Learning a Generalizable Trajectory Sampling Distribution for Model Predictive Control

Thomas Power and Dmitry Berenson

***Abstract*—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, environment, and cost parameters. 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 2-D double-integrator, a 12-DoF quadrotor and a seven-DoF kinematic manipulator suggest that using a learned sampling distribution with projection outperforms MPC baselines on both in-distribution and OOD environments over different cost functions, including OOD environments generated from real-world data.**

***Index Terms*—Deep generative models, deep learning in robotics and automation, motion and path planning, nonholonomic motion planning.**

I. Introduction

MODEL predictive control (MPC) methods have been widely used in robotics for applications, such as autonomous driving [1], bipedal locomotion [2], and manipulation of deformable objects [3]. For nonlinear systems, sampling-based approaches for MPC, such as the cross entropy method (CEM) and model predictive path integral (MPPI) control [1], [4] 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.

Manuscript received 20 November 2023; revised 18 January 2024 and 29 January 2024; accepted 5 February 2024. Date of publication 27 February 2024; date of current version 13 March 2024. This paper was recommended for publication by Associate Editor E. D. Momi and Editor M. Yim upon evaluation of the reviewers' comments. This work was supported in part by the NSF under Grant IIS-1750489 and Grant IIS-2113401, and in part by the ONR under Grant N00014-21-1-2118. *(Corresponding author: Thomas Power.)*

The authors are with the Robotics Department, University of Michigan, Ann Arbor, MI 48109 USA (e-mail: tpower@umich.edu; dmitryb@umich.edu).

This article has supplementary downloadable material available at https://doi.org/10.1109/TRO.2024.3370026, provided by the authors.

Digital Object Identifier 10.1109/TRO.2024.3370026

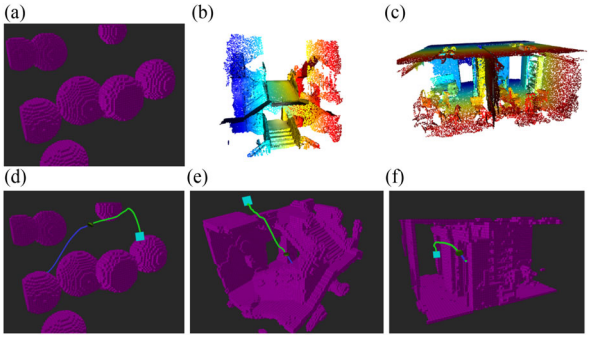

Fig. 1. (a) Training environment for our learned control sequence posterior. (b) and (c) Point clouds of two real-world environments taken from the 2-D–3-D–S dataset [5]. (d) One of our proposed methods, FlowMPPIProject, controlling a dynamic quadrotor in the training environment. (e) and (f) FlowMPPIProject 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.

Previous work has investigated the duality between control and inference [6], [7] and considered both planning and control as inference problems [8], [9], [10]. Several recent papers have considered the finite-horizon stochastic optimal control problem as Bayesian inference, and proposed methods of performing variational inference (VI) to approximate the distribution used to sample control sequences [11], [12], [13], [14]. In order to perform VI, 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 [11], [13], these distributions are initially uninformed and must be iteratively improved during deployment. In this article, 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 [15] and MPPI [1].

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 2-D 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 article, we extend our previous conference paper [16] on this topic by incorporating the learned sampling distribution with another sampling-based MPC algorithm, iCEM [15]. 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 seven-DoF 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 2-D double-integrator, a 3-D 12-DoF underactuated quadrotor, and a seven-DoF 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 (see Fig. 1). In addition we validate our methods on a seven-DoF manipulator in the real world.

II. Related Work

A. Planning and 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 [17]. This duality was later extended further by Todorov to nonlinear and deterministic systems subject to some restrictions, such as quadratic control cost and control-affine dynamics [6].

