad \mathcal{K}(\tau^i, \tau^j)[P(\tau^i)]_{l,m}\nabla_{[\tau^j]_m}[P(\tau^j)]_{l,m}.\end{aligned} \tag{4.33}$$

We see from equation (4.33) above that the gradient of the kernel consists of two terms. The first term projects the gradient of the unconstrained kernel to the tangent space of the constraints both at τ^i and τ^j.

The second term requires computing the derivative of the matrix-valued projection function. This term is expanded further in Appendix B.1, showing that it requires the evaluation of the second derivative of the constraint function $\nabla^2 h(\tau)$. For problems with constraints for which the second derivative is unavailable, we can remove this second term for individual constraints. We do this by setting the second derivative of a particular constraint to be the zero matrix (see Appendix B.1). Doing so effectively uses a locally linear approximation of the constraint to compute the repulsive gradient.

We will discuss how we define a kernel on trajectories in section 4.7.1.5.

4.7.1.2 Extension to Inequality Constraints

We extend the above method to inequality constraints with the use of slack variables. We turn the inequality constraints into equality constraints with slack variable $\mathbf{z}$:

$$g(\tau) + \frac{1}{2}\mathbf{z}^2 = 0. \tag{4.34}$$

The full set of equality constraints then becomes

$$\hat{h} = \begin{bmatrix} h(\tau) \\ g(\tau) + \frac{1}{2}\mathbf{z}^2 \end{bmatrix}. \tag{4.35}$$

Converting inequality constraints to equality constraints via squared slack variables is often avoided as it can introduce spurious non-local-minima that satisfy the Karush–Kuhn–Tucker (KKT) conditions [7]. To mitigate this issue we make an assumption on the regularity of the problem, denoted as (R) in [138]. The details of the assumption are technical and we do not include it here. The assumption essentially states that $\nabla\hat{h}$ is full rank at initialization and remains so during the optimization. Under these assumptions, Schropp [138] proved that the hyperbolic equilibrium points of the augmented system are local minima of the equality and

inequality-constrained optimization problem. Optimizing multiple trajectories in parallel provides additional robustness against this issue. Even should some particles become stuck at one of these undesirable fixed points, in Section 4.7.1.8 we propose a method for re-sampling the set of particles which redistributes particles away from these fixed points. While we could avoid this issue by using non-negative slack variables with the transformation $g(\tau) + \mathbf{z}$, where $\mathbf{z} > 0$, we found that this led to poorer constraint satisfaction in practice.

After introducing the slack variables, we compute the constrained Stein update with all constraints as equality constraints. We augment the state with $\mathbf{z}$ as

$$\hat{\tau} = \begin{bmatrix} \tau \\ \mathbf{z} \end{bmatrix}. \tag{4.36}$$

The projection is given by

$$P(\hat{\tau}) = I - \nabla\hat{h}(\hat{\tau})^T(\nabla\hat{h}(\hat{\tau})\nabla\hat{h}(\hat{\tau})^T)^{-1}\nabla\hat{h}(\hat{\tau}), \tag{4.37}$$

and the kernel is

$$\mathcal{K}_\perp(\hat{\tau}^i, \hat{\tau}^j) = \mathcal{K}(\tau^i, \tau^j)P(\hat{\tau}^i)P(\hat{\tau}^j). \tag{4.38}$$

Notice that the kernel uses the original τ and not the augmented $\hat{\tau}$. We then perform the constrained Stein update on the augmented state:

$$\begin{aligned} \phi^*_\perp(\hat{\tau}^i) =& \frac{1}{N}\sum_{j=1}^{N} \mathcal{K}_\perp(\hat{\tau}^i, \hat{\tau}^j) \begin{bmatrix} \nabla \log p(\tau^j | o = 1) \\ 0 \end{bmatrix} \\ &+ \nabla_{\hat{\tau}^j}\mathcal{K}_\perp(\hat{\tau}^i, \hat{\tau}^j) \end{aligned} \tag{4.39}$$

$$\phi_C(\hat{\tau}) = \nabla\hat{h}(\hat{\tau})^T(\nabla\hat{h}(\hat{\tau})\nabla\hat{h}(\hat{\tau})^T)^{-1}\,\hat{h}(\hat{\tau}). \tag{4.40}$$

Once we have performed the iterative optimization we have a set of trajectories. We then select a trajectory to execute by choosing the one that minimizes the penalty function

$$\hat{C}_\lambda(\hat{\tau}) = C(\tau) + \lambda\sum|\hat{h}(\hat{\tau})|. \tag{4.41}$$

4.7.1.3 Analysis

In this section, we provide an analysis of CSVTO. We demonstrate that stationary points of the gradient flow satisfy the first-order optimality conditions for the constrained variational optimization problem in (4.10), subject only to equality constraints.

Theorem 1. *Assume that ∇h is full rank. Let $\phi^* \in \mathcal{H}^d$ be the solution to (4.17) with the unconstrained kernel $\mathcal{K}$, and $\phi^*_\perp \in \mathcal{H}^d_\perp$ be the solution to (4.17) using the tangent space kernel $\mathcal{K}_\perp$. If the following holds:*

$$\alpha_J \phi^*_\perp(\tau) + \alpha_C \phi_C(\tau) = 0, \tag{4.42}$$

then the following must be true:

$$\phi^*(\tau) + \nabla h(\tau)^T \mu = 0 \tag{4.43}$$

$$h(\tau) = 0, \tag{4.44}$$

where μ is a vector of Lagrange multipliers.

Proof. Since ϕ_C and $\phi^*_\perp$ are orthogonal, then if equation (4.42) holds then $\phi_C = \phi^*_\perp = 0$. Next, we note that $\phi^*_\perp(\tau) = P(\tau)\hat{\phi}$, where $\hat{\phi} \in \mathcal{H}^d$ and further $P(\tau)\hat{\phi}(\tau) = 0 \implies P(\tau)\phi^*(\tau) = 0$. To see this, consider $P(\tau)\phi^*(\tau) \neq 0$. This would imply that $\nabla_\epsilon \mathcal{KL}(q_{[\epsilon P \phi^*]} || p(\tau | o = 1))|_{\epsilon=0} \neq 0$, which implies that there is a descent direction. This would mean that $\exists\, \phi_\perp$ such that $-\nabla_\epsilon \mathcal{KL}(q_{[\epsilon \phi^*_\perp]} || p(\tau | o = 1))|_{\epsilon=0} < -\nabla_\epsilon \mathcal{KL}(q_{[\epsilon \phi_\perp} || p(\tau | o = 1))_{\epsilon=0}$, which is a contradiction. Expanding $P(\tau)\phi^* = 0$ yields

$$\begin{aligned} \left[I - \nabla h(\tau)^T \left(\nabla h(\tau) \nabla h(\tau)^T \right)^{-1} \nabla h(\tau) \right] \phi^*(\tau) = 0 \\ \phi^*(\tau) - \nabla h(\tau)^T \left[\left(\nabla h(\tau) \nabla h(\tau)^T \right)^{-1} \nabla h(\tau) \phi^*(\tau) \right] = 0. \end{aligned} \tag{4.45}$$

Specifying $\mu = -\left(\nabla h(\tau) \nabla h(\tau)^T \right)^{-1} \nabla h(\tau) \phi^*(\tau)$ results in equation (4.43) being sat-

isfied. Now we expand $\phi_C = 0$ resulting in

$$\nabla h(\tau)^T (\nabla h(\tau) \nabla h(\tau)^T)^{-1} h(\tau) = 0. \tag{4.46}$$

To show feasibility at the stationary point we left multiply (4.46) by $\nabla h(\tau)$, which for full rank ∇h results in $h(\tau) = 0$. □

Theorem 1 holds when we can integrate the expectation in (4.19). However, we are approximating the expectation with particles so (4.43) may not hold in practice. However, the feasibility condition (4.44) remains true when using a particle approximation for q. To extend this proof to inequality constraints, note that in Section 4.7.1.2 we discussed the regularity conditions under which hyperbolic stable stationary points of the gradient flow on the augmented equality-constrained system satisfy first-order optimality conditions of the original system with both equality and inequality constraints.

4.7.1.4 Annealed SVGD for improved diversity

We employ an annealing technique for SVGD as proposed in [30]. We use a parameter $\gamma \in [0, 1]$ which controls the trade-off between the gradient of the posterior log-likelihood and the repulsive gradient. For $\gamma << 1$ the repulsive term dominates resulting in trajectories being strongly forced away from one another. As γ increases the gradient of the posterior likelihood has a greater effect resulting in trajectories being optimized to decrease the cost. When combined with ϕ_C this results in the optimization prioritizing diverse constraint-satisfying trajectories first, then decreasing cost later in the optimization. The annealed update is given by

$$\begin{aligned} \phi_\perp^i(\hat{\tau}) = & \frac{1}{N} \sum_{j=1}^{N} \gamma \mathcal{K}_\perp(\hat{\tau}_i, \hat{\tau}_j) \begin{bmatrix} \nabla \log p(\tau_j | o) \\ 0 \end{bmatrix} \\ & + \nabla_{\hat{\tau}_j} \mathcal{K}_\perp(\hat{\tau}_i, \hat{\tau}_j). \end{aligned} \tag{4.47}$$

We use a linear annealing schedule with $\gamma_k = \frac{k}{K}$, where K is the total number of iterations. When performing online re-planning, we only perform the annealing when

optimizing the trajectory the first time-step.

4.7.1.5 Trajectory Kernel

CSVTO relies on a base kernel $\mathcal{K}(\tau^i, \tau^j)$ operating on pairs of trajectories which defines the similarity between trajectories. As noted by Lambert et. al. [84], high dimensional spaces can result in diminishing repulsive forces, which can be problematic for trajectory optimization problems due to the time horizon. We use a similar approach to SVMPC [84] in that we decompose the kernel into the sum of kernels operating on smaller components of the trajectory. We use a sliding window approach to decompose the trajectory. For a given sliding window length W let $\tau^t = [x_{t:t+W}, u_{t-1:t-1+W}]^T$. The overall kernel is then given by

$$\mathcal{K}(\tau^i, \tau^j) = \frac{1}{T-W} \sum_{t}^{T-W} \mathcal{K}(\tau_t^i, \tau_t^j). \tag{4.48}$$

We use the Radial-Basis Function (RBF) kernel $\mathcal{K}(\tau_t^i, \tau_t^j) = \exp(-\frac{1}{h}||\tau_t^i - \tau_t^j||_2^2)$ as the base kernel, where h is the kernel bandwidth. We use the median heuristic as in [96] to select the kernel bandwidth:

$$h = \frac{\texttt{median}(||\tau_t^i - \tau_t^j||_2)^2}{\log(N)}, \tag{4.49}$$

where N is the number of particles.

4.7.1.6 Bounds constraint

Bounds constraints can, in principle, be handled as general inequality constraints as described in the above section. However, since this involves adding additional slack variables incorporating bounds constraints involves an additional $T \times 2(d_u + d_x)$ decision variables in the optimization, where d_x and d_u are the state and control dimensionalities, respectively. It is more computationally convenient to use a simple approach where at every iteration we directly project the trajectory to satisfy the

bound constraints. This is done by

$$\tau^* = \min(\max(\tau_{min}, \tau), \tau_{max}). \tag{4.50}$$

4.7.1.7 Initialization

As introduced in section 4.4, we have a user-specified prior on controls $p(U)$. To initialize CSVTO on a new problem, we proceed by sampling from this prior $p(U)$ and using the dynamics $f(x_t, u_t)$ to generate sampled trajectories. In this way, we ensure that the initial trajectory satisfies the dynamics constraints.

We use a different initialization scheme when running trajectory optimization online in a receding horizon fashion as in Algorithm 7, as it is typical to warm-start the optimization with the solution from the previous timestep. For a single particle, the trajectory consists of $\tau = (\mathbf{x}_1, ..., \mathbf{x}_T, \mathbf{u}_0, ..., \mathbf{u}_{T-1)})$. The shift operation computes $\tau' = (\mathbf{x}_2, ..., \mathbf{x}'_{T+1}, \mathbf{u}_1, ..., \mathbf{u}'_T)$. Here $\mathbf{x}'_{T+1}, \mathbf{u}'_T$ is the initialization for the newly considered future timestep. The initializations $\mathbf{x}'_{T+1}, \mathbf{u}'_T$ may be chosen in a problem-specific way. In our approach, we choose them by duplicating the previous timestep's state and control, i.e. $(\mathbf{x}'_{T+1}, \mathbf{u}'_T) = (\mathbf{x}_T, \mathbf{u}_{T-1})$.

When running the algorithm with inequality constraints, for both the online and warm-start optimizations we initialize the slack variable $\mathbf{z}$ with $\mathbf{z} = \sqrt{2|g(\tau)|}$ so that trajectories satisfying the inequality constraint are initialized to satisfy the transformed equality constraint.

The above heuristic is motivated by the assumption that the solution should not vary much between timesteps. However, the fact that we have a set of trajectories rather than a single one can invalidate this assumption, since we can only take a single action. Trajectories that have very different first actions from the action taken can end up being quite poor initializations, particularly in the presence of constraints that can render them infeasible. Over time these poor initializations can lead to the degradation in the quality of the particles, which motivates the next section in which we discuss a re-sampling technique to prevent sample impoverishment.

4.7.1.8 Re-sampling

As discussed above, the shift operation can lead to trajectories that are not executed becoming infeasible and rendering those particles useless for trajectory optimization. In addition, our cost and constraints are not necessarily convex, so, as with any local optimization method, poor initializations can lead to infeasibility. We take inspiration from the Particle Filter literature [54] and incorporate a re-sampling step which is executed when performing online re-planning. Every `resample_steps` timesteps we re-sample after performing the shift operation on the previous trajectory particles. To perform re-sampling, we compute weights using the penalty function

$$w_i = \frac{\exp(-\frac{\hat{C}_\lambda(\hat{\tau}_i)}{\beta})}{\sum_j^N \exp(-\frac{\hat{C}_\lambda(\hat{\tau}_j)}{\beta})}, \tag{4.51}$$

where β is a temperature parameter. We then re-sample a new set of particles according to weights w_i. It is common in the particle filter literature to additionally add noise, to prevent re-sampled particles collapsing. However, in our case, it is undesirable to add random noise to a constraint-satisfying trajectory as it may lead to constraint violation. We avoid this issue by sampling noise and projecting the noise to only have components in the tangent space of the constraints for a given trajectory. Suppose we have sampled trajectory τ_i from the set of particles. We first sample $\epsilon \sim \mathcal{N}(0, \sigma^2_{resample} I)$, and then update the trajectory with

$$\tau_{new} = \tau_i + P(\tau_i)\epsilon, \tag{4.52}$$

where $P(\tau_i)$ is the projection matrix from equation (4.29).

4.8 Results

We evaluate our approach in three experiments. The first is a constrained 12DoF quadrotor task which has nonlinear underactuated dynamics. The second experiment is a 7DoF robot manipulator task, where the aim is to move the robot end-effector to

Algorithm 6 A single step of CSVTO, this will run every timestep.

function CSVTO($\mathbf{x}_0, \tau, K,$ `anneal`)
  $\mathbf{z} \leftarrow \sqrt{2|g(\tau)|}$
  $\hat{\tau} \leftarrow [\tau, \mathbf{z}]^T$
  for k $\in \{1, ..., \text{K}\}$ **do**
    for i $\in \{i, ..., \text{N}\}$ **do**
      $\phi_C^i \leftarrow$ via eq. 4.40
      if `anneal` **then**
        $\gamma \leftarrow \frac{k}{K}$
        $\phi_\perp^i \leftarrow$ via eq. 4.47
      else
        $\phi_\perp^i \leftarrow$ via eq. 4.39
      $\hat{\tau}^i \leftarrow \hat{\tau}^i + \alpha_J \phi_\perp^i + \alpha_C \phi_C^i$
      $\hat{\tau}^i \leftarrow$ ProjectInBounds($\hat{\tau}^i$).
  ▷ Get the best trajectory according to penalty function
  $\hat{\tau}^* \leftarrow \arg\min_\tau \hat{C}_\lambda(\hat{\tau})$
  ▷ Discard slack variables
  $\tau^*, \tau \leftarrow \hat{\tau}^*, \hat{\tau}$
  return τ^*, τ

Algorithm 7 CSVTO running with online re-planning

function CSVTO_MPC($\mathbf{x}_0, \tau_0$)
  for t $\in \{1, ..., \text{T}\}$ **do**
    ▷ Resample
    if MOD(t, `resample_steps`) $= 0$ **then**
      $\tau_t \leftarrow$ RESAMPLE(τ_t)
    if t $= 1$ **then**
      $K \leftarrow K_w$
      `anneal` $\leftarrow$ `True`
    else
      $K \leftarrow K_o$
      `anneal` $\leftarrow$ `False`
    $\tau_t^*, \tau_t \leftarrow$ CSVTO($x_t, \tau_t, K,$ `anneal`)
    ▷ Select first control from the best trajectory
    $\mathbf{u}_t \leftarrow \tau*$
    $\mathbf{x}_t \leftarrow$ STEPENV($\mathbf{u}_t$)
    ▷ Shift operation
    $\tau_{t+1} \leftarrow$ SHIFT(τ_t)

a goal location while being constrained to move along the surface of a table. The third experiment is also a 7DoF robot manipulator task, where the aim is to manipulate a wrench to a goal angle. Both of these 7DoF manipulator tasks involve planning in highly constrained domains. We perform the manipulator experiments in IsaacGym [99]. The hyperparameters we use in all experiments are shown in Table 4.1. For all experiments, the costs and constraint functions are written using PyTorch [115], and automatic differentiation is used to evaluate all relevant first and second derivatives.

4.8.1 Baselines

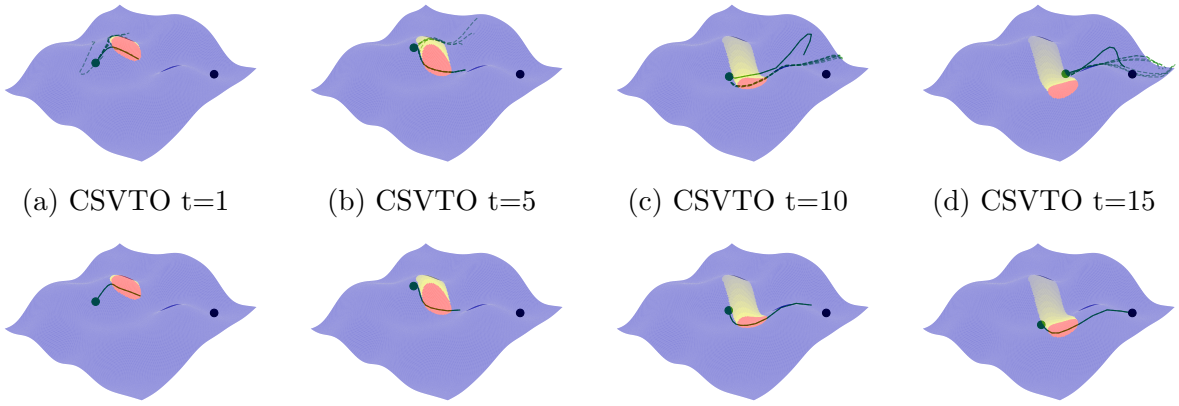
We compare our trajectory optimization approach to both sampling-based and gradient-based methods. We compare against IPOPT [157], a general non-linear constrained optimization solver, which has been widely used for robot trajectory optimization [6, 118]. We use the MUMPS [3] linear solver for IPOPT. When running IPOPT, where second derivatives are available we use exact derivatives computed via automatic differentiation in PyTorch, where they are not available we use the

Limited-memory Broyden–Fletcher–Goldfarb–Shanno algorithm (L-BFGS) [93] to approximate the Hessian. The method used will be indicated for each experiment. For CSVTO and IPOPT, we use a direct transcription scheme; IPOPT solves the optimization problem as expressed in problem (4.1). For IPOPT we use the open-source implementation provided by [157].