Early work by Attias framed planning for discrete state and action spaces as an inference problem over a hidden Markov model and proposed a message-passing algorithm for planning [8]. Since then, many message-passing methods have been proposed for planning in continuous action spaces [9], for solving stochastic optimal control (SOC) problems [9], [10], [18] and for policy learning [19]. For situations in which exact inference is intractable, such as for nonlinear 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 [20], leading to the path-integral approach for control [7], [21], [22], [23], [24], which solves the SOC problem by performing approximate inference with Monte-Carlo expectations over trajectories. MPPI [1] control is one of these algorithms and has enjoyed widespread use for robot control problems. Another widely used sampling-based MPC algorithm is the CEM [4], with a recent variant, improved-CEM (iCEM) [15], 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 [25], resulting in much smoother sampled control sequences. However, the resulting sampling distribution is still Gaussian, and thus cannot capture multimodal trajectories.

Recently, several approaches to control-as-inference have been developed that rely on VI to perform approximate inference [11], [12], [13], [14]. VI 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 [26]. 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 [26]. Okada and Taniguchi represent the variational posterior as a Gaussian mixture [13], and show how this posterior can be used with both MPPI and CEM, Lambert et al. proposed using a particle representation [11], this method uses Stein variational gradient descent [27], 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 nondifferentiable costs and dynamics. This method has also been extended to handle parameter uncertainty [14]. These representations allow for greater flexibility in representing complex and multimodal 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 the best of 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.

B. Learning Sampling Distributions for Planning

Our work is related to work on learning sampling distributions from data for motion planning. Previous work [28] has proposed learning a rejection sampling policy via reinforcement learning. 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 [29], [30]. These methods were restricted to geometric planning, but recent work [31] proposed an approach for kinodynamic planning, which learns a generator and discriminator that are used to sample states that are consistent with the dynamics. Recent work by Lai et al. [32] used 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. [33] 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 unimodal distributions and the sampling distribution is not dependent on the environment.

Adaptive and learned importance samplers have been used for path integral controllers [34], [35]. 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 [36]. Their approach uses a bilevel 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.

III. Problem Statement

This article focuses on the problem of finite-horizon SOC. 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 $X = \{x_0, x_1, \ldots x_T\}$ and $U = \{u_0, u_1, \ldots u_{T-1}\}$.

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 $E_{p(X|U)}[J(\tau)]$ for a given cost function J, where $p(X|U) = \prod_{t=0}^{T-1} p(x_{t+1}|x_t, u_t)$. Note that we will use J to mean both the cost on the total trajectory $J(\tau)$ and the cost of an individual state action pair $J(x, u)$. This article focuses on the problem of collision-free navigation, where J 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, nonsmooth 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.

IV. Preliminaries

A. VI for SOC

We can reformulate SOC as an inference problem (as in [10], [11], [13], [37]). First, we introduce a binary "optimality" random variable o for a trajectory such that

$$p(o = 1|\tau) \propto \exp(-J(\tau)). \tag{1}$$

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 VI to approximate it with a tractable distribution $q(\tau)$, which minimizes the KL-divergence $\mathcal{KL}(q(\tau)||p(\tau|o = 1))$ [26]. 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

$$\begin{aligned}\mathcal{KL}\left(q(\tau)||p(\tau|o=1)\right) &= \int q(\tau)\log\frac{q(\tau)}{p(\tau|o=1)}d\tau \\ &= \int q(X,U)\log\frac{p(X|U)q(U)p(o=1)}{p(o=1|X,U)p(X|U)p(U)}dX\,dU. \end{aligned}\tag{2}$$

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

$$\mathcal{F} = \int q(X,U)\log\frac{q(U)}{p(o=1|X,U)p(U)}dX\,dU \tag{3}$$

$$= -\mathbb{E}_{q(\tau)}[\log p(o|\tau) + \log p(U)] - \mathcal{H}(q(U)) \tag{4}$$

$$= \mathbb{E}_{q(\tau)}[\log \hat{J}(X,U)] - \mathcal{H}(q(U)) \tag{5}$$

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

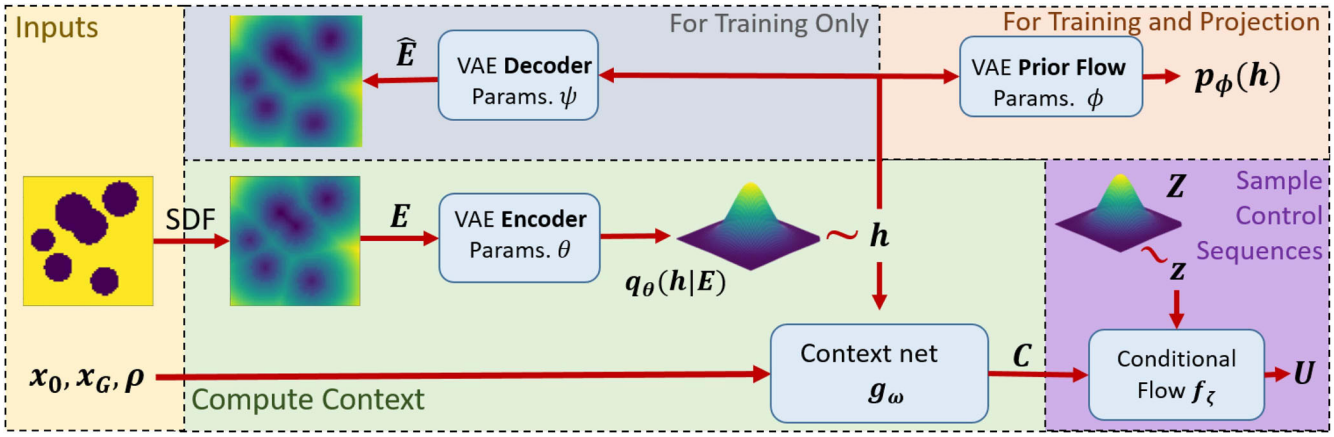

Fig. 2. Architecture of our method for sampling control sequences. We take as input initial and goal states x_0, x_G, and the environment, converted to an SDF E. E is input into a VAE to produce a latent distribution $q_\theta(h|E)$, which we sample to get the environment embedding h. This h is used, 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 CNF 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.

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* solution.

B. VI 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 [38], [39], [40]. 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:

$$p(x) = p(z) \left| \det \frac{\partial f}{\partial z} \right|^{-1} \tag{6}$$

$$\log p(y) = \log p(z) - \log \left| \det \frac{\partial f}{\partial z} \right|. \tag{7}$$

Normalizing flows can be used as a parameterization of the variational posterior [38]. 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

$$\begin{aligned} \mathcal{KL}\left(q_\theta(y)||p(y)\right) &= \int q_\theta(y) \log \frac{q_\theta(y)}{p(y)} dx \\ &= \mathbb{E}_{q_\theta(y)}[\log q_\theta(y) - \log p(y)] \\ &= \mathbb{E}_{p(z)} \left[\log p(z) - \log \left| \det \frac{\partial f_\theta}{\partial z} \right| - \log p(y) \right]. \end{aligned} \tag{8}$$

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.

V. Methods

Our proposed architecture for learning an MPC sampling distribution is shown in Fig. 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.

A. Overview of Learning the Control Sequence Posterior

The control sequence posterior introduced in Section IV-A 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) [41] 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) [42] 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) (see see Fig. 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-mentioned 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 $q_\zeta(U|C)$. In the following, we describe each component of the method to learn $q_\zeta(U|C)$ in detail.

B. 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 VI-A. 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 $q_\theta(h|E)$, 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. Chen et al. [43] 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

$$\begin{aligned}\mathcal{L}_{\text{VAE}} &= \mathbb{E}_{q_\theta(h|E)}\left[-\log p_\psi(E|h)\right] + \mathcal{KL}(q_\theta(h|E)||p_\phi(h)) \\ &= \mathbb{E}_{q_\theta(h|E)}\left[-\log p_\psi(E|h) + \log q_\theta(h|E) - \log p_\phi(h))\right].\end{aligned} \tag{9}$$

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.