We additionally compare against MPPI [167] and SVMPC [84]. MPPI and SVMPC are methods for performing unconstrained trajectory optimization, with constraints commonly incorporated with penalties. For these methods we use the penalty function $\hat{C}_{(\lambda,\mu)}(\tau) = C(\tau) + \lambda \sum |h(\tau)| + \mu \sum |g(\tau)|^+$, where $|g(\tau)|^+$ is a vector consisting of only the positive values of $g(\tau)$. We use separate penalty weights for equality and inequality constraints. We evaluate each of these baselines on two different magnitudes of penalty weights on equality constraints λ. In the SVMPC paper, the authors show that their method can be used both with and without gradients. We evaluate against two versions of SVMPC, one using a sample-based approximation to the gradient and another using the true gradient. For SVMPC and MPPI we use a shooting scheme since they can only handle constraints via penalties, which can lead to poor satisfaction of the dynamics constraint. We use our own implementations for MPPI and SVMPC in PyTorch.

4.8.2 12DoF Quadrotor

We evaluate our method on a 12DoF underactuated quadrotor problem. The goal is to navigate the quadrotor from a start state to a goal state. We chose this problem to demonstrate our approach on a problem with complex nonlinear underactuated dynamics. The experimental setup is shown in Figures 4.3 and 4.4. The state of the quadrotor is $\mathbf{x} = [x, y, z, p, q, r, \dot{x}, \dot{y}, \dot{z}, \dot{p}, \dot{q}, \dot{r}]^T$, where (x, y, z) is the 3D position and (p, q, r) are the Euler angles. The control is the thrust $\mathbf{u} = [u_1, u_2, u_3, u_4]^T \in \mathbb{R}^4$. We place bounds constraints on the (x, y) location of the quadrotor to be within a $10m \times 10m$ area centred at $(0, 0)$. The goal is to travel from start locations sampled uniformly from $x, y \in [3.0m, 4.5m]$ to a goal location of $(4, 4)$ within 100 time steps. We place an equality constraint that the quadrotor must travel along a nonlinear



(a) CSVTO t=1 (b) CSVTO t=5 (c) CSVTO t=10 (d) CSVTO t=15

(e) IPOPT t=1 (f) IPOPT t=5 (g) IPOPT t=10 (h) IPOPT t=15

Figure 4.3: Experimental setup for the quadrotor task. The quadrotor must travel to the goal location, avoiding the obstacle in red while remaining on the blue manifold. The fading yellow shows the path of the obstacle from previous timesteps. a-d) CSVTO maintains a set of trajectories (dashed), with the selected trajectory shown as a solid curve. CSVTO can keep a diverse set of trajectories and switches between them to avoid the moving obstacle. e-f) IPOPT generates an initial trajectory that makes good progress toward the goal and obeys the manifold constraint. However, even after the first timestep the obstacle has moved to render this trajectory infeasible. As the obstacle moves further IPOPT is unable to find an alternative trajectory and ends in a collision.

surface $z = f_{surf}(x, y)$. For this surface, we sample z values from a Gaussian Process (GP) prior with an RBF kernel and zero mean function on a 10×10 grid of (x, y) points. We use the sampled values as observations for a GP with the same kernel and mean function and use the corresponding posterior mean function as the equality constraint. We sample a single surface in this way and use it for all experiments. The dynamics for the 12DoF quadrotor are from [135] and are given by

$$\begin{bmatrix} x \\ y \\ z \\ p \\ q \\ r \\ \dot{x} \\ \dot{y} \\ \dot{z} \\ \dot{p} \\ \dot{q} \\ \dot{r} \end{bmatrix}_{t+1} = \begin{bmatrix} x \\ y \\ z \\ p \\ q \\ r \\ \dot{x} \\ \dot{y} \\ \dot{z} \\ \dot{p} \\ \dot{q} \\ \dot{r} \end{bmatrix}_{t} + \Delta t \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \\ \dot{p} + \dot{q}s(p)t(q) + \dot{r}c(p)t(q) \\ \dot{q}c(p) - \dot{r}s\dot{p} \\ \dot{q}\frac{s(p)}{c(q)} + \dot{r}\frac{c(p))}{c(q)} \\ -(s(p)s(r) + c(r)c(p)s(q))K\frac{u_1}{m} \\ -(c(r)s(p) - c(p)s(r)s(q))K\frac{u_1}{m} \\ g - c(p)s(q))K\frac{u_1}{m} \\ \frac{(I_y - I_z)\dot{q}\dot{r} + Ku_2}{I_x} \\ \frac{(I_z - I_x)\dot{p}\dot{r} + Ku_3}{I_y} \\ \frac{(I_x - I_y)\dot{p}\dot{q} + Ku_4}{I_z} \end{bmatrix}_{t} \tag{4.53}$$

where $c(p), s(p), t(p)$ are $\cos, \sin, \tan$ functions, respectively. We use parameters $m = 1kg, I_x = 0.5kg \cdot m^2, I_y = 0.1kg \cdot m^2, I_z = 0.3kg \cdot m^2, K = 5, g = -9.81m \cdot s^{-2}$. We use the same dynamics both for planning and for simulation.

We consider three variants of this task with different obstacle avoidance constraints: 1) We consider the case with no obstacles; 2) We consider the case of static obstacles. For the static obstacles case, we wish to demonstrate our method in a cluttered environment with arbitrarily shaped obstacles. We do this by generating the obstacles similarly to the surface constraint, which results in a smooth obstacle constraint. We consider a constraint function $f_{obs}(x, y)$, where the obstacle-free region is $\{(x, y), f_{obs}(x, y) \leq 0\}$. We sample values for $f_{obs}(x, y)$ from a GP prior with an RBF kernel and a constant mean function of -0.5, so that there is a bias

towards being obstacle-free, on a 10×10 grid of (x, y) points. We then use these points as the observations for a GP with the same mean function and kernel as the GP prior. We also add observations at $(-4, -4)$ and $(4, 4)$ of -2, to ensure the start and goal regions are obstacle free. We use the resulting GP posterior mean function as $f_{obs}(x, y)$. We do this once and keep the same obstacle constraint for all trials. The resulting obstacle constraint is shown in Figure 4.4; 3) Finally, we consider a cylindrical obstacle in the x-y plane that moves during the trial in a path that is unknown to the planner; at every timestep, the planner plans assuming the obstacle will remain fixed. If the quadrotor collides with an obstacle during execution then we consider the task failed.

The planning horizon is 12. The posterior $\log p(\tau|o)$ for this problem is a quadratic cost given by

$$\begin{aligned} \log p(\tau|o) = (x_T - x_{goal})^T P(x_T - x_{goal}) + \\ \sum_{t=1}^{T-1} (x_t - x_{goal})^T Q(x_t - x_{goal}) + u_{t-1}^T R u_{t-1}. \end{aligned} \tag{4.54}$$

The control cost is equivalent to the prior on controls $p(u_t) = \mathcal{N}(0, R^{-1})$. The values we use for the costs are

$$Q = \text{Diag}(5, 5, 0.5, 2.5, 2.5, 0.025, 2.5, 2.5, 2.5, 2.5, 2.5, 2.5) \tag{4.55}$$

$$P = 2Q \tag{4.56}$$

$$R = \text{Diag}(1, 16, 16, 16). \tag{4.57}$$

For this problem, we use automatic differentiation to compute all required second derivatives for both IPOPT and CSVTO. We run IPOPT with two different maximum iteration settings. For the first, we limit the maximum number of iterations to 100 for the initial warm start and to 10 for subsequent time steps. We limit the number of iterations so that IPOPT has a comparable computation time to other baselines. The next setting is to set the maximum iterations to 1000, which allows IPOPT to run until convergence for most queries. We refer to this method as

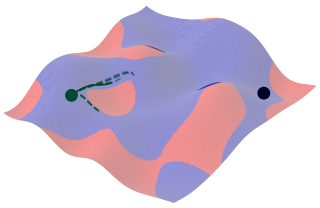


Figure 4.4: Experimental set-up for the quadrotor with static obstacles task. The quadrotor must travel to the goal location, avoiding the obstacles in red while remaining on the blue manifold.

IPOPT-1000. As we will show in Section 4.8.5, this method is substantially slower than other baselines and prohibitively slow for MPC applications, but we included this baseline to see how well IPOPT performs when computation time is not an issue. For the baselines using a penalty method we use $\mu = 2000$ and test two variants for λ: $\lambda = 100$ and $\lambda = 1000$.

In Figure 4.5 we compare CSVTO and IPOPT run for 200 iterations for a single planning query with multiple different initializations, indicating that for the same initializations, CSVTO finds a lower cost local minimum. To generate these initializations, we sample a nominal control sequence from the prior $p(U)$ and use small Gaussian perturbations with $\sigma = 0.01$ around this nominal control sequence as the initialization. The initial state sequence is found by applying these controls with the dynamics. We repeat this process 10 times for a different nominal control sequence. The results demonstrate that parallel trajectory optimization with CSVTO is beneficial even when the initial trajectory distribution is not diverse.

We ran the quadrotor experiments for the three different obstacle cases for 20 trials with randomly sampled starts. The results are shown in Figure 4.6. CSVTO succeeds for 20/20 trials for the no obstacles and dynamic obstacles cases, and 19/20 for the static obstacle case, all with a goal threshold of 0.3m. For the static-obstacle and dynamic-obstacle experiments, IPOPT is the next best performing with 20/20 trials for no-obstacles at a goal threshold of 0.4m, but success falls to 15/20 for both the static-obstacles and dynamic-obstacle case. We see that running IPOPT with more iterations improves performance for the static obstacles case, but in the other

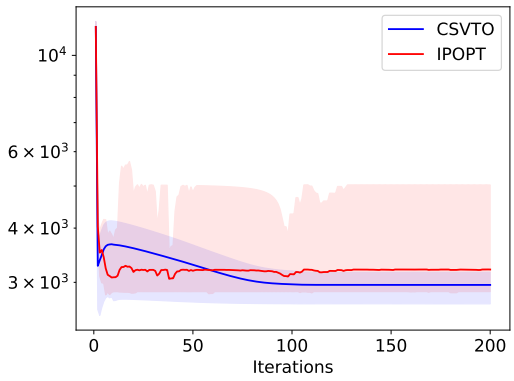


Figure 4.5: Comparison between CSVTO and IPOPT with multiple initializations on the quadrotor task with static obstacles. We compare CSVTO with 8 trajectory samples vs. 8 runs of IPOPT, both from the same initializations and record the minimum cost achieved from the 8 trajectories over 200 iterations of both. We run 10 trials for each method. The shaded regions show the range of the minimum cost achieved over the 10 trials. We see that from the same initializations, CSVTO finds a solution with a lower cost.

two cases, there is no significant difference in performance when allowing IPOPT to run until convergence. However, running IPOPT to convergence has substantially higher computation time, which we will discuss further in section 4.8.5. For the no-obstacles and dynamic-obstacle cases, we see that sample-based methods perform well according to the task success rate, however, they fail to satisfy the surface equality constraint. In addition, both MPPI and SVMPC fail for the static obstacles case.

Trajectories generated from IPOPT vs CSVTO for the dynamic obstacles case are shown in Figure 4.3, IPOPT generates a trajectory aiming to go around the obstacle, but the movement of the obstacle renders that trajectory infeasible as time progresses. IPOPT is not able to adapt the trajectory to go around the obstacle. In contrast, CSVTO generates a multimodal set of trajectories that go either way around the obstacle. It is then able to update the trajectories effectively, avoiding the obstacle and reaching the goal. We do see that IPOPT achieves the lowest constraint violation in the case of no obstacle or a static obstacle, while CSVTO achieves the

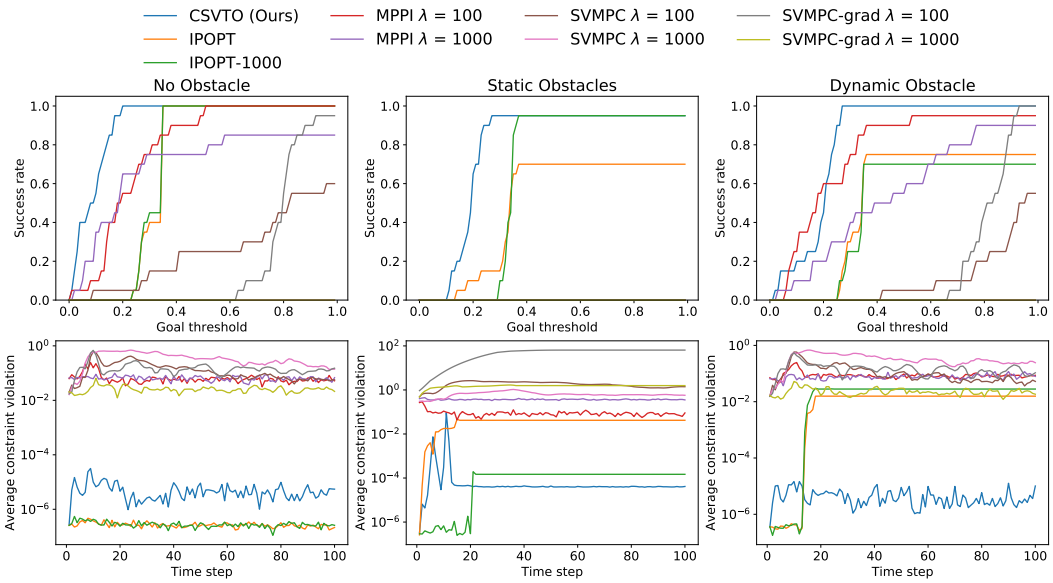


Figure 4.6: Results for quadrotor experiments. The top row shows the success rate as we increase the size of the goal region. The bottom row shows the average surface constraint violation as a function of time. Left) No obstacle. Middle) Static obstacles. Right) Dynamic obstacle.

lowest constraint violation when there is a dynamic obstacle.

4.8.3 Robot Manipulator on Surface

In this task, we consider a 7DoF robot manipulator where the end effector is constrained to move in SE(2) along the surface of a table. The robot must move to a goal location while avoiding obstacles on the surface. The setup is shown in Figure 4.7. This system's state space is the robot's joint configuration $q \in \mathbb{R}^d$. The actions are the joint velocity $\dot{q}$ and the dynamics are given by Euler integration $q_{t+1} = q_t + \dot{q}_t \, dt$, with $dt = 0.1$. The prior distribution over actions is $p(U) = \mathcal{N}(0, \sigma^2 I)$, where $\sigma = 0.5$. The planning horizon is 15. The cost is $C(\tau) = 2500||p_T^{xy} - p_{goal}^{xy}||_2 + 250 \sum_{t=1}^{T-1} ||p_t^{xy} - p_{goal}^{xy}||_2$, where p_t^{xy} is the end effector x, y position which is computed from the forward kinematics. The equality constraints on this system are $p_t^z = 0.8$, which is the height of the table, and additionally, there is an orientation constraint that the z-axis of the robot end effector must be orthogonal to the table, i.e. the

inner product of the table z-axis and the robot z-axis should be equal to -1.

While obeying the table constraint the robot must also avoid 3 obstacles from the Yale-CMU-Berkeley (YCB) object dataset [20]. We enforce this with a constraint that the signed distance to the obstacles must be positive, which we compute from the meshes of the objects. Since signed distance functions (SDFs) are composable via the `min` operator, we combine the SDFs of the three obstacles into a single inequality constraint per timestep rather than an inequality constraint per obstacle. This is to reduce the total number of inequality constraints, as introducing more inequality constraints results in more slack variables and a higher dimensional problem. To evaluate this constraint, offline we generate points on the surface of the robot. Online, we use forward kinematics to map all of these points to the world frame and evaluate their SDF value, selecting the minimum SDF value as the value of the constraint. To compute the gradient of the constraint, consider that for any surface point we can compute the gradient of the SDF value with respect to the point from the object mesh. We then use automatic differentiation to backpropagate this gradient through the forward kinematics to compute a gradient of the SDF value with respect to the joint configuration. Finally, to calculate an overall gradient, we use a weighted combination of the gradients for each surface point, with the weight computed via a `softmin` operation on the SDF values.

The resulting inequality constraint is not twice differentiable, both because of non-smooth object geometries and because of composing SDFs with the `min` operator. Due to this, for CSVTO we omit the second-order term in equation (4.33) for the inequality constraint, and for IPOPT we use L-BFGS to approximate second-order information. Computing the SDF value and gradient is a computationally expensive operation, so we pre-compute grids of the SDF values and the SDF gradients and do a look-up when performing the optimization. We use a $320 \times 320 \times 480$ grid with a resolution of $2.5mm$. There are also joint limit constraints on all of the robot joints.

For the penalty-based baselines, we use penalty parameters of $\mu = 2000$ and variants with $\lambda = 100$ and $\lambda = 1000$. For IPOPT, we found that running until convergence was prohibitively costly, taking several minutes to converge per optimization. For this reason, we limited the maximum number of iterations for IPOPT

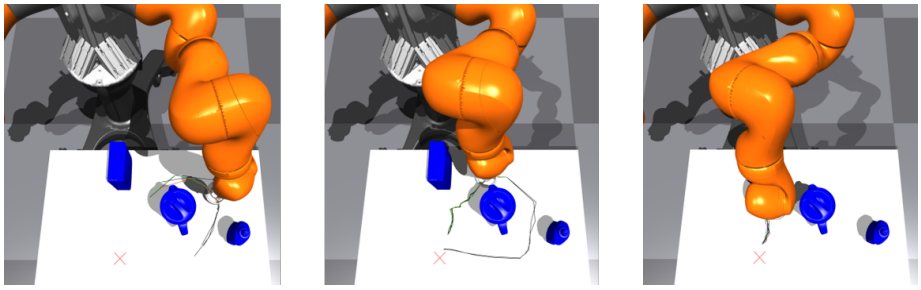


Figure 4.7: Snapshots from CSVTO used for the robot manipulator on a surface experiment. The robot must move the end-effector to a goal location while remaining on the surface of the table and avoiding the obstacles. CSVTO generates trajectories that explore different routes to the goal.

to be the same as CSVTO, resulting in a similar computation time. This is discussed further in Section 4.8.5.

Due to contact with the table, the dynamics of the system used for planning can deviate from those in the simulation. When computing the constraint violation, we use the actual constraint violation in the simulator rather than the planned constraint violation.

We run this experiment for 20 trials with random goals and show the results in Figure 4.8. Our results show that CSVTO succeeds in all 20 trials with a goal threshold 0f 0.1m and achieves the lowest constraint violation of all methods. The next closest baseline, IPOPT succeeds 19/20 times, with the failure case resulting from a poor local minima with q_t and q_{t+1} on either side of an obstacle, but a large distance from one another. This resulted in the robot becoming stuck on the obstacle and unable to make progress.

4.8.4 Robot Manipulator using wrench

In this task, we consider a 7DoF robot manipulator in which the goal is to manipulate a wrench to a goal angle. To turn the wrench, the robot must be able to supply at least 1Nm of torque. The setup is shown in Figure 4.9. The state space is $[q \quad \phi \quad \theta]^T$. $q \in \mathbb{R}^7$ is the configuration space of the robot. ϕ parameterizes the distance between the robot end-effector and the wrench in the x-y plane as $l + \phi$ where l is a nominal distance. θ is the wrench angle. The actions are the joint

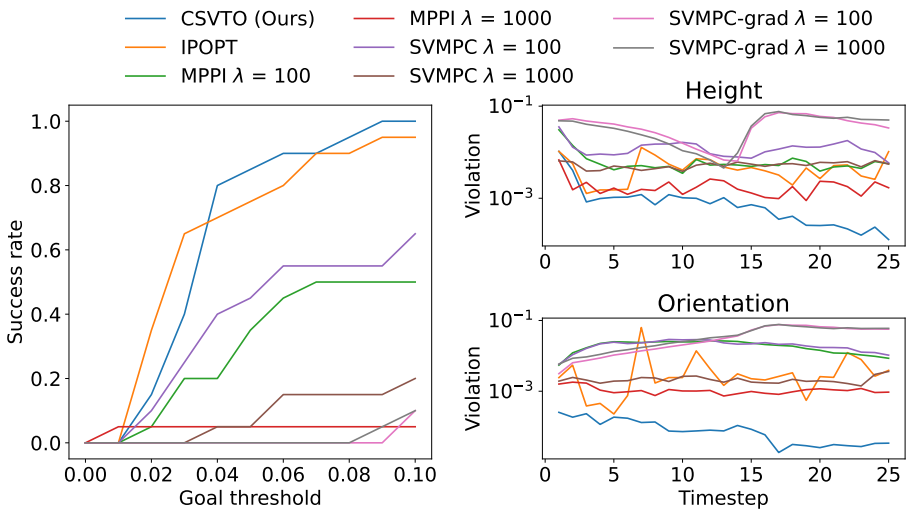


Figure 4.8: Results for robot manipulator on surface experiments Left column shows success rate as we increase the size of the goal region. Right column shows average constraint violation as a function of time for both the height constraint and the orientation constraint.

velocities $\dot{q}$. The dynamics of the joint configuration are given by Euler integration $q_{t+1} = q_t + \dot{q}_t\, dt$, with $dt = 0.1$. We use a simple geometric model for dynamics of ϕ and θ. Assuming that the robot end-effector remains grasping the wrench we compute the next ϕ as $\phi = ||p^{xy}_{ee} - p^{xy}_{wrench}||_2 - l$. To compute the next joint angle θ we use $\theta_{t+1} = \theta_t + \tan \frac{\Delta x_{ee}}{\Delta y_{ee}}$. The prior distribution over actions is $p(U) = \mathcal{N}(0, \sigma^2 I)$, where $\sigma = 1$.

The equality constraints of the system are that p^z_{ee} should be at a fixed height, and additionally that $\theta_T = \theta_{goal}$. There is also a constraint that the end-effector orientation of the robot remains fixed relative to the wrench. To do this we compute the desired end-effector orientation from the wrench angle, and compute the relative rotation between the desired and actual end-effector orientation in axis-angle form, constraining the angle to be zero. In total, combining the dynamics constraints for ϕ and θ with the other equality constraints there are four equality constraints on the pose of the end-effector per time step. When reporting the constraint violation, we report the maximum violation of these four constraints. The inequality constraints of the system are that the desired torque should be achievable within the robot joint

limits. This constraint is $\texttt{min\_torque} \leq J(q)^T(l+\phi) \leq \texttt{max\_torque}$ where J is the manipulator jacobian. There are also joint limit bound constraints, and a bound constraint on ϕ. Computing the second derivative of this constraint requires computing the second derivative of the manipulator Jacobian, which is costly. To avoid this, for CSVTO we omit the second-order terms in equation (4.33) and for IPOPT we use L-BFGS. There is no cost C for this experiment, instead, the inference problem reduces to conditioning the prior on constraint satisfaction. The planning horizon is 12.

For the penalty-based methods we use $\mu = 1000$ and variants with $\lambda = 1000$ and $\lambda = 10000$. We run IPOPT both until convergence with a max number of iterations of 1000 and additionally with a max iterations of 200 at warmup and 20 online, which results in a similar computation time to CSVTO.

We run this experiment for 20 trials with random initializations and show the results in the bottom row of Figure 4.10. This problem is challenging because the dynamics are based on a simple inaccurate geometric model. Compliance in the gripper causes deviation from this geometric model, and the model is only accurate so long as all constraints hold. Our results show that CSVTO can succeed in all 20 trials with a goal threshold of 0.06 radians and achieves the lowest constraint violation. The next closest baseline, SVMPC-grad with $\lambda = 10000$ succeeds 19/20 times with a goal threshold of 0.09 radians, dropping to 11/20 at 0.06 radians. We find that running IPOPT to convergence leads to poor performance, as the solver is unable to converge to a feasible solution. Limiting the maximum iterations to 200 for the initial warm-start and 20 for subsequent online iterations leads to improved task performance, achieving a success rate of 12/20.

We also demonstrate CSVTO on real hardware for the robot manipulator manipulating a wrench task, shown in Figure 4.1. After generating a configuration-space trajectory using CSVTO, we command the robot to move to the first configuration waypoint of that trajectory using a joint impedance controller. Once the robot has reached the desired waypoint, we perform re-planning to generate a new configuration-space trajectory. We use the same hyperparameters as those in the simulator for this experiment. During execution, we applied disturbances by perturbing

the robot end-effector. The impedance controller can reject small disturbances, but larger disturbances require re-planning from the perturbed location. Figure 4.1 shows one such perturbation. Despite large disturbances, our method was able to readjust the grasp and complete the task successfully.

4.8.5 Computation Time

To determine the computation times for CSVTO and each baseline, we ran 10 trials for each experiment on a computer with an Intel i9-11900KF Processor with an NVIDIA RTX 3090 GPU. We record the average computation times for the initial trajectory as well as subsequent online trajectories, which we refer to as t_w and t_o, respectively. We also record the standard deviations of the computation times. The number of iterations used for the warm-up and online phase is K_w and K_o, respectively. For IPOPT this is a maximum number of iterations, and the solver may terminate early. For all other methods, all iterations are used.

4.8.5.1 12DoF Quadrotor

The average computation time of CSVTO compared to baselines for all quadrotor experiments is shown in Table 4.2. For this experiment, computing the gradient was a major computational bottleneck, thus for the sample-based methods we allowed them more iterations. We see that MPPI and SVMPC are faster than CSVTO with online trajectory computation times of $0.366s$, $0.439s$, and $0.589s$, respectively. For the no-obstacles and dynamic-obstacle cases, IPOPT is also faster than CSVTO with an average online computation time of $0.429s$ and $0.479s$ due to early termination. However, for the static obstacles case, this rises to $0.768s$ compared to CSVTO at $0.650s$. When running IPOPT to convergence, the solving time is substantially larger, with an average computation time for the static obstacle case of $15.8s$. We also see that the standard deviations are very large, due to the variability in how quickly the solver converges. Combining these with the results from Section 4.8.2, we see that CSVTO outperforms IPOPT to convergence with substantially faster computation times.

4.8.5.2 Robot 7DoF Manipulator

The computation times for all methods on both 7DoF manipulation experiments are shown in Table 4.3. For the manipulator on a surface experiment, the difference in computation speed of the sample-based vs gradient-based algorithms per iteration was less pronounced than for the quadrotor experiment. We thus kept the number of iterations the same for all experiments, with 100 warm-up iterations and 10 online iterations. CSVTO and IPOPT have similar computation times at $1.12s$ and $1.14s$ to compute a trajectory online. MPPI is again the fastest algorithm at $0.691s$ to generate a trajectory online, though the performance is lower both in terms of task success and constraint violation. Initial attempts to run IPOPT with a maximum of 1000 iterations took several minutes to solve, which rendered it impractical.

For the wrench task, CSVTO and SVMPC-grad have similar computation times. While CSVTO requires the computation of the second derivative of the constraints, the cost evaluation of SVMPC-grad requires a loop through the time horizon, slowing down both cost and gradient evaluation. Since CSVTO employs a collocation scheme this process is vectorized. Whether CSVTO or SVMPC-grad is faster depends on the relative cost of computing the second derivatives vs. looping through the time horizon. Each iteration of IPOPT was faster than CSVTO for this experiment, as IPOPT using the L-BFGS approximation computes no second derivatives, whereas CSVTO only neglected the second derivatives of the force inequality constraint. We thus allowed IPOPT more iterations, as seen in Table 4.3. Attempting to allow IPOPT to run with a much larger maximum iteration number resulted in much slower solving times and worse performance.

Table 4.1: Hyperparameter values for the three experiments

Experiment	# particles	α_J	α_C	ϵ	K_w	K_o	`resample_steps`	β	$\sigma_{resample}$	λ	W
Quadrotor	8	0.1	1	0.5	100	10	10	0.55	0.1	1000	3
Manipulator on Surface	8	0.01	1	0.1	100	10	1	0.1	0.01	1000	3
Manipulator wrench	4	0.01	1	0.25	100	10	1	0.1	0.01	1000	3

Table 4.2: Mean and standard deviation of computation times for CSVTO and all baseline methods for the 12DoF quadrotor experiments. t_w and t_o are the average times taken to generate the trajectories for the warm-up phase and online phase, respectively

			No Obstacles		Static Obstacles		Dynamic Obstacle	
Method	K_w	K_o	t_w (s)	t_o (s)	t_w (s)	t_o (s)	t_w (s)	t_o (s)
CSVTO (Ours)	100	10	5.92 ± 0.235	0.589 ± 0.003	6.56 ± 0.39	0.650 ± 0.025	6.47 ± 0.344	0.643 ± 0.021
IPOPT	100	10	4.36 ± 2.29	0.429 ± 0.008	7.19 ± 2.48	0.768 ± 0.069	3.19 ± 1.89	0.479 ± 0.097
IPOPT-1000	1000	1000	17.5 ± 30.2	2.40 ± 1.10	39.2 ± 32.0	15.8 ± 10.1	10.8 ± 24.5	2.45 ± 2.26
SVMPC-grad	100	10	8.25 ± 0.080	0.771 ± 0.217	8.24 ± 0.061	0.765 ± 0.23	8.28 ± 0.054	0.850 ± 0.014
SVMPC	250	25	4.26 ± 0.030	0.439 ± 0.002	6.07 ± 0.031	0.621 ± 0.003	4.35 ± 0.24	0.449 ± 0.017
MPPI	250	25	3.63 ± 0.021	0.366 ± 0.0019	5.45 ± 0.039	0.55 ± 0.003	3.66 ± 0.13	0.373 ± 0.016

Table 4.3: Average computation times for CSVTO and all baseline methods for the 7DoF robot manipulator experiments. t_w and t_o are the average times taken to generate the trajectories for the warm-up phase and online phase, respectively

	Surface				Wrench			
Method	K_w	K_o	t_w (s)	t_o (s)	K_w	K_o	t_w (s)	t_o
CSVTO (Ours)	100	10	9.41 ± 0.42	1.12 ± 0.19	100	10	9.62 ± 0.84	0.64 ± 0.004
IPOPT	100	10	10.26 ± 3.5	1.14 ± 0.27	200	20	5.82 ± 0.54	0.493 ± 0.028
IPOPT-1000	1000	1000	—	—	1000	1000	30.8 ± 2.51	22.7 ± 2.84
SVMPC-grad	100	10	8.55 ± 0.072	1.10 ± 0.27	100	10	9.54 ± 0.071	0.732 ± 0.004
SVMPC	100	10	7.27 ± 0.097	0.758 ± 0.010	100	10	7.44 ± 0.15	0.571 ± 0.007
MPPI	100	10	6.91 ± 0.12	0.691 ± 0.028	100	10	7.05 ± 0.11	0.506 ± 0.006

4.9 Discussion

In this section we will discuss some of the advantages of CSVTO over baselines, and then discuss some limitations and finally highlight areas for future work.

4.9.1 Local minima

CSVTO produces diverse approximately constraint-satisfying trajectories. By encouraging diversity through the course of the optimization the algorithm searches the solution space more widely and can result in multi-modal sets of solutions, for example, see Figure 4.3. We found that this behavior is beneficial for escaping from local minima. This was most clearly demonstrated in the 12DoF Quadrotor experiment. We found that in the case of no obstacles, IPOPT was consistently able to get relatively close to the goal, achieving a 100% success rate at a goal region of 0.4m. However, it was unable to escape a local minimum in the vicinity of the goal region. This local minimum appears to be induced by the surface constraint, as IPOPT frequently became stuck at a position where it needed to climb in height to reach the goal while satisfying the constraint, incurring a large control cost. In contrast, CSVTO was able to achieve a 100% success with a much smaller goal region of 0.2m.

4.9.2 Initialization

CSVTO optimizes a set of trajectories in parallel. Each of these trajectories has a different random initialization, and, as mentioned, the objective encourages trajectory diversity. We find that this approach is effective at making the algorithm more robust to poor initialization. This is most clearly seen in the 7DoF wrench manipulation experiment, shown in Figure 4.9. This system is highly constrained, and we can see from figure 4.9 that the trajectories generated by IPOPT can be very low quality when poorly initialized. This is reflected in the success rates, where in our experiments CSVTO succeeds for 20/20 of the trials vs. 12/20 for IPOPT.

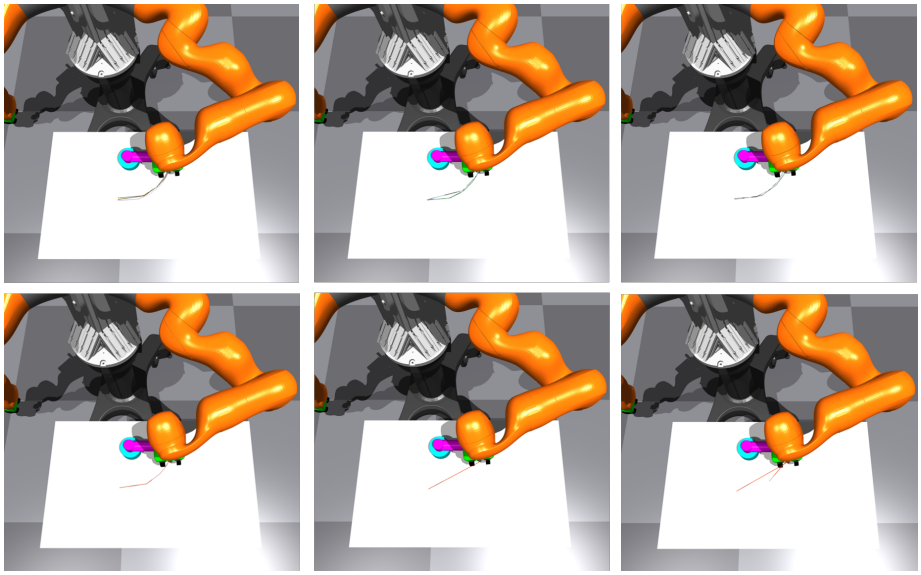


Figure 4.9: The robot manipulator turning a wrench experimental set-up. The goal is to turn the wrench by 90 degrees. End-effector planned path at the first time-step visualized for three different initial trajectories generated by CSVTO (Top) and IPOPT (Bottom). CSVTO's end-effector path traces an arc around the wrench center to turn the wrench, while IPOPT paths are often poor, containing very large steps and lacking smoothness

.

4.9.3 Limitations & Future Work

Differentiability Our method requires that all costs and constraints are differentiable. This is a restrictive assumption, particularly when treating dynamics as a constraint. Many contact-rich robot manipulation tasks exhibit discontinuities that invalidate this assumption.

Slack variables Our approach converts inequality constraints to equality constraints by introducing slack variables. While this is a natural way of incorporating inequality constraints into our method, it results in increasing the number of decision variables by the number of inequality constraints. This is likely to be problematic for long-horizon planning tasks with many inequality constraints. A possible solution would be solving a QP subproblem at every iteration to determine the active

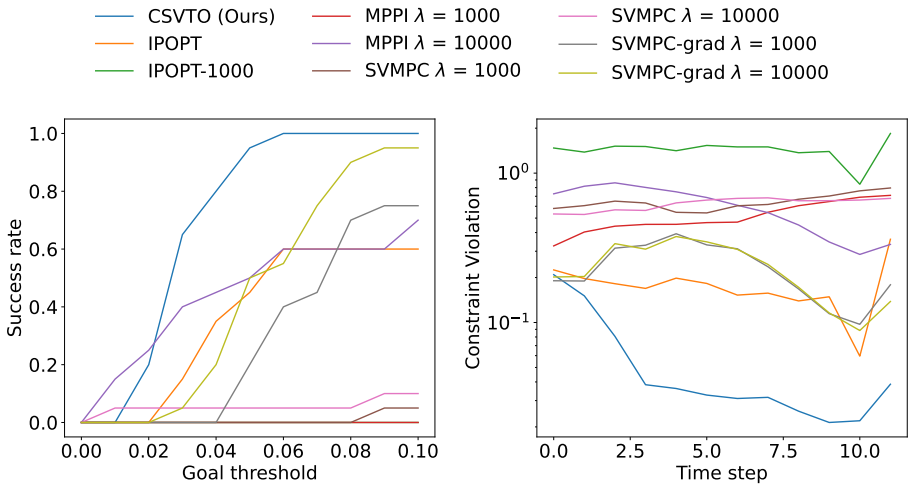


Figure 4.10: Results for the robot manipulator using the wrench. Left column shows the success rate as we increase the size of the goal region. The right column shows the average constraint violation as a function of time time, where we compute the constraint violation at a given time via the maximum violation among the equality constraints.

inequality constraints as in [39], however, this has the issue that we would need to solve an individual QP subproblem for every particle.

Computation time inadequate for real-time control We note from Table 4.2, in the dynamic obstacle quadrotor task the average computation time for online trajectory generation is $0.643s$ for CSVTO, compared to MPPI, the fastest baseline, taking $0.373s$. In this case, the solve times for the current implementation of CSVTO and all baselines are insufficient for real-time reactive control. Our method, all baselines other than IPOPT, and all cost and constraint functions were implemented in Python, using automatic differentiation in PyTorch to compute the relevant first and second derivatives. Implementing these methods in C++, using a library such as CasADI [4] for automatic differentiation, may enable real-time performance on these systems in future work.

Kernel selection While our approach decomposes the kernel into a sum of kernels operating on sub-trajectories, each of these kernels is an RBF kernel. While the RBF

has attractive properties, such as strict positive-definiteness and smoothness, we believe that exploring task-specific kernels for trajectory optimization is an interesting avenue for future work.