C. Learning $q_\zeta(U|C)$

We use a CNF 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 (4) then becomes

$$\begin{aligned}\mathcal{F} = &-\mathbb{E}_{q(\tau)}\left[\log p(o|\tau)\right] \\ &+ \mathbb{E}_{p(Z)}\left[\log p(Z) - \log\left|\det \frac{\partial f_\zeta(Z, C)}{\partial Z}\right|\right].\end{aligned} \tag{10}$$

We can then optimize ζ to minimize the free energy.

By using a CNF, we are amortizing the cost of computing the posterior across environments. The CNF $U = f_\zeta(Z, C)$ is invertible with respect to Z, i.e., $Z = f^{-1}(U, C)$. For our CNF we use an architecture based on real-NVP [39] architecture with conditional coupling layers [41], the structure is specified in Section VI-C. 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 (10) via gradient descent requires the cost and dynamics to be differentiable. To avoid this, we estimate gradients, using the method in [13]: At each iteration, we sample R control sequences $U_{1..R}$ from $q_\zeta(U|C)$ and compute weights

$$w_i = \frac{q_\zeta(U_i|C)^{-\beta} p(o|\tau_i)^{\frac{1}{\alpha}}}{\frac{1}{R}\sum_{j=1}^{R} q_\zeta(U_j|C)^{-\beta} p(o|\tau_j)^{\frac{1}{\alpha}}} \tag{11}$$

where $p(o|\tau) = \exp(-J(\tau))$. 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

$$\mathcal{L}_{\text{flow}} = -\sum_{i=1}^{R} w_i \log q_\zeta(U_i|C). \tag{12}$$

This process is equivalent to performing mirror descent on the variational free energy, see Okada and Taniguchi's work [13] 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:

$$\min_{\theta,\phi,\psi,\omega,\zeta} \mathcal{L}_{\text{flow}} + a\mathcal{L}_{\text{VAE}} \tag{13}$$

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

$$\min_{\omega,\zeta} \mathcal{L}_{\text{flow}}. \tag{14}$$

D. 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 MPPI [1] and iCEM [15], 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.

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

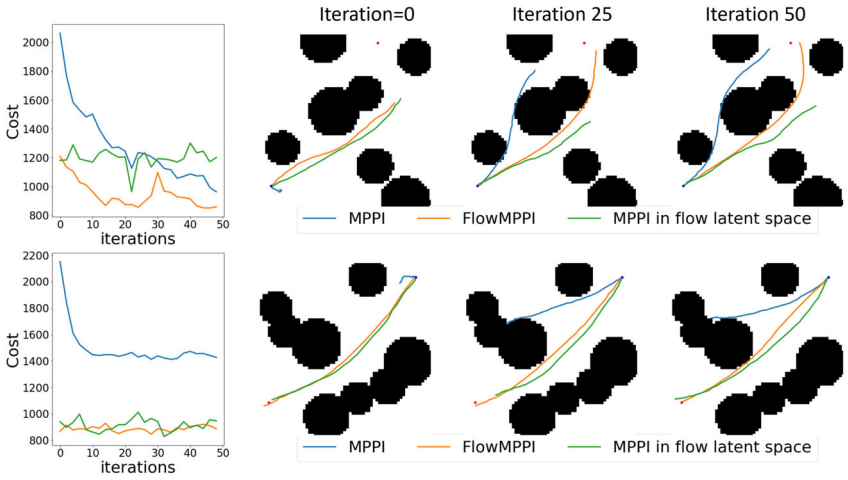

Fig. 3. Two examples in which we run 50 iterations of MPPI, FlowMPPI, and MPPI in the latent space of the flow in a 2-D navigation task with double-integrator dynamics. Top: Initial samples from the flow are goal directed, but do not yet fully reach the goal, in contrast the initial samples for MPPI 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 and mirror descent was noted in [44]. 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 (see Fig. 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 (4), all low-cost control sequences should have high density under $q_\zeta(U|C)$, there will inevitably be an approximation gap, i.e., $\mathcal{KL}(q_\zeta(U|C)||p(\tau|o=1)) > 0$. 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 1. 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 $S^Z_{\text{nominal}} = \lambda \epsilon_Z^T (f_\psi^{-1}(U, C) - \epsilon_Z)$. 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 $S^U_{\text{nominal}} = \lambda ||U_k - U)||^2_{\Sigma^{-1}}$. 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.

2) FlowiCEM: CEM [4] 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 [15] 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

Algorithm 1: 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

1: **function**FlowMPPIStep
2: ▷Perform shift operation on nominal U
3: **for** t $\in \{1, \ldots, T-1\}$ **do**
4: $U_{t-1} \leftarrow U_t$
5: $U_{T-1} \sim \mathcal{N}(0, \Sigma)$
6: ▷Generate samples by perturbing nominal U
7: **for**k $\in \{1, \ldots, \frac{K}{2}\}$**do**
8: $\epsilon_U \sim \mathcal{N}(0, \Sigma)$
9: $U_k \leftarrow U + \epsilon_U$
10: $\tau_k \sim p(\tau|U_k)$ ▷ Sample trajectory
11: $S_k \leftarrow J(\tau_k) + \lambda U_k \Sigma^{-1} \epsilon_U$ ▷ Compute cost
12: ▷Generate samples from control sequence posterior
13: **for**k $\in \{\frac{K}{2} + 1, \ldots, K\}$ **do**
14: $\epsilon_Z \sim \mathcal{N}(0, I)$
15: $U_k \leftarrow f_\zeta(\epsilon_Z, C)$
16: $\tau_k \sim p(\tau|U_k)$ ▷ Sample trajectory
17: $S_k \leftarrow J(\tau_k) + S_{nominal}$ ▷ Compute cost
18: ▷Compute new nominal U
19: $\beta \leftarrow \min_k S_k$
20: $\eta = \sum_{k=1}^{K} \exp(-\frac{1}{\lambda}(S_k - \beta))$
21: **for**k $\in \{1, \ldots, K\}$**do**
22: $w_k \leftarrow \frac{1}{\eta} \exp(-\frac{1}{\lambda}(S_k - \beta))$
23: $U \leftarrow \sum_{k=1}^{K} w_k U_k$
24: **return** U

Algorithm 2: 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

1: **function**FlowiCEMStep
2: ▷Perform shift operation on nominal mean
3: $\mu \leftarrow$ shifted μ
4: **if** $U_{kept-elites} \neq \{\}$ **then**
5: $U_{kept-elites} \leftarrow$ shifted $U_{kept-elites}$
6: $\mu_{T-1} \sim \mathcal{N}(0, \Sigma)$
7: ▷Generate initial elites from flow
8: $\epsilon_Z \sim \mathcal{N}(0, I)$
9: $U_{flow} \leftarrow f_\zeta(\epsilon_Z, C)$
10: **for**i $\in \{1, \ldots,$ iters$\}$**do**
11: ▷Generate samples from colored noise
12: $U \leftarrow$ SAMPLECOLOREDNOISE (μ, σ^2)
13: ▷Add samples from flow to population
14: **if**i $==$ 0**then**
15: $U \leftarrow U \cup U_{flow}$
16: ▷Add kept-elites to population
17: $U \leftarrow U \cup U_{kept-elites}$
18: $\tau \sim p(\tau|U)$ ▷ Sample trajectories
19: costs $\leftarrow J(\tau)$ ▷ Evaluate costs
20: $U_{elites} \leftarrow N_{elites}$ lowest-cost trajectories
21: $\mu \leftarrow (1-m)$mean$(U_{elites}) + m\mu$
22: $\sigma \leftarrow (1-m)$std$(U_{elites}) + m\sigma$
23: $U_{kept-elites} \leftarrow BestN_{kept-elites}$ elites
24: **return** lowest-cost elite from $U_{kept-elites}$

reasons discussed in Section V-D1). 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 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 2. 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 $N_{\text{kept-elites}} < N_{\text{elites}}$ 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.