4.10 Conclusion

In this chapter, we presented Constrained Stein Variational Trajectory Optimization (CSVTO), an algorithm for performing constrained trajectory optimization on a set of trajectories in parallel. To develop CSVTO we formulated constrained trajectory optimization as a Bayesian inference problem, and proposed a constrained Stein Variational Gradient Descent (SVGD) algorithm inspired by O-SVGD [179] for approximating the posterior over trajectories with a set of particles. Our results demonstrate that CSVTO outperforms baselines in challenging highly-constrained tasks, such as a 7DoF wrench manipulation task, where CSVTO succeeds in 20/20 trials vs 12/20 for IPOPT. Additionally, our results demonstrate that generating diverse constraint-satisfying trajectories improves robustness to disturbances, such as changes in the environment, as well as robustness to initialization.

CHAPTER V

Parallel Trajectory Optimization for Dexterous Manipulation

5.1 Introduction

In Chapter IV, we introduced a method for constrained trajectory optimization on a set of trajectories in parallel. We demonstrated this method applied to two constrained manipulation tasks; planning with the end-effector constrained to move on a surface and turning a wrench. In these situations, the tasks can be completed by the gross motion of the arm. However, for many tasks we need robots to exhibit more dexterity in their manipulation, particularly when operating in environments and using tools that are designed for humans. For instance, consider a robot turning a precision screwdriver. The robot needs to supply sufficient torque to turn the screwdriver while maintaining the grasp. While we could use the gross motion of the arm to turn the screwdriver, by using a dexterous multi-fingered hand we can achieve continuous turning without having to reorient the arm. Another possible solution would be to design a custom end-effector to turn the screwdriver continuously. However, this is just a single task. For a multi-stage task using a screwdriver may only be necessary for one stage, and other stages may require picking up an object, or the use of other tools such as a wrench.

In this chapter [1], we demonstrate that CSVTO is effective for optimizing trajec-

[1] This chapter is based on unpublished work done with Fan Yang

tories for challenging, dexterous manipulation tasks. We show CSVTO being applied to two tasks using a multi-fingered hand, turning a 1DoF valve, and turning a precision screwdriver. Our results show that for the screwdriver case, the challenge of the resulting optimization leads to IPOPT, a general nonlinear solver [157], to drop the screwdriver in 3/10 attempts. In contrast, CSVTO can successfully turn the screwdriver for all 10 attempts.

5.2 Related Work

5.2.1 Planning

There is a long history of developing planning methods for dexterous manipulation [153, 172, 62]. Many of these methods explicitly reason about changes in contact state. For instance, Jijie et al. [172] propose sample-based motion planning in a hybrid configuration space for finger gait planning. Recently, the CMGMP algorithm was proposed by Cheng et al. [25, 26] which automatically generates possible contact modes in a sample-based motion planning framework. Other approaches have used Monte-Carlo Tree Search (MCTS) to plan contact modes [27]. Optimization-based approaches have also been used for dexterous manipulation [92, 55, 143, 134, 152]. Liu [92] proposed a multi-step optimization framework that first optimizes contact forces, and then optimizes motion. Other methods have focused on optimizing a plan within a given contact mode [143], or given an existing sequence of constraints [26]. To optimize the sequence of contacts one approach is to explicitly search for a schedule of contacts while optimizing a trajectory, which can be formulated as a Mixed Integer Program [100]. Other methods rely on an implicit representation of the contact schedule, i.e. contact-implicit trajectory optimization [121, 63]. This approach leads to optimization problems that are difficult to solve, recent work by Chen et. al. combines this approach with MCTS to improve planning times and feasibility [23].

5.2.2 Learning based

Recently, learning approaches have become more popular for dexterous manipulation. These methods do not require hand-specified models and precisely known object geometries but learn from data. Some approaches learn models which are then used for planning. For instance, methods have been proposed that use locally linear models [80], as well as neural network dynamics models [108]. Other approaches have used reinforcement learning to learn policies directly from experience [5, 175], or from demonstration [130]. Recent work has proposed combining learning-based approaches and model-based planning within a hierarchical framework [47, 177].

5.3 Problem Statement

This chapter considers the dexterous manipulation of an object using a multi-fingered hand. The configuration of the object is given by q^O, and the configuration of the i-th finger is given by q^i. The objective is to find a sequence of commanded joint positions $\{\hat{q}_t^i, t \in [1,T], i \in I_{fingers}\}$ such that the object moves from some initial configuration q_0^O to a goal configuration q_g^O. We assume the contact mode does not change during the task. Specifically, fingers in contact will remain in contact and no additional contacts will be introduced during the manipulation. We assume that the system is quasi-static.

5.4 Methods

To solve the problem outlined in Section 5.3, we frame it as a trajectory optimization problem. We optimize for a sequence of joint configurations $\{q_t^i,\ t \in [1,T], i \in I_{fingers}\}$, and commanded change in joint configuration $\{\Delta q_t^i,\ t \in [1,T], i \in I_{fingers}\}$. Note that $\hat{q}_{t+1}^i = q_t^i + \Delta q_t^i$, and that $\hat{q}_t^i$ is the *commanded* joint configuration whereas q_t^i is the *actual* joint configuration. Making this distinction allows us to reason about applied forces, for instance by commanding a joint configuration which would lead to penetration of the obstacle we apply a force to the object.

The objective of the trajectory optimization problem is to minimize the following

$$\min_{\substack{q_t^i, \Delta q_t^i, q_t^O \\ t\in[1,T], i\in I_{fingers}}} \sum_{t=0}^{T-1} \left(||q_{t+1} - q_t||^2 + \sum_i ||\Delta q_{i,t}||^2 \right) + ||q_T^O - q_g^O||^2, \tag{5.1}$$

where $q_t = [q_t^O, \{q_t^i, \ \ i \in I_{fingers}\}]$ is the overall configuration of the fingers and object at time t. This objective encourages the trajectory to take small steps in configuration space while minimizing the final distance to the goal for the object configuration. To ensure the trajectory is physically feasible, we use a variety of different constraints that we now discuss.

5.4.1 Contact Constraints

The contact constraints are in place to ensure that the relevant fingers make contact with the object. To do this, we assume that we have access to a signed distance function for the object, expressed as $\phi_O(p^O)$, which allows us to compute both the signed distance and the surface normals of the object at points p^O in the object frame. For each fingertip we first pre-compute N points on the surface of the fingertip mesh, expressed in the fingertip frame as $\{p_k^f\}_{k=1,\dots,N}$. When querying the value of the contact constraints for configurations $(q^O, \{q^i\}_{i\in I_fingers})$ we map all points to the world frame using the forward kinematics to compute $\{p_k^W\}_{k=1,\dots,N}$, and then transform these to the object frame. We then compute the signed distance and surface normal for each of these points $\{\phi_O(p_k^O), \nabla_{p_k^O}\phi_O(p_k^O)\}_{k=1,\dots,N}$. In principle the overall signed distance is $\min_k \phi_O(p_k^O)$, however, this minimum operator is non-differentiable. We instead use the well-known *softmin* smooth approximation

$$w_k = \frac{\exp -\beta\phi_O(p_K^O)}{\sum_{k=1}^N \exp -\beta\phi_O(p_k^O)}, \tag{5.2}$$

where β is a temperature parameter; we use $\beta = 1000$. Our contact constraint for each finger is $\sum_k w_k \phi_k^O(p_k^O) = 0$. To get the derivative of this constraint, note that the derivative of the SDF of the k-th point in the object frame, $\nabla_{p_k^O}\phi_O(p_k^O)$, is the surface normal. The gradient of the SDF of the k-th point with respect to the object

configuration is $-\nabla_{p_k^O}\phi_O(p_k^O)J^O(q^O)$, where $J^O(q^O)$ is the jacobian of the object at configuration q^O. Similarly, the gradient of the SDF of the k-th point with respect to the finger configuration is $\nabla_{p_k^i}\phi(p_k^i)J^i(q^i)$, where $\nabla_{p_k^i}\phi(p_k^i)$ is the surface normal translated to the i-th finger frame, and $J^i(q^i)$ is the jacobian of the i-th finger. Using these we can compute an overall gradient of the SDF using the differentialibility of the softmin operation. In addition, we also use the softmin weight to estimate the contact points $p_c^O = \sum_k w_k p_k^O$. Note that for this constraint we do not enforce a particular contact point, therefore the precise contact point can vary through the optimization.

5.4.2 Kinematics Constraints

We use kinematics constraints to enforce consistency between the change in the configuration of the fingertips and the movement of the object. For each contact, we estimate the velocity of the contact point on the fingertip in the object frame as

$$v_t^i = J_{p^i}(q_t^i)(q_{t+1}^i - q_t^i), \tag{5.3}$$

where $J_{p^i}^i$ is the contact Jacobian for the i-th finger evaluated at the contact point p^i. We then estimate the velocity of the contact point on the object as

$$v_t^O = J_{p^O}^O(q_t^O)(q_{t+1}^O - q_t^O), \tag{5.4}$$

where $J_{p^O}^O$ is the contact Jacobian of the object evaluated at the contact point. Assuming a sticking contact, the instantaneous velocities for the contact point on the finger and object should be equal to one another.

5.4.3 Force Constraints

We include the contact forces per finger as a decision variable $\{f_t^i,\ t \in [1, T], i \in I_{fingers}\}$. The purpose of this is for two reasons, the first is to enforce friction constraints, and the second to enforce a wrench balance to ensure the quasi-static assumption holds. We assume that force balance only holds at the *end* of a transition.

Therefore the force f_t^i is defined as the instantaneous contact force after we have transitioned from q_t^i to q_{t+1}^i, immediately before beginning the transition to q_{t+2}^i. At this instance, the controller has a set-point of $q_t^i + \Delta q_t^i$, but has actually achieved q_{t+1}^i. The forces must satisfy the following constraints

$$J_{p^i}(q_{t+1}^i)^T f_t^i = K_p(q_t^i + \Delta q_t^i - q_{t+1}^i), \tag{5.5}$$

$$\sum_i f_t^i \times p^O = 0, \tag{5.6}$$

The second of these two constraints enforces static equilibrium. Note that this constraint is a torque balance constraint. We do not include full force balance, since in all of our experiments the position of the object is constrained; it is the orientation that is controlled. Extending to full 6D motion is intended for future work.

5.4.4 Friction Constraints

To ensure we have a sticking or rolling contact, we place inequality constraints that the forces must lie in the friction cone. We use a linearized friction cone. Thus have the following constraint

$$Af_t^i \leq 0, \tag{5.7}$$

Where A describes the linearized friction cone. When linearizing the friction cone transforming the conic constraint into 8 linear inequality constraints. This constraint ensures that the contact force at the end of transition from q_t to q_{t+1} lies the linearized friction cone. We also add an additional constraint that the contact force at the beginning of the transition lies in the linearized friction cone. We approximate this force as $J_p(q_t^i)(q_{t+1}^i - q_t^i)$. Note that this assumes the contact force is proportional to the contact velocity, and effectively assumes that $J^{T\dagger} = J$, where $J^{T\dagger}$ is the psuedo-inverse of J^T. This leads to the following inequality constraint

$$AJ_p(q_t^i)(q_{t+1}^i - q_t^i) \leq 0. \tag{5.8}$$

5.4.5 Rolling Contacts

The contact constraint outlined above does not enforce a particular contact point, rather this contact point varies through the optimization. In addition, the kinematics constraint and friction constraints themselves only prohibit sliding movement. Therefore our trajectory optimization allows for rolling contacts. This is crucial for increasing flexibility given the limited degrees of freedom for each finger.

5.5 Results

In this section, we evaluate the application of CSVTO to challenging dexterous manipulation tasks. We evaluate CSVTO on two different experimental setups using the allegro hand. For the first, we use the thumb and index finger of the allegro to turn a 1-D valve. The configuration of the object is the valve angle θ. For the second experiment, we use the thumb, index, and middle fingers of the allegro to turn a precision screwdriver. We model the screwdriver as connected to a surface using a ball joint, hence the configuration for the object is $q^O = [\theta, \psi, \phi]^T$. We run these experiments in IsaacGym. For all experiments, we run with a time horizon of 15.

5.5.1 Baseline

We compare using CSVTO to solve the trajectory optimization with IPOPT [157], a general nonlinear optimizer. We use IPOPT, using L-BFGS to approximate the Hessian of the Lagrangian. For CSVTO we neglect second order terms, using only the gradients of thee cost and constraints. For both IPOPT and CSVTO we set the maximum number of iterations to be 1000.

5.5.2 Valve turning

We set the goal valve angle to be $\theta_g = \frac{\pi}{3}$, where the initial valve angle is 0. We show an example execution of trajectory planned by CSVTO in Figure 5.2. The performance of CSVTO and IPOPT is shown in Figure 5.1. We solve the same optimization problem for 10 different random initializations. CSVTO and IPOPT

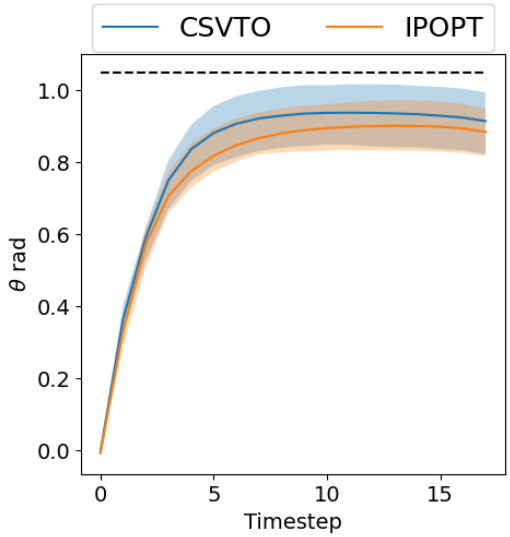


Figure 5.1: The average executed valve angle for the allegro valve turning task over 10 initializations. The shaded region shows the range of executed valve angles. The goal valve angle is shown in dotted black.

perform similarly for this experiment, successfully turning the valve, though not able to precisely reach the goal due to the kinematics of the robot. The average solve time for CSVTO is 193s compared with IPOPT at 711s, showing that CSVTO achieves similar performance with much faster solve times.

5.5.3 Screwdriver turning

We set the goal screwdriver orientation to be $q_g^O = [0, 0, -\frac{\pi}{3}]^T$, where the initial screwdriver orientation is $[0, 0, 0]^T$. As a result, the trajectory should turn the

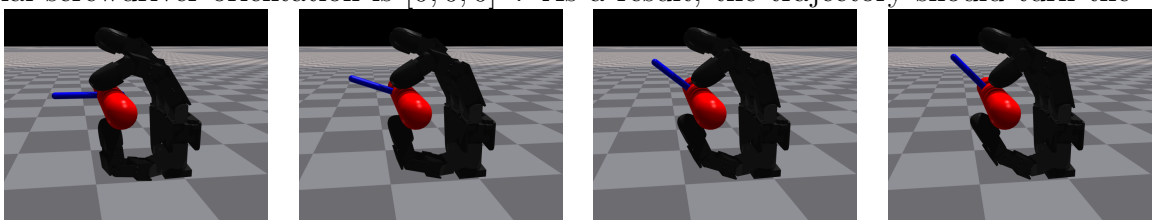


Figure 5.2: Snapshots of execution of trajectory planned by CSVTO, turning the valve.

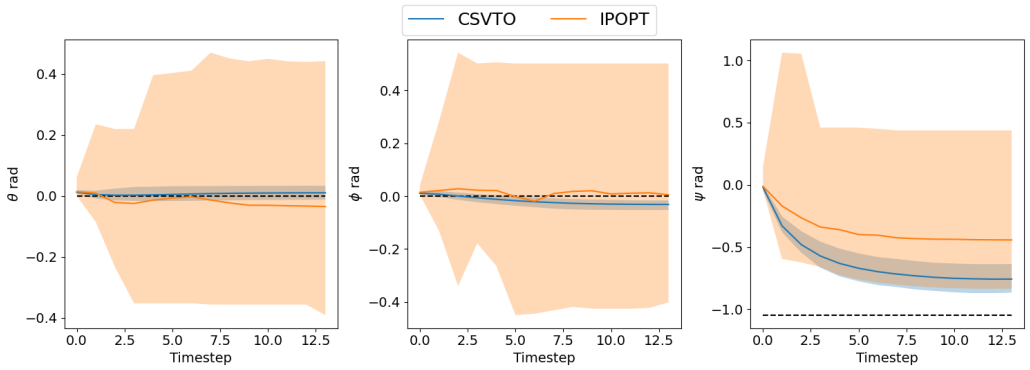


Figure 5.3: The average executed screwdriver angles for the allegro valve turning task over 10 initializations. The shaded region shows the range of executed valve angles. The goal valve angles are shown in dotted black.

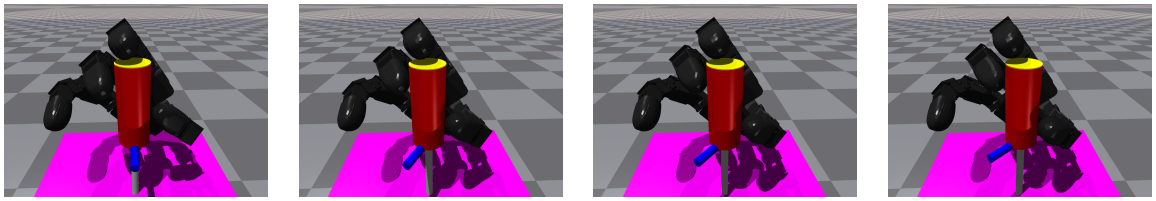


Figure 5.4: Snapshots of execution of trajectory planned by CSVTO, turning the precision screwdriver.

screwdriver while keeping it in an upright position. We show an example execution of trajectory planned by CSVTO in Figure 5.4. The performance of CSVTO and IPOPT is shown in Figure 5.3. We solve the same optimization problem for 10 different random initializations. We see that for this experiment CSVTO outperforms IPOPT. For all 10 trials, CSVTO is able to turn the screwdriver successfully while maintaining the grasp. In contrast, as seen by the large range of outcomes for IPOPT, IPOPT drops the screwdriver in 3/10 trials. For this task the average solve time for CSVTO is 315 compared with IPOPT at 1166, showing that CSVTO achieves better performance with much faster solve times.

5.6 Conclusion

In this chapter we demonstrated CSVTO on challenging dexterous manipulation tasks using a multi-fingered hand. We demonstrated that we can outperform IPOPT [157], a general nonlinear optimization solver for the task of turning a precision screwdriver, where poor initialization results in IPOPT dropping the screwdriver in 3/10 cases, compared to CSVTO which does not drop the screwdriver in any of the 10 trials. In future work we intend to accelerate the trajectory optimization by learning a generative model of trajectories with which to initialize the optimizer. In addition, we intend to include the contact mode into the optimization so that we can optimize the sequence of contacts to complete a task. For example, we would like to be able to optimize the sequence of contacts required to continuously turn the precision screwdriver.

CHAPTER VI

Sampling Constrained Trajectories Using Composable Diffusion Models

Chapter IV proposed a method for generating multiple diverse constraint-satisfying trajectories in parallel. However, this method can be costly. In this chapter, we propose using diffusion models to learn a distribution over constraint-satisfying low-cost trajectories. This learned distribution will then used with CSVTO to efficiently generate multiple constraint satisfying trajectories. We propose exploiting the composability of diffusion models to generalize the learned generative model to out-of-distribution constraints which consist of the composition of multiple in-distribution constraints. Our results demonstrate the benefit of our approach by showing improvement over baselines on a constrained 12DoF Quadrotor task and a 7DoF robot manipulator task. Future work will be to extend this to a wide range of parameterized constraints relevant to manipulation planning. [1]

6.1 Introduction

Trajectory optimization and optimal control are important tools for generating complex robot behavior [17, 139, 56, 118, 18]. When performing trajectory optimization, ensuring constraint satisfaction is crucial to ensure the trajectories are

[1] The work in this chapter appeared in [126]

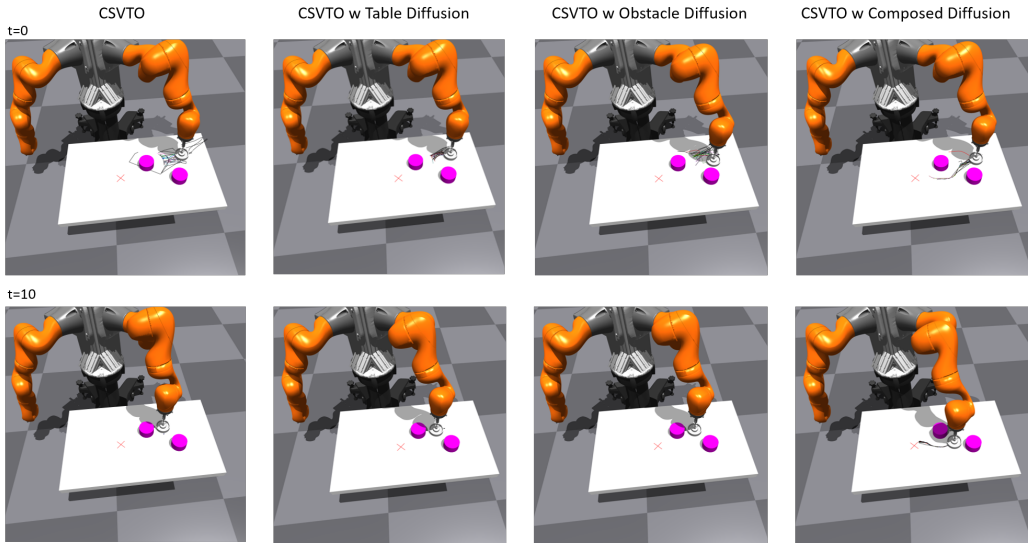


Figure 6.1: Example trajectories for the 7DoF manipulator on a table experiment. At the first timestep, the initial trajectories from CSVTO are quite poor, and CSVTO becomes stuck unable to pass through the narrow passage between obstacles. CSVTO with the single-constraint diffusion models both generate initial trajectories towards the goal but fail to make progress past the initial passage. CSVTO with the composed diffusion generates trajectories that immediately pass through the narrow passage and satisfy the table constraint, and successfully traverses the passage.

safe. Satisfying these constraints can be very difficult as constraint-satisfying trajectories may lie on implicitly-defined lower-dimensional manifolds that have zero measure, presenting difficulties for sample-based methods. In addition, many useful tasks entail constrained optimization problems that are non-convex and exhibit multiple local minima. This makes trajectory optimization difficult for gradient-based methods, as poor initialization may lead to poor local minima or even infeasible solutions. This is particularly problematic when re-solving the optimization problem online under limited computation time when disturbances can lead to the previous solution becoming a poor initialization for the current optimization problem.

In this chapter, we formulate the constrained trajectory optimization problem as a Bayesian inference problem. This view has advantages as it aims to find a distribution over trajectories rather than a single trajectory alone. As noted by Lambert et. al. [83], commonly used Variational Inference approaches [16] lead to

minimizing entropy-regularized objectives [83] which can improve exploration of the search space and give greater robustness to initialization. Previous methods taking the inference view of trajectory optimization have only been able to incorporate constraints via penalties in the cost [176, 106, 83, 84]. A drawback of penalty methods is that selecting the relative weights of the penalties is challenging due to possible conflicts with the objective function.

Recently, Power and Berenson [124] proposed Constrained Stein Variational Trajectory Optimization (CSVTO), an algorithm that uses a non-parametric approximation of the posterior over low-cost constraint-satisfying trajectories. By generating diverse sets of constraint-satisfying trajectories, this method is more robust to initialization than baselines that rely on a single trajectory. However, the method can still fail if all of the trajectory initializations are poor. In addition, the computational time increases as the number of constraints and time horizon increases. When running with a limited computational budget we generally do not have time to run until convergence. In this chapter, we will discuss our proposed future work learning a generative model of constraint-satisfying trajectories which is used with CSVTO. Crucially, we propose using *composable* diffusion models to generalize the learned generative model to out-of-distribution constraints which consist of the compositions of constraints seen in training. This composition ability is important for tasks where the task-specific constraints are not known at training-time and training on all possible constraints the robot might encounter is intractable. Our preliminary results show improved performance with a finite computational budget for two experiments; a 12DoF quadrotor and a 7DoF manipulator.

6.2 Related Work

6.2.1 Learning-based Constrained Planning

Learning-based approaches have previously been used to improve planning in constrained domains. Qureshi et al. proposed Constrained Motion Planning Networks (CoMPNetX) [129], a learning-based method for constrained sample-based motion

planning. Generative Adversarial Networks (GANs) have been used to learn distributions of configurations satisfying constraints, by Lembono et al [88] for use with constrained sample-based motion planning and by [113] for generating initializations for a trajectory optimization problem. Kicki et al. proposed an approach for generating constraint-satisfying trajectories with a neural network that outputs a B-spline parameterization of the trajectories [72].

6.2.2 Diffusion Models in Robotics

Diffusion probabilistic models [53, 141, 142] are a class of generative models that have been recently applied to generating high-quality images [53], trajectories [61, 2] learning multi-modal policies [28] and learning costs for grasp optimization [155]. Two mechanisms have been proposed for incorporating conditioning information, classifier-guidance [32], which uses the gradients of an additionally learned classifier with the unconditional diffusion model, and classifier-free guidance [52] which instead learns a conditional diffusion model which takes the context information as input to the diffusion model. One interesting feature of diffusion models that has been recently explored is the composition of context information at test time, generalizing to novel combinations of context [94, 34, 2].

6.3 Problem Statement

We frame the constrained optimal control problem introduced in Section 4.3 as a probabilistic inference problem, using ideas developed in Section 4.4. We can consider the constrained optimization problem as an unconstrained optimization problem with infinite cost assigned to constraint violations. This results in $p(o = 1 = |\tau) = 0 \implies p(\tau|o = 1) = 0$, hence constraint violating trajectories are zero probability. We can convert the unconstrained optimization problem with infinite cost to the following

constrained optimization problem on the space of probability distributions:

$$
\begin{aligned}
q^* = \min_{q} \quad & \tilde{\mathcal{F}}(q) \\
\text{s.t.} \quad & \\
& P_q(h(\tau) = 0) = 1 \\
& P_q(g(\tau) \leq 0) = 1 \\
& \forall t \in \{1, \ldots, T\} \\
& P_q(f(\mathbf{x}_{t-1}, \mathbf{u}_{t-1}) = \mathbf{x}_t) = 1 \\
& P_q(\mathbf{u}_{\min} \leq \mathbf{u}_{t-1} \leq \mathbf{u}_{\max}) = 1 \\
& P_q(\mathbf{x}_{\min} \leq \mathbf{x}_t \leq \mathbf{x}_{\max}) = 1.
\end{aligned} \tag{6.1}
$$

CSVTO solves this functional optimization problem with a non-parametric approximation of q^*, and solves a single planning query for a specified cost C, constraints g, h, and dynamics f. We can then say that $q^* = q^*(C, g, h, f)$. We will assume f is fixed, and that the cost C is parameterized by a start x_0 and a goal x_g. We assume that the constraints g, h are parameterized by $\{\theta, y\}$, where $\theta \in \mathbb{R}^n$ are continuous parameterizations of the constraint and $y \in [1, ..., M]$ is an indicator variable for the constraint type. For instance, different types of constraints could be obstacle avoidance constraints vs. end-effector pose constraints. Thus $q^* = q^*(x_0, x_g, \theta)$. Rather than repeatedly solve this optimization problem from scratch, we aim to learn a generative model which approximates this q^*. For a given x_0, x_g, θ we use CSVTO to generate sampled trajectories $(\mathbf{X}, \mathbf{U})$ from q^*. The data from which we will learn our generative model is $\{\{\mathbf{X}_i, \mathbf{U}_i\}_{i=1}^K, x_0, x_g, \theta, y\}^N$. By using this generative model as an initialization, our goal is to achieve better performance and lower constraint violation within a limited computational budget. In addition, we seek to generalize to unseen combinations of constraints, i.e. for constraints h_i, h_j seen individually during training we aim to generalize to the case where it is necessary to satisfy both h_i and h_j.

6.4 Methods

Given trajectory samples $(\mathbf{X}, \mathbf{U}) \sim q^*(x_0, x_g, \theta)$, we use a conditional diffusion model $p_\psi(X, U|x_0, x_g, \theta)$ to learn a generative model of the data. We will first give an overview of diffusion models.

6.4.1 Diffusion Models

Diffusion probabilistic models [53, 141, 142] are a class of generative models that have been shown to be highly effective for learning distributions of trajectories [61, 2]. Given a dataset $\mathcal{D} = \{\tau\}^N$, the data samples are τ_0 and a predefined forward noising process $q(\tau_{k+1}|\tau_k) = \mathcal{N}(\sqrt{\alpha_k}\tau_k, (1-\alpha_k)\mathbf{I})$ is used to progressively add noise to the data for K steps, resulting in $\tau_1, ..., \tau_K$ increasingly noisy latent vectors. K and α_k are chosen such that $\tau_K \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$. To sample from the model, we use a trainable reverse process $p_\psi(\tau_{k-1}|\tau_k) = \mathcal{N}(\mu(\tau_k, k), \Sigma_k)$, where μ is parameterized by a neural network, and Σ_k is typically fixed, but can in principle be learned. Diffusion models are learned with the loss

$$\mathcal{L}(\psi) = \mathbb{E}_{k\sim[1,K],\tau_0\sim\mathcal{D},\tau_k\sim q(\tau_k|\tau_0),\epsilon\sim\mathcal{N}(\mathbf{0},\mathbf{I})}[||\epsilon - \epsilon_\psi(\tau_k, k)||^2] \tag{6.2}$$

where ϵ_ψ is a neural network. The mean of the reverse process μ is then calculated from this ϵ_ψ.

Classifier-free Guidance for Conditional Diffusion Models In the previous section, we described an unconditional diffusion model. However, we would like to generate trajectories conditioned on the start, goal, and constraints. For convenience, we label all contextual information as c, the dataset is then $\mathcal{D} = \{\tau, c\}^N$. We use a technique known as classifier-free guidance [52]. The diffusion model is modified to also take the context as an input $\epsilon(\tau_k, c, k)$. The loss then becomes

$$\begin{aligned}&\mathcal{L}(\psi) = \\ &\mathbb{E}_{k\sim[1,K],\tau_0,c\sim\mathcal{D},\tau_k\sim q(\tau_k|\tau_0),\epsilon\sim\mathcal{N}(\mathbf{0},\mathbf{I})}[||\epsilon - \epsilon_\psi(\tau_k, c, k)||^2].\end{aligned} \tag{6.3}$$

During training, with some dropout probability $p_{unconditional}$ we replace c with $\emptyset$, this effectively trains a conditional generative model and an unconditional generative model together. When sampling, the conditional and unconditional models are combined via

$$\hat{\epsilon}(\tau_k, c, k) = \epsilon_\psi(\tau_k, \emptyset, k) + \omega(\epsilon_\psi(\tau_k, c, k) - \epsilon_\psi(\tau_k, \emptyset, k)), \tag{6.4}$$

where ω controls the influence of the conditioning information. We can see that $\omega = 1$ corresponds to simply using $\hat{\epsilon} = \epsilon_\psi(\tau_k, c, k)$, ω is typically chosen to be larger than 1 to more strongly incorporate the conditioning information.

6.4.2 Composing Constraints

As introduced in the previous section, the conditional diffusion model is $\epsilon_\psi(\tau_k, c, k)$, where c is the contextual information. The contextual information is $\{x_0, x_g, \theta, \hat{y}\}$, where $\hat{y}$ is a one-hot encoding of y. We are interested in composing constraints, such that we can generalize to novel combinations of constraints that have not been seen together during training. Suppose we have L constraints, then the contextual information is $c = \{x_0, x_g, \theta_1, \hat{y}_1, ..., \theta_L, \hat{y}_L\}$. These are composed at test time via

$$\begin{aligned}\hat{\epsilon}(\tau_k, c, k) =& \epsilon_\psi(\tau_k, \emptyset, k) + \\ & \sum_{i=1}^{L} \omega_i(\epsilon_\psi(\tau_k, \{x_0, x_g, \theta_i, \hat{y}_i\}, k) - \epsilon_\psi(\tau_k, \emptyset, k)),\end{aligned} \tag{6.5}$$

where ω_i is a hyperparameter that controls the relative influence of each constraint.

6.4.3 Architecture

For the neural network architecture we use the 1-D convolutional U-Net described in [61]. We encode the start, goal, and constraint information with a multi-layer perceptron (MLP) to a $\mathbb{R}^{256}$ vector which is used to condition the network via Feature-wise Linear Modulation (FiLM) [117]. To query and train the unconditional diffusion

model, we replace this vector with the zero vector. We train with Adam and a learning rate of 1×10^{-4}.

6.4.4 Using the learned model for planning with CSVTO

In principle, a perfect generative model planning would simply consist of sampling from the diffusion model, as in [61, 2]. However, to ensure that trajectories satisfy the constraints we use the samples as the initialization for CSVTO. CSVTO starts from an initial set of particles and updates the particle set, driving the particles towards constraint satisfaction and low cost while also promoting diversity. Suppose we are running CSVTO with N particles. In the first time-step, we sample N trajectories from the generative model to get a set of initial trajectory samples. At subsequent timesteps, CSVTO gives N initial trajectories which are the shifted result of the previous time-steps optimization. We sample an additional N trajectories from the generative model and choose the best N of the $2N$ trajectories to serve as the initialization for the optimization.

6.5 Results

We evaluate our approach in three experiments. The first is a constrained 12DoF quadrotor task which has nonlinear underactuated dynamics. The second experiment is a 7DoF robot manipulator task where the aim is to move the robot end-effector to a goal location while being constrained to move along the surface of a table.

6.5.1 Ablations

We compare using our proposed composable learned generative model for trajectory optimization with ablations. The first is CSVTO without a learned generative model to sample trajectories from. For the second ablation we compare against using the learned diffusion model without the composability, e.g. for a task that consists of satisfying constraint h_1 and h_2, we compare using diffusion models which only take into account a single constraint.

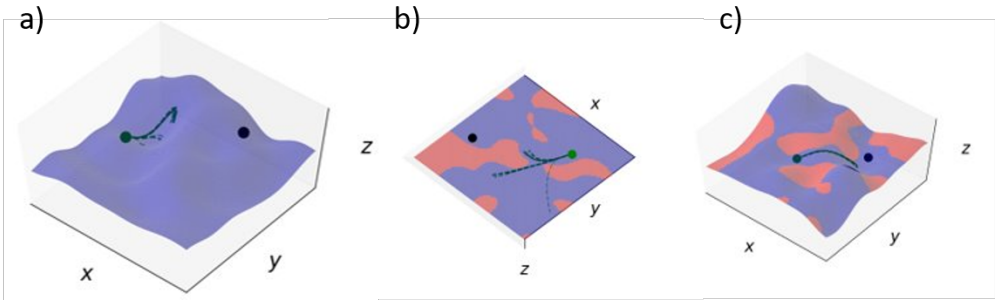


Figure 6.2: Experimental setup for the training and evaluation of the quadrotor tasks. The quadrotor must travel to the goal location. a) The quadrotor is constrained to travel along a non-linear surface shown in purple. b) The quadrotor must avoid the infeasible regions in the x-y plane shown in red. c) The quadrotor must satisfy both the previous constraints, avoiding infeasible regions while staying on the non-linear surface. The combination of these two constraints is not seen during training.

6.5.2 12DoF Quadrotor

For this task there are two types of constraints; a constraint that the quadrotor must travel along a nonlinear surface $z = f_{surf}(x, y)$, and that the quadrotor must avoid obstacles in the x-y plane, with in-collision configurations described by $f_{obs}(x, y) < 0$. To generate different versions of each of these two types of constraints, we sample f_{obs} and f_{surf} from a Gaussian Process prior with an RBF kernel and zero mean function. To do this, we sample 10×10 function evaluations on a 10×10 x-y grid. We then fit a GP to these function evaluations and use the posterior mean of this GP as the constraint function. The constraint is then parameterized by the 10×10 function values. We collect a dataset that consists of trajectories that satisfy either the surface constraint or the obstacle avoidance constraint. To collect the dataset, we generate 10000 surface constraints and 10000 obstacle constraints. For each constraint we run CSVTO with 16 particles and generate trajectories for 10 different starts and goals, resulting in $20000 \times 16 \times 10 = 3.2 \times 10^6$ trajectories. We evaluate with an unseen setup in which the quadrotor must satisfy both the obstacle constraint and the surface constraint at the same time. Examples of the training and evaluation set-ups are shown in Figure 6.2.

We run this experiment for 20 trials with randomly sampled starts, goals, surface,

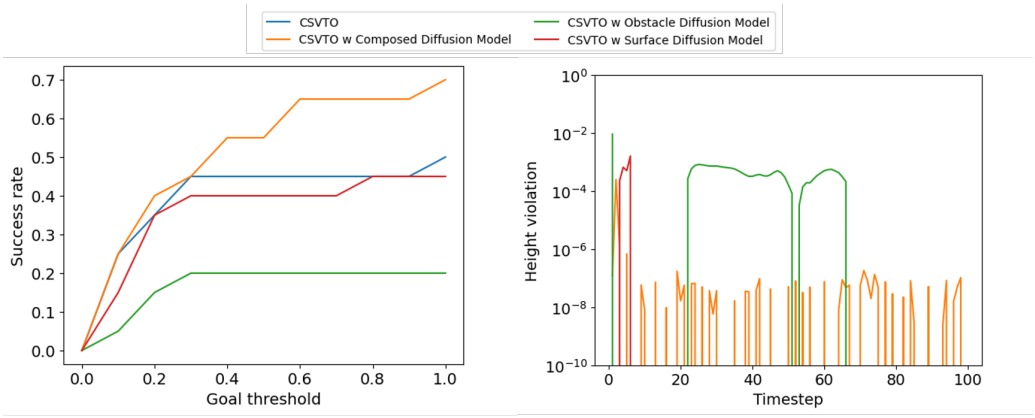


Figure 6.3: Results for quadrotor experiments. The left row shows the success rate as we increase the size of the goal region. The right shows the average constraint violation as a function of time

and obstacle constraints. The results are shown in Figure 6.3. CSVTO with the composed diffusion outperforms the ablations, achieving succeeding 13/20 times at a goal threshold of 0.6m, compared with 9/20 for CSVTO with no diffusion, the next best baseline. We see that CSVTO with a diffusion model that only takes into account the surface constraint performs similarly to CSVTO with no diffusion, whereas CSVTO which only takes into account the obstacle constraint performs significantly worse.

6.5.3 Robot Manipulator on Surface

For this experiment, we use the same experimental setup as that described in 6.5.3. In this task, there are again two types of constraints, a surface constraint that the end-effector must be constrained to along the surface of the table, and an obstacle constraint that the end-effector must avoid two cylindrical obstacles in the x-y plane. For the obstacle constraint, $\theta \in \mathbb{R}^4$ is the x-y positions of the center of both obstacles, while for the surface constraint, $\theta = [h, 0, 0, 0]$, where h is the height of the table. We collect a dataset that consists of trajectories that satisfy either the table constraint or the obstacle avoidance constraint. To collect the dataset, we generate 10000 surface constraints and 10000 obstacle constraints. For each constraint we run CSVTO with 16 particles and generate trajectories for 10 different starts and goals,

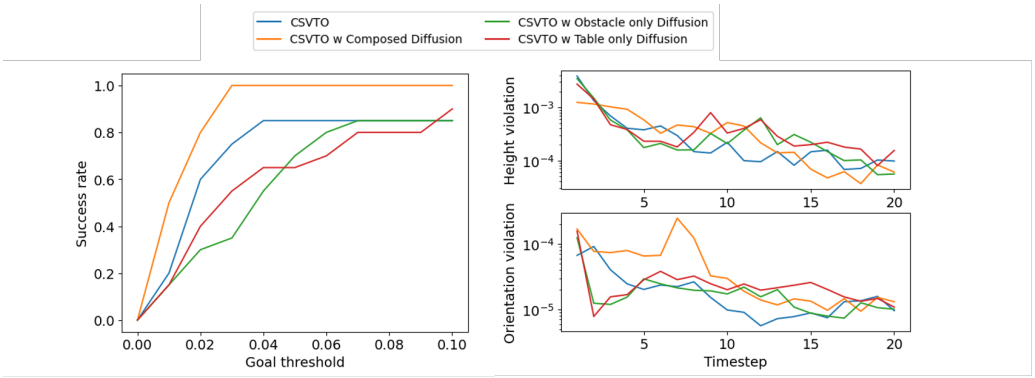


Figure 6.4: Results for manipulator table experiments. The left row shows the success rate as we increase the size of the goal region. The right shows the average constraint violation as a function of time

resulting in $20000 \times 16 \times 10 = 3.2 \times 10^6$ trajectories. We evaluate our approach in a scenario in which the robot must satisfy both constraints at the same time.

We run this experiment for 20 trials with randomly sampled starts, goals, surface, and obstacle constraints. The results are shown in Figure 6.4. CSVTO with the composed diffusion outperforms the ablations, achieving succeeding 20/20 times at a goal threshold of 0.04m, compared with 17/20 for CSVTO with no diffusion, the next best baseline. We see that CSVTO with a diffusion model that only takes into account one of the two constraints performs slightly worse than CSVTO. Examples of trajectories generated by all methods are shown in Figure 6.1.

6.6 Challenges and Future Work

6.6.1 Generalizing constraints

While our work shows the benefit of learning composable diffusion models of constraint-satisfying trajectories, we have so far only demonstrated these on fairly limited constraints for manipulation planning. For the 7DoF manipulation task, we only consider an SE(2) end-effector constraint at a fixed height, and obstacle constraint that consists of ensuring the end-effector avoids 2D obstacles in the plane. We intend to extend the generality of the constraints considered; the 2D obstacle

constraint can be extended to avoiding 3D obstacles with the full 7DoF arm, and the SE(2) end-effector constraint can be extended to Task-space regions (TSRs) [14]. With these more general constraint parameterizations a single diffusion model may be able to be used to generate constraint-satisfying trajectories for diverse tasks such as turning a wrench, wiping a cluttered table, or opening a cabinet door, all in the presence of 3D obstacles.

6.6.2 Improving constraint-satisfaction

Currently our learned diffusion models are trained on constraint-satisfying trajectories. This means they approximately, but do not exactly satisfy the constraints. This necessitates using a post-processing step to enforce constraint satisfaction. Our preliminary results show that using CSVTO results in good performance. However, we intend to further improve the constraint satisfaction of the sampled trajectories. Sampling from diffusion models is an iterative process, where trajectories are iteratively updated with the output of the neural network. We intend to investigate if the techniques used in CSVTO to update trajectories toward constraint satisfaction can be applied to the iterative sampling in a diffusion model. The current diffusion model we are using uses a fixed number of diffusion timesteps, which cannot be changed after training the model. This presents a challenge, as if we aim to enforce constraint satisfaction during sampling we only have a fixed number of samples in which to do so. One possible route to alleviating this issue is to consider continuous-time Score-based Models based on Stochastic Differential Equations (SDEs), which are a generalization of diffusion models. Using these methods we may be able to adaptively change the number of steps when sampling. Another challenge is that when sampling from diffusion models, the initial samples are pure Gaussian noise. During training, the diffusion model may learn a path from Gaussian noise to a low-cost constraint satisfaction that traverses a highly infeasible region of the search space. If during sampling, we project to constraint satisfaction too early, we may find that the trajectory leaves the learned diffusion path and is effectively OOD for the diffusion model.

CHAPTER VII

Conclusion and Outlook

In this thesis, we first introduced a method for planning for manipulation with a highly miss-specified model from image observations, which relies on learning a Gaussian Process to estimate the model uncertainty (Chapter II). We afterwards presented a method that learns a generative model of control sequences to complete a given task. We also demonstrated that we can adapt this generative model to Out-of-Distribution (OOD) environments (Chapter III). We then presented a method that views constrained trajectory optimization as inference and generates diverse sets of constraint-satisfying trajectories for completing manipulation tasks. By generating diverse sets of trajectories we demonstrated that we are better able to adapt to online disturbances since at any given time we have a set of trajectories to select from (Chapter IV). We additionaly demonstrated the application of this method to challenging dexterous manipulation tasks with a multi-fingered hand (Chapter V). Finally, in Chapter VI we proposed learning a generative model of constraint-satisfying trajectories for manipulation planning which are used to initialize the trajectory optimization method introduced in Chapter IV.

7.1 Future Work

7.1.1 Simple Model Reductions for Manipulation Planning

The work in Chapter II assumes that a single suitable simple model reduction is given a-priori for the task. In general this will not be the case, and an interesting avenue of future work is identifying simple models. In addition, the method assumes that a single simple model is sufficient to complete a given task, which may not be true in general. For instance, consider the problem of tidying an open book. First, the robot should close the book, for which it could model the book as an articulated rigid object. Next, the robot should place the book on a bookshelf. If the book is closed and remains so, an appropriate simple model for this task could be to treat the book as a rigid object. An interesting future direction would be to explore how a robot can autonomously select from a library of simple models in order to complete long horizon tasks.

Another key limitation of the method presented in Chapter II is that it restricts the robot to act in concordance with the simple model; the robot is unable to exploit useful behavior of the underlying system if that behavior contradicts the simple model. For instance, for a robot routing a cable through a narrow gap, the ability of the cable to conform around obstacles in the vicinity of the gap may be helpful for accomplishing the task. If the robot is using a rigid body simple model, this behavior will be penalized for contradicting the simple model. An interesting direction for future work could investigate how to reduce this limitation. For instance, the simple model approximation could be used to bootstrap an initial learning phase, and then a more flexible learning method could then exploit the true dynamics of the underlying system.

7.1.2 Planning & Continual Learning:

In Chapters III and VI we proposed learning generative models for trajectories for the purpose of planning. In both cases, we first collected a large dataset, then trained on this dataset, and then deployed the resulting model. However, in real-

world robotics the tasks and environments are never completely static - objects may move in the environment, lighting and weather conditions change, and wear and tear can change the robot's dynamics. A general-purpose robot will need to continually learn and update its models as it is deployed in its environment. In future work, an interesting direction is exploring continual learning methods in the context of learning withing planning frameworks. This will necessitate developing planners that track changing conditions, and determine when, what, and how new data should be collected.

7.1.3 Integrating Planning, Perception & Learning:

To make robots more reliable in unstructured environments, they must be able to make decisions under uncertainty. A central source of uncertainty is perception uncertainty. In Chapter II, we incorporated the uncertainty of learned perception components when performing MPC. However, there is more work that could be done on this integration. In particular, investigating methods that closely couple planning, perception, and learning. Such methods will incorporate perception uncertainty into the planning process, as well as planning to reduce perception uncertainty, perceive when learning is necessary, and plan to collect the appropriate data. For example, a robot manipulating a novel object may detect that it is uncertain about the object, determine that it needs a new view of the object, and plan a trajectory that moves the object to this new view. A central challenge in this work will be how to ensure tractability when solving these high-dimensional joint problems. Gradient-based particle methods such as those used in Chapter IV are a promising direction for this.

APPENDICES

APPENDIX A

Appendix for Chapter 3: Learning a Generalizable Trajectory Sampling Distribution for Model Predictive Control

A.1 Training & Architecture Details

A.1.1 Hyperparameter Tuning

There are several hyperparameters to tune in our approach. The scalar a in equation 3.13 was tuned so that $a\mathcal{L}_{VAE}$ and $\mathcal{L}_{flow}$ were of approximately similar magnitude. The scalar b in equation 3.16 was selected to be equal to the dimensionality of the SDF observation divided by the dimensionality of the latent environment embedding. This value was chosen initially to make the projection loss similar across the quadcopter and the double integrator, and we found this automatic tuning worked well in practice. Hyperparameters α, β together control the trade-off between entropy and optimality, and we tuned these via a grid search and selected the values that resulted in the best performance in the training environment when used with FlowMPPI.

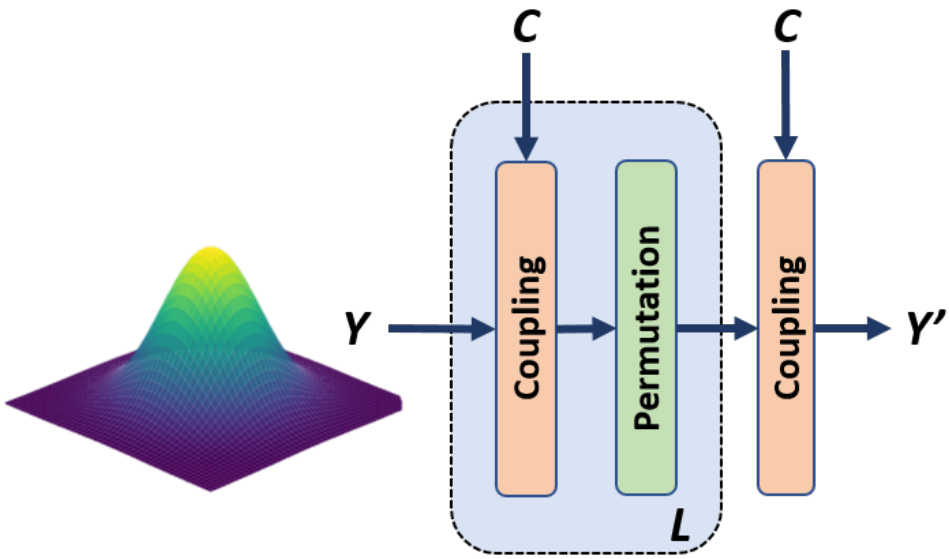


Figure A.1: The architecture for both the prior flow and the control sequence posterior flow, based on [33] and [168], showing a mapping from arbitrary Y to Y'. Each flow consists of L chained transformation blocks. A transformation block consists of a coupling layer and a random permutation. There is a final conditional coupling layer on the output. For the vae prior, there is no context thus we use standard coupling layers and not conditional coupling layers.

Table A.1: Training and architecture hyperparameters

Variable	Planar Navigation	3D 12DoF Quadrotor	7DoF Manipulator
control Σ_e	N/A	N/A	$0.5(1 - \frac{\text{epoch}}{\#\text{epochs}})$
α	2.5×10^{-3}	2.5×10^{-3}	$\frac{\#\text{ epochs}}{500\text{epoch}}$
β	$\frac{\#\text{ epochs}}{400\text{epoch}}$	$\frac{\#\text{ epochs}}{400\text{epoch}}$	1
# epochs	1000	2000	1000
Init. learn rate	1×10^{-4}	1×10^{-4}	1×10^{-3}
# Training envs.	10000	20000	20000
# (x_0, x_G) per training env.	100	100	100
h dim	256	256	256
a	5	5	5
b	$\frac{5}{16}$	$\frac{5}{1024}$	$\frac{1}{1024}$
VAE train epochs	100	100	200
$p_\phi(h)$ depth	4	4	4
f_ζ depth	12	12	20

A.1.2 Environment details

The environments are $4m \times 4m$ for the planar navigation task, $4m \times 4m \times 4m$ for the 12 DoF quadrotor, and $1.5m \times 1.5m \times 1.5m$ for the 7DoF manipulator. The environments are generated as occupancy grids, from which we compute the SDF. For each training environment, we randomly sample 100 collision free start & goal pairs. We sample start velocities from a Normal distribution, and set the goal velocity to be zero. During evaluation, for both the in-distribution and out-of-distribution environments, we sample 100 start, goal and environment tuples and evaluate all methods on these tuples. The exception to this is the real-world environments, where we keep the environments fixed and sample 100 start and goal pairs per real-world environment and evaluate all methods on these pairs. To ensure the navigation problem is non-trivial, we sample starts and goals that are at least $4m$ away.

A.1.2.1 Real-world environments

The two real-world environments are taken from area 3 from the 2D-3D-S dataset [8]. To generate the two environments, we used the 3D mesh from the dataset and defined a subset of the area to be the environment. We then generated an occupancy grid by densely sampling the mesh, which we then used to compute the SDF.

Table A.2: Controller agnostic parameters used for the evaluations

Variable	Planar Navigation	12DoF Quadrotor	7Dof Manipulator
Control Horizon H	40	40	40
Trial length T	100	100	100
Dynamics Δt	0.05	0.025	0.025

Table A.3: Controller hyperparameters used for the experiments for both our proposed method and the baselines

Controller	Variable	Planar Navigation	12DoF Quadrotor	7DoF Manipulator
MPPI	λ	1	1	1
	Σ	1	0.25	0.25
	iterations	1	4	4
SVMPC	Σ	0.5	0.5	1
	# particles	4	4	4
	Learning rate	0.1	0.1	1
	iterations	4	4	4
	warm-up iterations	25	25	25
iCEM	Σ	0.75	0.5	0.5
	noise parameter	2.5	3	3
	% elites	0.1	0.1	0.1
	% kept elites	0.3	0.5	0.5
	iterations	4	4	4
	momentum m	0.1	0.1	0.1
FlowMPPI	λ	1	1	1
	Σ	1	0.5	0.25
	iterations	1	2	4
FlowiCEM	Σ	0.75	0.5	0.5
	noise parameter	2.5	3	3
	% elites	0.1	0.1	0.1
	% kept elites	0.5	0.3	0.5
	iterations	4	4	4
	momentum m	0.1	0.1	0.1
Projection	M	10	10	10
	Proj. learn. rate	2×10^{-3}	$2{\times}10^{-3}$	$1{\times}10^{-2}$

A.1.3 Planar Navigation

The dynamics for the planar navigation system are

$$\begin{bmatrix} x \\ y \\ \dot{x} \\ \dot{y} \end{bmatrix}_{t+1} = \begin{bmatrix} 1 & 0 & \Delta t & 0 \\ 0 & 1 & 0 & \Delta t \\ 0 & 0 & 0.95 & 0 \\ 0 & 0 & 0 & 0.95 \end{bmatrix} \begin{bmatrix} x \\ y \\ \dot{x} \\ \dot{y} \end{bmatrix}_{t} + \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ \Delta t & 0 \\ 0 & \Delta t \end{bmatrix} \mathbf{u} \tag{A.1}$$

A.1.4 12DoF Quadrotor

The dynamics for the 12DoF quadrotor are from [135] and are given by

$$\begin{bmatrix} x \\ y \\ z \\ p \\ q \\ r \\ \dot{x} \\ \dot{y} \\ \dot{z} \\ \dot{p} \\ \dot{q} \\ \dot{r} \end{bmatrix}_{t+1} = \begin{bmatrix} x \\ y \\ z \\ p \\ q \\ r \\ \dot{x} \\ \dot{y} \\ \dot{z} \\ \dot{p} \\ \dot{q} \\ \dot{r} \end{bmatrix}_{t} + \Delta t \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \\ \dot{p} + \dot{q}s(p)t(q) + \dot{r}c(p)t(q) \\ \dot{q}c(p) - \dot{r}s\dot{p} \\ \dot{q}\frac{s(p)}{c(q)} + \dot{r}\frac{c(p))}{c(q)} \\ -(s(p)s(r) + c(r)c(p)s(q))K\frac{u_1}{m} \\ -(c(r)s(p) - c(p)s(r)s(q))K\frac{u_1}{m} \\ g - c(p)s(q))K\frac{u_1}{m} \\ \frac{(I_y - I_z)\dot{q}\dot{r} + Ku_2}{I_x} \\ \frac{(I_z - I_x)\dot{p}\dot{r} + Ku_3}{I_y} \\ \frac{(I_x - I_y)\dot{p}\dot{q} + Ku_4}{I_z} \end{bmatrix}_{t} \tag{A.2}$$

Where $c(p), s(p), t(p)$ are $\cos, \sin, \tan$ functions respectively. We use a parameters $m = 1, I_x = 0.5, I_y = 0.1, I_z = 0.3, K = 5, g = -9.81$. The quadrotor geometry is modeled as a cylinder with radius $0.1m$ and height $0.05m$.

A.1.5 7DoF Manipulator

We use a kinematic model of the 7DoF manipulator

$$q = q + u\Delta t \tag{A.3}$$

Where q is the robot joint configuration and u are the controls.

A.2 Algorithms

Algorithm 8 Sample from $q(U|C)$

function SAMPLEU(C, K)
  for i $\in \{k, ..., K\}$ **do**
    $Z_k \sim \mathcal{N}(0, I)$
    $U_k \leftarrow f_\zeta(Z_k, C)$
    $q_\zeta(U_k|C) \leftarrow$ from $\hat{Z}_k$ via eq. (3.7)
  return $\{U_k, q_\zeta(U_k|C)\}_{k=1}^K$

Algorithm 9 Sample from $q(U|C)$ with Perturbation

function SAMPLEPERTU(C, Σ_ϵ, K)
  for i $\in \{k, ..., K\}$ **do**
    if $\Sigma_\epsilon = 0$ **then**
      return SAMPLEU
    $Z_k \sim \mathcal{N}(0, I)$
    $\epsilon_k \sim \mathcal{N}(0, \Sigma_\epsilon)$
    $U_k \leftarrow f_\zeta(Z_k, C) + \epsilon_k$
    $\hat{Z}_k \leftarrow f_\zeta^{-1}(U_k, C)$
    $q_\zeta(U_k|C) \leftarrow$ from $\hat{Z}_k$ via eq. (3.7)
  return $\{U_k, q_\zeta(U_k|C)\}_{k=1}^K$

Algorithm 10 Flow Training

Inputs: N iterations, K samples, $\Theta^1 = \{\theta^1, \psi^1, \phi^1, \omega^1, \zeta^1\}$ initial parameters, control perturbation covariance Σ_ϵ, learning rate η, loss hyperparameters (α, β)

for n $\in \{1, ..., N\}$ **do**
  $h \leftarrow \phi$
  $\hat{E} \leftarrow p_\psi(E|h)$
  Compute $\log p_\phi(h)$ via eq. (3.7)
  Compute $\mathcal{L}_{VAE}$
  $C \leftarrow g_\omega(x_0, x_G, \rho, h)$
  $\{U_k, q_\zeta(U_k|C)\}_{k=1}^K \leftarrow \textsc{SamplePertU}(C, \Sigma_\epsilon, K)$
  $\mathcal{L} \leftarrow \mathcal{L}_{VAE}$
  for k $\in \{1, ..., K\}$ **do**
    $w_k \leftarrow$ from $(\{U_i, \log q_\zeta(U_i|C)\}_{i=1}^K, \alpha, \beta\})$ via (3.11)
    $\mathcal{L} \leftarrow \mathcal{L} - w_k \cdot \log q_\zeta(U_k|C)$
  $\Theta^{n+1} \leftarrow \Theta^n - \eta \frac{\partial \mathcal{L}}{\partial \Theta}$

APPENDIX B

Appendix for Chapter 4: Constrained Stein Variational Trajectory Optimization

B.1 Matrix Derivative of $P(\tau)]$

In Equation (4.33) we showed that the repulsive gradient is split into two terms, one of which contains the matrix derivative $\nabla_{[\tau]_k} P(\tau)$. In this section, we show how to compute this derivative. For notational convenience, let $\tau \in \mathbb{R}^N$ (thus $P(\tau) \in \mathbb{R}^{N \times N}$), $h(\tau) \in \mathbb{R}^M$ (where M is the number of constraints), and we omit the dependence on τ when writing the constraint derivative $\nabla h(\tau)$. $\nabla_{[\tau]_k} P(\tau)$ is a matrix of shape $N \times N$. We refer to the second derivative of the lth constraint $\nabla^2 h_l(\tau)$ as H_l, which is an $N \times N$ matrix. The matrix derivative $\nabla_{[\tau]_k} P(\tau)$, as defined in Equation (4.33), can be expanded into three terms:

$$\nabla_{[\tau]_k} [P(\tau)]_{i,k} = 2A_{i,k} - B_{i,k}, \tag{B.1}$$

where $A, B \in \mathbb{R}^{N \times N}$ and $i, k \in \{1, ..., N\}$. $A_{i,k}$ is given by

$$A_{i,k} = \sum_{l}^{M} [H_l]_{k,i} [\left(\nabla h \nabla h^T\right)^{-1} \nabla h]_{l,k}. \tag{B.2}$$

To compute $B_{i,k}$, we first consider the matrix $D_k \in \mathbb{R}^{M \times M}$:

$$[D_k]_{l,m} = \sum_{j}^{N} \left([H_l]_{k,j} [\nabla h]_{l,j} + [H_m]_{j,k} [\nabla h]_{m,j} \right), \tag{B.3}$$

for $l, m \in \{1, ..., M\}$. We then finally compute $B_{i,k}$ as

$$\begin{aligned} &B_{i,k} = \\ &\sum_{l}^{M} \sum_{m}^{M} [D_k]_{l,m} [\nabla h^T \left(\nabla h \nabla h^T \right)^{-1}]_{i,l} [\left(\nabla h \nabla h^T \right)^{-1} \nabla h]_{m,k}. \end{aligned} \tag{B.4}$$

When neglecting second-order terms for the lth constraint $h_l(\tau)$ (as discussed in Section 4.7.1.1), we set $H_l = \mathbf{0}$ when computing $A_{i,k}$ and $B_{i,k}$.

BIBLIOGRAPHY

BIBLIOGRAPHY

[1] Agrawal, P., A. Nair, P. Abbeel, J. Malik, and S. Levine (2016), Learning to poke by poking: Experiential learning of intuitive physics, in *NeurIPS*.

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

[3] Amestoy, P. R., A. Guermouche, J.-Y. L'Excellent, and S. Pralet (2006), Hybrid scheduling for the parallel solution of linear systems, *Parallel Computing*, *32*(2), 136–156.

[4] Andersson, J. A. E., J. Gillis, G. Horn, J. B. Rawlings, and M. Diehl (2019), CasADi – A software framework for nonlinear optimization and optimal control, *Mathematical Programming Computation*, *11*(1), 1–36, doi: 10.1007/s12532-018-0139-4.

[5] Andrychowicz, O. M., et al. (2020), Learning dexterous in-hand manipulation, *The International Journal of Robotics Research*, *39*(1), 3–20.

[6] Apgar, T., P. Clary, K. Green, A. Fern, and J. W. Hurst (2018), Fast online trajectory optimization for the bipedal robot cassie, in *Robot.: Sci. Syst.*

[7] Armand, P., and D. Orban (2007), The squared slacks transformation in nonlinear programming, *Sultan Qaboos University Journal for Science [SQUJS]*, *17*.

[8] Armeni, I., S. Sax, A. R. Zamir, and S. Savarese (2017), Joint 2d-3d-semantic data for indoor scene understanding, arxiv.1702.01105.

[9] Attias, H. (2003), Planning by probabilistic inference, in *Proc. 9th Int. Workshop on Artificial Intelligence and Statistics*, pp. 9–16.

[10] Balci, I. M., E. Bakolas, B. Vlahov, and E. A. Theodorou (2022), Constrained covariance steering based tube-mppi, in *2022 American Control Conference (ACC)*, pp. 4197–4202.

[11] Banijamali, E., R. Shu, M. Ghavamzadeh, H. H. Bui, and A. Ghodsi (2018), Robust locally-linear controllable embedding, in *AISTATS*.

[12] Barcelos, L., A. Lambert, R. Oliveira, P. Borges, B. Boots, and F. Ramos (2021), Dual Online Stein Variational Inference for Control and Dynamics, in *Robot.: Sci. Syst.*

[13] Bechtle, S., Y. Lin, A. Rai, L. Righetti, and F. Meier (2019), Curious ilqr: Resolving uncertainty in model-based rl, in *CoRL*.

[14] Berenson, D., S. Srinivasa, and J. Kuffner (2011), Task space regions: A framework for pose-constrained manipulation planning, *Int. J. Rob. Res.*, *30*(12), 1435–1460.

[15] Bhardwaj, M., B. Sundaralingam, A. Mousavian, N. D. Ratliff, D. Fox, F. Ramos, and B. Boots (2022), Storm: An integrated framework for fast joint-space model-predictive control for reactive manipulation, in *Proc. Conf. Robot. Learn.*, pp. 750–759.

[16] Blei, D. M., A. Kucukelbir, and J. D. McAuliffe (2017), Variational inference: A review for statisticians, *J. Am. Stat. Assoc.*, *112*(518), 859–877.

[17] Bonalli, R., A. Cauligi, A. Bylard, and M. Pavone (2019), Gusto: Guaranteed sequential trajectory optimization via sequential convex programming, in *Proc. IEEE Int. Conf. Robot. Autom.*, pp. 6741–6747.

[18] Brasseur, C., A. Sherikov, C. Collette, D. Dimitrov, and P.-B. Wieber (2015), A robust linear mpc approach to online generation of 3d biped walking motion, in *Proc. 15th IEEE-RAS Int. Conf. Humanoid Robots*, p. 595–601.

[19] Byravan, A., F. Leeb, F. Meier, and D. Fox (2018), SE3-Pose-Nets: Structured Deep Dynamics Models for Visuomotor Control, in *ICRA*.

[20] Calli, B., A. Singh, A. Walsman, S. Srinivasa, P. Abbeel, and A. M. Dollar (2015), The ycb object and model set: Towards common benchmarks for manipulation research, in *Int. Conf. on Advanced Robotics*, pp. 510–517, IEEE.

[21] Carius, J., R. Ranftl, F. Farshidian, and M. Hutter (2022), Constrained stochastic optimal control with learned importance sampling: A path integral approach, *Int. J. Rob. Res.*, *41*(2), 189–209.

[22] Chebotar, Y., A. Handa, V. Makoviychuk, M. Macklin, J. Issac, N. Ratliff, and D. Fox (2019), Closing the sim-to-real loop: Adapting simulation randomization with real world experience, in *ICRA*.

[23] Chen, C., P. Culbertson, M. Lepert, M. Schwager, and J. Bohg (2021), Trajectotree: Trajectory optimization meets tree search for planning multi-contact dexterous manipulation, *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 8262–8268.

[24] Chen, X., D. P. Kingma, T. Salimans, Y. Duan, P. Dhariwal, J. Schulman, I. Sutskever, and P. Abbeel (2017), Variational lossy autoencoder, in *Proc. Int. Conf. Learn. Representations*.

[25] Cheng, X., E. Huang, Y. Hou, and M. T. Mason (2021), Contact mode guided sampling-based planning for quasistatic dexterous manipulation in 2d, in *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 6520–6526.

[26] Cheng, X., E. Huang, Y. Hou, and M. T. Mason (2022), Contact mode guided motion planning for quasidynamic dexterous manipulation in 3d, in *2022 International Conference on Robotics and Automation (ICRA)*, pp. 2730–2736.

[27] Cheng, X., S. Patil, Z. Temel, O. Kroemer, and M. T. Mason (2024), Enhancing dexterity in robotic manipulation via hierarchical contact exploration, *IEEE Robotics and Automation Letters*, *9*(1), 390–397.

[28] Chi, C., S. Feng, Y. Du, Z. Xu, E. Cousineau, B. Burchfiel, and S. Song (2023), Diffusion policy: Visuomotor policy learning via action diffusion, in *Proceedings of Robotics: Science and Systems (RSS)*.

[29] Chua, K., R. Calandra, R. McAllister, and S. Levine (2018), Deep reinforcement learning in a handful of trials using probabilistic dynamics models, in *NeurIPS*.

[30] D'Angelo, F., and V. Fortuin (2021), Annealed stein variational gradient descent, arxiv.2101.09815.

[31] Deisenroth, M., and C. Rasmussen (2011), Pilco: A model-based and data-efficient approach to policy search, in *ICML*.

[32] Dhariwal, P., and A. Q. Nichol (2021), Diffusion models beat GANs on image synthesis, in *Advances in Neural Information Processing Systems*, edited by A. Beygelzimer, Y. Dauphin, P. Liang, and J. W. Vaughan.

[33] Dinh, L., J. Sohl-Dickstein, and S. Bengio (2017), Density estimation using real NVP, in *Proc. Int. Conf. Learn. Representations*.

[34] Du, Y., C. Durkan, R. Strudel, J. B. Tenenbaum, S. Dieleman, R. Fergus, J. Sohl-Dickstein, A. Doucet, and W. S. Grathwohl (2023), Reduce, reuse, recycle: Compositional generation with energy-based diffusion models and MCMC, in *Proceedings of the 40th International Conference on Machine Learning*, pp. 8489–8510.

[35] Ebert, F., C. Finn, S. Dasari, A. Xie, A. Lee, and S. Levine (2018), Visual foresight: Model-based deep reinforcement learning for vision-based robotic control, *arXiv preprint 1812.00568*.

[36] Farshidian, F., and J. Buchli (2015), Risk sensitive, nonlinear optimal control: Iterative linear exponential-quadratic optimal control with gaussian noise, *arXiv preprint: 1512.07173*.

[37] Feng, S., E. Whitman, X. Xinjilefu, and C. G. Atkeson (2014), Optimization based full body control for the atlas robot, in *Humanoids*.

[38] Feng, Y., D. J. X. Ng, and A. Easwaran (2021), Improving variational autoencoder based out-of-distribution detection for embedded real-time applications, *ACM Trans. Embed. Comput. Syst.*, *20*(5s).

[39] Feppon, F., Allaire, G., and Dapogny, C. (2020), Null space gradient flows for constrained optimization with applications to shape optimization, *ESAIM: COCV*, *26*, 90.

[40] Finean, M. N., W. Merkt, and I. Havoutis (2021), Predicted composite signed-distance fields for real-time motion planning in dynamic environments, *Proceedings of the International Conference on Automated Planning and Scheduling*, *31*, 616–624.

[41] Finn, C., and S. Levine (2016), Deep visual foresight for planning robot motion, *ICRA*.

[42] Fleming, W. H., and S. K. Mitter (1982), Optimal control and nonlinear filtering for nondegenerate diffusion processes, *Stochastics*, *8*(1), 63–77.

[43] Frigola, R., Y. Chen, and C. E. Rasmussen (2014), Variational gaussian process state-space models, in *NeurIPS*.

[44] Gardner, J. R., G. Pleiss, D. Bindel, K. Q. Weinberger, and A. G. Wilson (2018), Gpytorch: Blackbox matrix-matrix gaussian process inference with gpu acceleration, in *NeurIPS*.

[45] Giftthaler, M., and J. Buchli (2017), A projection approach to equality constrained iterative linear quadratic optimal control, in *Proc. 17th IEEE-RAS Int. Conf. Humanoid Robotics*, p. 61–66.

[46] Gill, P. E., W. Murray, and M. A. Saunders (2005), Snopt: An sqp algorithm for large-scale constrained optimization, *SIAM Rev.*, *47*(1), 99–131.

[47] Gordon, E. K., and R. S. Zarrin (2023), Online augmentation of learned grasp sequence policies for more adaptable and data-efficient in-hand manipulation, in *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 5970–5976.

[48] Ha, J.-S., D. Driess, and M. Toussaint (2020), A probabilistic framework for constrained manipulations and task and motion planning under uncertainty, in *Proc. IEEE Int. Conf. Robot. Autom.*, pp. 6745–6751, IEEE.

[49] Hafner, D., T. Lillicrap, I. Fischer, R. Villegas, D. Ha, H. Lee, and J. Davidson (2019), Learning latent dynamics for planning from pixels, in *ICML*.

[50] Hafner, D., T. P. Lillicrap, J. Ba, and M. Norouzi (2020), Dream to control: Learning behaviors by latent imagination, in *ICLR*.

[51] Hauser, J., and A. Saccon (2006), A barrier function method for the optimization of trajectory functionals with constraints, in *IEEE 45th Conf. Decision and Control*, pp. 864–869.

[52] Ho, J., and T. Salimans (2022), Classifier-free diffusion guidance.

[53] Ho, J., A. Jain, and P. Abbeel (2020), Denoising diffusion probabilistic models, in *Proceedings of the 34th International Conference on Neural Information Processing Systems*.

[54] Hol, J. D., T. B. Schon, and F. Gustafsson (2006), On resampling algorithms for particle filters, in *2006 IEEE Nonlinear Statistical Signal Processing Workshop*, pp. 79–82.

[55] Horowitz, M. B., and J. W. Burdick (2012), Combined grasp and manipulation planning as a trajectory optimization problem, in *2012 IEEE International Conference on Robotics and Automation*, pp. 584–591.

[56] Howell, T. A., B. E. Jackson, and Z. Manchester (2019), Altro: A fast solver for constrained trajectory optimization, in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, pp. 7674–7679.

[57] Ichter, B., J. Harrison, and M. Pavone (2018), Learning sampling distributions for robot motion planning, in *Proc. IEEE Int. Conf. Robot. Autom.*, pp. 7087–7094.

[58] Jaillet, L., and J. M. Porta (2013), Path planning under kinematic constraints by rapidly exploring manifolds, *IEEE Trans. Robot.*, *29*(1), 105–117.

[59] James, S., A. J. Davison, and E. Johns (2017), Transferring end-to-end visuomotor control from simulation to real world for a multi-stage task, in *CoRL*.

[60] Jankowiak, M., G. Pleiss, and J. R. Gardner (2020), Parametric gaussian process regressors, in *ICML*.

[61] Janner, M., Y. Du, J. Tenenbaum, and S. Levine (2022), Planning with diffusion for flexible behavior synthesis, in *International Conference on Machine Learning*.

[62] Ji, X., and J. Xiao (2001), Planning motions compliant to complex contact states, *The International Journal of Robotics Research*, *20*(6), 446–465.

[63] Jiang, Y., M. Yu, X. Zhu, M. Tomizuka, and X. Li (2024), Contact-implicit model predictive control for dexterous in-hand manipulation: A long-horizon and robust approach.

[64] Jongen, H. T., and O. Stein (2004), Constrained global optimization: Adaptive gradient flows, in *Frontiers in Global Optimization*, edited by C. A. Floudas and P. Pardalos, p. 223–236.

[65] Kalman, R. E. (1960), A New Approach to Linear Filtering and Prediction Problems, *J. Basic Eng.*, *82*(1), 35–45.

[66] Kappen, H., and H.-C. Euler (2016), Adaptive importance sampling for control and inference, *J. Stat. Phys.*, *162*, 1244–1266.

[67] Kappen, H. J. (2005), Linear theory for control of nonlinear stochastic systems, *Phys. Rev. Lett.*, *95*, 200,201.

[68] Kappen, H. J., V. Gómez, and M. Opper (2012), Optimal control as a graphical model inference problem, *Mach. Learn.*, *87*(2), 159–182.

[69] Karaman, S., and E. Frazzoli (2011), Sampling-based algorithms for optimal motion planning, *The International Journal of Robotics Research*, *30*(7), 846–894.

[70] Karaman, S., and E. Frazzoli (2011), Sampling-based algorithms for optimal motion planning, *Int. J. Rob. Res.*, *30*(7), 846–894.

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

[72] Kicki, P., P. Liu, D. Tateo, H. Bou-Ammar, K. Walas, P. Skrzypczyński, and J. Peters (2023), Fast kinodynamic planning on the constraint manifold with deep neural networks, arxiv.2301.04330.

[73] Kingma, D. P., and P. Dhariwal (2018), Glow: Generative flow with invertible 1x1 convolutions, in *Proc. Int. Conf. Neural Information Processing Systems*.

[74] Kingma, D. P., and M. Welling (2014), Auto-encoding variational bayes, in *Proc. Int. Conf. Learn. Representations*.

[75] Kirichenko, P., P. Izmailov, and A. G. Wilson (2020), Why normalizing flows fail to detect out-of-distribution data, in *Proc. Int. Conf. Neural Information Processing Systems*.

[76] Ko, J., and D. Fox (2009), Gp-bayesfilters: Bayesian filtering using gaussian process prediction and observation models, *AuRo*, *27*, 75–90.

[77] Kobilarov, M. (2012), Cross-entropy motion planning, *Int. J. Rob. Res.*, *31*(7), 855–871.

[78] Kousik, S., P. Holmes, and R. Vasudevan (2019), Safe, Aggressive Quadrotor Flight via Reachability-Based Trajectory Design, in *DSCC*.

[79] Kuffner, J., and S. LaValle (2000), 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*, vol. 2, pp. 995–1001 vol.2.

[80] Kumar, V., E. Todorov, and S. Levine (2016), Optimal control with learned local models: Application to dexterous manipulation, in *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 378–383.

[81] Lai, T., W. Zhi, T. Hermans, and F. Ramos (2021), Parallelised diffeomorphic sampling-based motion planning, in *Proc. Conf. Robot. Learn.*, pp. 81–90.

[82] Lakshminarayanan, B., A. Pritzel, and C. Blundell (2017), Simple and scalable predictive uncertainty estimation using deep ensembles, in *NeurIPS*.

[83] Lambert, A., and B. Boots (2021), Entropy regularized motion planning via stein variational inference, arxiv.2107.05146.

[84] Lambert, A., A. Fishman, D. Fox, B. Boots, and F. Ramos (2020), Stein variational model predictive control, in *Proc. Conf. Robot Learn.*

[85] Lambert, A., B. Hou, R. Scalise, S. S. Srinivasa, and B. Boots (2022), Stein variational probabilistic roadmaps, in *Proc. IEEE Int. Conf. Robot. Autom.*, pp. 11,094–11,101.

[86] Laskin, M., A. Srinivas, and P. Abbeel (2020), Curl: Contrastive unsupervised representations for reinforcement learning, *ICML*.

[87] Lavalle, S. M. (2006), *Planning Algorithms*, Cambridge University Press.

[88] Lembono, T. S., E. Pignat, J. Jankowski, and S. Calinon (2021), Learning constrained distributions of robot configurations with generative adversarial network, *IEEE Robotics and Automation Letters*, *6*(2), 4233–4240.

[89] Li, L., Y. Miao, A. H. Qureshi, and M. C. Yip (2021), Mpc-mpnet: Model-predictive motion planning networks for fast, near-optimal planning under kinodynamic constraints, arxiv.2101.06798.

[90] Li, W., and E. Todorov (2004), Iterative linear quadratic regulator design for nonlinear biological movement systems, in *Proc. IEEE Int. Conf. Informat. Control. Autom. Robot.*

[91] Li, Y., Z. Littlefield, and K. E. Bekris (2016), Asymptotically optimal sampling-based kinodynamic planning, *Int. J. Rob. Res.*, *35*(5), 528–564.

[92] Liu, C. K. (2009), Dextrous manipulation from a grasping pose, *ACM Trans. Graph.*, *28*(3).

[93] Liu, D. C., and J. Nocedal (1989), On the limited memory bfgs method for large scale optimization, *Mathematical programming*, *45*(1-3), 503–528.

[94] Liu, N., S. Li, Y. Du, A. Torralba, and J. B. Tenenbaum (2022), Compositional visual generation with composable diffusion models, in *ECCV 2022*, edited by S. Avidan, G. Brostow, M. Cissé, G. M. Farinella, and T. Hassner, pp. 423–439.

[95] Liu, Q. (2017), Stein variational gradient descent as gradient flow, in *Proc. Int. Conf. Neural Information Processing Systems*, edited by I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett.

[96] Liu, Q., and D. Wang (2016), Stein variational gradient descent: A general purpose bayesian inference algorithm, in *Proc. Int. Conf. Neural Information Processing Systems*, p. 2378–2386.

[97] Liu, Q., J. D. Lee, and M. Jordan (2016), A kernelized stein discrepancy for goodness-of-fit tests, in *Proc. Int. Conf. Mach. Learn.*, p. 276–284.

[98] Loew, T., T. Bandyopadhyay, J. Williams, and P. Borges (2021), Prompt: Probabilistic motion primitives based trajectory planning, in *Robot.: Sci. Syst.*

[99] Makoviychuk, V., et al. (2021), Isaac gym: High performance gpu-based physics simulation for robot learning, arxiv.2108.10470.

[100] Marcucci, T., R. Deits, M. Gabiccini, A. Bicchi, and R. Tedrake (2017), Approximate hybrid model predictive control for multi-contact push recovery in complex environments, in *2017 IEEE-RAS 17th International Conference on Humanoid Robotics (Humanoids)*, pp. 31–38, doi:10.1109/HUMANOIDS.2017.8239534.

[101] Mayne, D. Q. (1966), A second-order gradient method for determining optimal trajectories of non-linear discrete-time systems, *Int. J. Control*, *3*, 85–95.

[102] Mcconachie, D., and D. Berenson (2018), Estimating model utility for deformable object manipulation using multiarmed bandit methods, *T-ASE*, *15*(3), 967–979.

[103] McConachie, D., T. Power, P. Mitrano, and D. Berenson (2020), Learning when to trust a dynamics model for planning in reduced state spaces, *RA-L*, *5*(2), 3540–3547.

[104] Miller, S., J. van den Berg, M. Fritz, T. Darrell, K. Goldberg, and P. Abbeel (2012), A geometric approach to robotic laundry folding, *IJRR*, *31*(2), 249–267.

[105] Mirabel, J., S. Tonneau, P. Fernbach, A.-K. Seppälä, M. Campana, N. Mansard, and F. Lamiraux (2016), Hpp: A new software for constrained motion planning, in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, pp. 383–389.

[106] Mukadam, M., X. Yan, and B. Boots (2016), Gaussian process motion planning, in *Proc. IEEE Int. Conf. Robot. Autom.*, pp. 9–15.

[107] Mukadam, M., J. Dong, X. Yan, F. Dellaert, and B. Boots (2018), Continuous-time gaussian process motion planning via probabilistic inference, *Int. J. Rob. Res.*, *37*(11), 1319–1340.

[108] Nagabandi, A., K. Konoglie, S. Levine, and V. Kumar (2019), Deep Dynamics Models for Learning Dexterous Manipulation, in *Conference on Robot Learning (CoRL)*.

[109] Nalisnick, E., A. Matsukawa, Y. W. Teh, and B. Lakshminarayanan (2019), Detecting out-of-distribution inputs to deep generative models using typicality, arxiv.1906.02994.

[110] Neal, R. M. (1996), *Bayesian Learning for Neural Networks*, Springer-Verlag.

[111] Ni, R., T. Schneider, D. Panozzo, Z. Pan, and X. Gao (2021), Robust & asymptotically locally optimal uav-trajectory generation based on spline subdivision, in *Proc. IEEE Int. Conf. Robot. Autom.*, pp. 7715–7721.

[112] Okada, M., and T. Taniguchi (2020), Variational inference mpc for bayesian model-based reinforcement learning, in *Proc. Conf. Robot Learn.*, pp. 258–272.

[113] Ortiz-Haro, J., J.-S. Ha, D. Driess, and M. Toussaint (2021), Structured deep generative models for sampling on constraint manifolds in sequential manipulation, in *Conference on Robot Learning*.

[114] Osa, T. (2020), Multimodal trajectory optimization for motion planning, *Int. J. Rob. Res.*, *39*(8), 983–1001.

[115] Paszke, A., et al. (2019), Pytorch: An imperative style, high-performance deep learning library, in *Proc. Int. Conf. Neural Information Processing Systems*.

[116] Pavlasek, J., S. R. Lewis, B. Sundaralingam, F. Ramos, and T. Hermans (2023), Ready, set, plan! planning to goal sets using generalized bayesian inference, in *Proc. Conf. Robot. Learn.*, pp. 3672–3686.

[117] Perez, E., F. Strub, H. de Vries, V. Dumoulin, and A. C. Courville (2018), Film: Visual reasoning with a general conditioning layer, in *AAAI*.

[118] Phoon, M. S., P. S. Schmitt, and G. V. Wichert (2022), Constraint-based task specification and trajectory optimization for sequential manipulation, in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, pp. 197–202.

[119] Pinneri, C., S. Sawant, S. Blaes, J. Achterhold, J. Stueckler, M. Rolinek, and G. Martius (2021), Sample-efficient cross-entropy method for real-time planning, in *Proc. Conf. Robot Learn.*, vol. 155, pp. 1049–1065.

[120] Porta, J. M., L. Jaillet, and O. Bohigas (2012), Randomized path planning on manifolds based on higher-dimensional continuation, *Int. J. Rob. Res.*, *31*(2), 201–215.

[121] Posa, M., C. Cantu, and R. Tedrake (2014), A direct method for trajectory optimization of rigid bodies through contact, *The International Journal of Robotics Research*, *33*(1), 69–81.

[122] Power, T., and D. Berenson (2021), Keep it simple: Data-efficient learning for controlling complex systems with simple models, *IEEE Robot. Autom. Lett.*, *6*(2), 1184–1191.

[123] Power, T., and D. Berenson (2022), Variational inference mpc using normalizing flows and out-of-distribution projection, in *Robot.: Sci. Syst.*

[124] Power, T., and D. Berenson (2023), Constrained stein variational trajectory optimization, arxiv.2308.12110.

[125] Power, T., and D. Berenson (2024), Learning a generalizable trajectory sampling distribution for model predictive control, *IEEE Transactions on Robotics*, *40*, 2111–2127, doi:10.1109/TRO.2024.3370026.

[126] Power, T., R. Soltani-Zarrin, S. Iba, and D. Berenson (2023), Sampling constrained trajectories using composable diffusion models, in *IROS 2023 Workshop on Differentiable Probabilistic Robotics: Emerging Perspectives on Robot Learning*.

[127] Pratt, J., C.-M. Chew, A. Torres, P. Dilworth, and G. Pratt (2001), Virtual model control: An intuitive approach for bipedal locomotion, *IJRR*, *20*(2), 129–143.

[128] Qureshi, A. H., and M. C. Yip (2018), Deeply informed neural sampling for robot motion planning, in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, pp. 6582–6588.

[129] Qureshi, A. H., J. Dong, A. Baig, and M. C. Yip (2022), Constrained motion planning networks x, *IEEE Transactions on Robotics*, *38*(2), 868–886, doi: 10.1109/TRO.2021.3096070.

[130] Rajeswaran, A., V. Kumar, A. Gupta, G. Vezzani, J. Schulman, E. Todorov, and S. Levine (2018), Learning Complex Dexterous Manipulation with Deep Reinforcement Learning and Demonstrations, in *Proceedings of Robotics: Science and Systems (RSS)*.

[131] Rawlik, K., M. Toussaint, and S. Vijayakumar (2013), On stochastic optimal control and reinforcement learning by approximate inference, in *Robot.: Sci. Syst.*

[132] Rawlik, K. C. (2013), On probabilistic inference approaches to stochastic optimal control, Ph.D. thesis, The University of Edinburgh.

[133] Rezende, D. J., and S. Mohamed (2015), Variational inference with normalizing flows, in *Proc. Int. Conf. Mach. Learn.*, p. 1530–1538.

[134] Rozzi, F., L. Roveda, and K. Haninger (2024), Combining sampling- and gradient-based planning for contact-rich manipulation.

[135] Sabatino, F. (2015), Quadrotor control: modeling, nonlinear control design, and simulation, Master's thesis, KTH Royal Institute of Technology.

[136] Sacks, J., and B. Boots (2023), Learning sampling distributions for model predictive control, in *Proc. Conf. Robot. Learn.*, pp. 1733–1742.

[137] Sacks, J., and B. Boots (2023), Learning sampling distributions for model predictive control, in *Proceedings of The 6th Conference on Robot Learning*, *Proceedings of Machine Learning Research*, vol. 205, edited by K. Liu, D. Kulic, and J. Ichnowski, pp. 1733–1742, PMLR.

[138] Schropp, J., and I. Singer (2000), A dynamical systems approach to constrained minimization, *Numer. Funct. Anal. Optim.*, *21*(3-4), 537–551.

[139] Schulman, J., et al. (2014), Motion planning with sequential convex optimization and convex collision checking, *Int. J. Rob. Res.*, *33*(9), 1251–1270.

[140] Shikhman, V., and O. Stein (2009), Constrained optimization: projected gradient flows, *J. Optim. Theory. Appl.*, *140*(1), 117–130.

[141] Sohl-Dickstein, J., E. Weiss, N. Maheswaranathan, and S. Ganguli (2015), Deep unsupervised learning using nonequilibrium thermodynamics, in *Proceedings of the 32nd International Conference on Machine Learning*, pp. 2256–2265.

[142] Song, Y., and S. Ermon (2019), Generative modeling by estimating gradients of the data distribution, in *Proceedings of the 33rd International Conference on Neural Information Processing Systems*.

[143] Sundaralingam, B., and T. Hermans (2019), Relaxed-rigidity constraints: kinematic trajectory optimization and collision avoidance for in-grasp manipulation, *Autonomous Robots*, *43*, 469–483.

[144] Theodorou, E. A., and E. Todorov (2012), Relative entropy and free energy dualities: Connections to path integral and kl control, in *IEEE 51st Conf. Decision and Control*, pp. 1466–1473.

[145] Theodorou, E. A., J. Buchli, and S. Schaal (2009), Path integral-based stochastic optimal control for rigid body dynamics, in *IEEE Symp. Adaptive Dynamic Programming and Reinforcement Learning*, pp. 219–225.

[146] Tobin, J., R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel (2017), Domain randomization for transferring deep neural networks from simulation to the real world, in *IROS*.

[147] Todorov, E. (2006), Linearly-solvable markov decision problems, in *Proc. Int. Conf. Neural Information Processing Systems*, vol. 19.

[148] Todorov, E. (2008), General duality between optimal control and estimation, in *IEEE Conf. Decision and Control*, pp. 4286–4292.

[149] Todorov, E., T. Erez, and Y. Tassa (2012), Mujoco: A physics engine for model-based control, in *IROS*.

[150] Toussaint, M. (2009), Robot trajectory optimization using approximate inference, in *Proc. Int. Conf. Mach. Learn.*, p. 1049–1056.

[151] Toussaint, M., and A. Storkey (2006), Probabilistic inference for solving discrete and continuous state markov decision processes, in *Proc. Int. Conf. Mach. Learn.*, p. 945–952.

[152] Toussaint, M., N. Ratliff, J. Bohg, L. Righetti, P. Englert, and S. Schaal (2014), Dual execution of optimized contact interaction trajectories, in *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 47–54.

[153] Trinkle, J., and J. Hunter (1991), A framework for planning dexterous manipulation, in *Proceedings. 1991 IEEE International Conference on Robotics and Automation*, pp. 1245–1251 vol.2.

[154] Urain, J., A. T. Le, A. Lambert, G. Chalvatzaki, B. Boots, and J. Peters (2022), Learning implicit priors for motion optimization, in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, pp. 7672–7679.

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

[156] Von Stryk, O., and R. Bulirsch (1992), Direct and indirect methods for trajectory optimization, *Annals of operations research*, *37*, 357–373.

[157] Wächter, A., and L. T. Biegler (2006), On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming, *Math. Program.*, *106*(1), 25–57.

[158] Wagener, N., C. Cheng, J. Sacks, and B. Boots (2019), An online learning approach to model predictive control, in *Robot.: Sci. Sys.*

[159] Wan, E. A., and R. Van Der Merwe (2000), The unscented kalman filter for nonlinear estimation, in *AS-SPCC*, pp. 153–158.

[160] Wang, A., T. Kurutach, P. Abbeel, and A. Tamar (2019), Learning robotic manipulation through visual planning and acting, in *RSS*.

[161] Wang, D., Z. Tang, C. Bajaj, and Q. Liu (2019), Stein variational gradient descent with matrix-valued kernels, *Proc. Int. Conf. Neural Information Processing Systems*, *32*.

[162] Wang, Z., O. So, J. Gibson, B. Vlahov, M. Gandhi, G.-H. Liu, and E. Theodorou (2021), Variational Inference MPC using Tsallis Divergence, in *Robot.: Sci. Syst.*

[163] Watson, J., and J. Peters (2023), Inferring smooth control: Monte carlo posterior policy iteration with gaussian processes, in *Proc. Conf. Robot. Learn.*, pp. 67–79.

[164] Watson, J., H. Abdulsamad, and J. Peters (2020), Stochastic optimal control as approximate input inference, in *Proc. Conf. Robot Learn.*, vol. 100, pp. 697–716.

[165] Webb, D. J., and J. van den Berg (2013), Kinodynamic rrt*: Asymptotically optimal motion planning for robots with linear dynamics, in *Proc. IEEE Int. Conf. Robot. Autom.*, pp. 5054–5061, doi:10.1109/ICRA.2013.6631299.

[166] Williams, G., N. Wagener, B. Goldfain, P. Drews, J. M. Rehg, B. Boots, and E. A. Theodorou (2017), Information theoretic mpc for model-based reinforcement learning, in *ICRA*.

[167] Williams, G., P. Drews, B. Goldfain, J. M. Rehg, and E. A. Theodorou (2018), Information-theoretic model predictive control: Theory and applications to autonomous driving, *IEEE Trans. Robot.*, *34*(6), 1603–1622.

[168] Winkler, C., D. Worrall, E. Hoogeboom, and M. Welling (2019), Learning likelihoods with conditional normalizing flows, arxiv.1912.00042.

[169] Xiao, Z., Q. Yan, and Y. Amit (2020), Likelihood regret: An out-of-distribution detection score for variational auto-encoder, in *Proc. Int. Conf. Neural Information Processing Systems*.

[170] Xie, A., F. Ebert, S. Levine, and C. Finn (2019), Improvisation through physical understanding: Using novel objects as tools with visual foresight, in *RSS*.

[171] Xie, Z., C. K. Liu, and K. Hauser (2017), Differential dynamic programming with nonlinear constraints, in *Proc. IEEE Int. Conf. Robot. Autom.*, pp. 695–702.

[172] Xu, J., T.-K. Koo, and Z. Li (2010), Sampling-based finger gaits planning for multifingered robotic hand, *Autonomous Robots*, *28*, 385–402.

[173] Yamashita, H. (1980), A differential equation approach to nonlinear programming, *Math. Program.*, *18*(1), 155–168.

[174] Yan, M., Y. Zhu, N. Jin, and J. Bohg (2020), Self-supervised learning of state estimation for manipulating deformable linear objects, *RA-L*, *5*(2), 2372–2379.

[175] Yu, C., and P. Wang (2022), Dexterous manipulation for multi-fingered robotic hands with reinforcement learning: A review, *Frontiers in Neurorobotics*, *16*.

[176] Yu, H., and Y. Chen (2023), A gaussian variational inference approach to motion planning, *IEEE Robot. Autom. Lett.*, *8*(5), 2518–2525.

[177] Zarrin, R. S., R. Jitosho, and K. Yamane (2023), Hybrid learning- and model-based planning and control of in-hand manipulation, in *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 8720–8726.

[178] Zhang, C., J. Huh, and D. D. Lee (2018), Learning implicit sampling distributions for motion planning, in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, pp. 3654–3661.

[179] Zhang, R., Q. Liu, and X. Tong (2022), Sampling in constrained domains with orthogonal-space variational gradient descent, *Proc. Int. Conf. Neural Information Processing Systems*, *35*, 37,108–37,120.

[180] Zucker, M., N. Ratliff, A. D. Dragan, M. Pivtoraiko, M. Klingensmith, C. M. Dellin, J. A. Bagnell, and S. S. Srinivasa (2013), Chomp: Covariant hamiltonian optimization for motion planning, *Int. J. Rob. Res.*, *32*(9-10), 1164–1193.