E. 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 OOD 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 low-cost trajectories. For the OOD score, we use the VAE we have discussed in

Section V-B. VAEs and other deep latent variable models have been used to detect OOD data in prior work [45], [46], [47]; 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 3-D 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

$$\mathcal{L}_{\text{OOD}}(h) = -\log p_\phi(h) \tag{15}$$

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 $\mathcal{L}_{\text{VAE}}$, so we should expect it to be lower for in-distribution data. Previous work on OOD detection using normalizing flows [48] 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 overreliance 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 [43]. Likewise, we found the learned prior yielded much better OOD detection than using a Gaussian prior, which is the standard VAE prior (see Fig. 4).

We can perform gradient descent on $\mathcal{L}_{\text{OOD}}$ 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 $\mathcal{L}_{\text{flow}}$ 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

$$\hat{h} = \arg\min_h \; b\mathcal{L}_{\text{OOD}} + \mathcal{L}_{\text{flow}} \tag{16}$$

for scalar $b > 0$. We project h to $\hat{h}$ by minimizing the (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

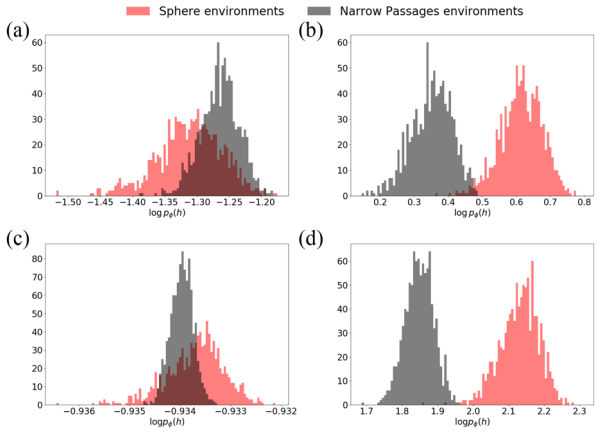

Fig. 4. Comparison of our OOD scores with using a VAE with a standard Gaussian prior for in-distribution (red) and OOD (gray) simulated environments. (a) Planar navigation using a Gaussian prior. (b) Planar navigation using a Normalizing flow prior. (c) 12-DoF quadrotor using a Gaussian prior. (d) 12-DoF quadrotor using a Normalizing flow prior, These scores are computed by sampling h from $q_\theta(h|E)$ 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 OOD environments in both cases.

Algorithm 3: Projection.

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

$h^1 \leftarrow q_\theta(h|E)$
for $n \in \{1, \ldots, N\}$ **do**
  Compute $\log p_\phi(h^n)$ via (7)
  $C \leftarrow g_\omega(x_0, x_G, h^n)$
  $\{U_k, q_\zeta(U_k|C)\}_{k=1}^K \leftarrow \text{SAMPLEPERTU}(C, \Sigma_\epsilon, K)$
  $\mathcal{L} \leftarrow -p_\phi(h^n)$
  for $k \in \{1, \ldots, K\}$ **do**
    $w_k \leftarrow$ from $(\{U_i, \log q_\zeta(U_i|C)\}_{i=1}^K, \alpha, \beta\})$ via (11)
    $\mathcal{L} \leftarrow \mathcal{L} - w_k \cdot \log q_\zeta(U_k|C)$
  $h^{n+1} \leftarrow h^n - \eta \frac{\partial \mathcal{L}}{\partial h}$

M steps of gradient descent on the above-mentioned combined loss at initialization, followed by a single step at each iteration of the MPC. The algorithm for projection is shown in Algorithm 3. The algorithm SAMPLEPERTU is shown in the appendix.

VI. EVALUATION

In this section, we will evaluate our proposed approaches FlowMPPI and FlowiCEM with and without projection on three simulated systems; a 2-D point robot, a 3-D 12-DoF quadrotor, and a seven-DoF 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 12-DoF quadrotor system and the seven-DoF

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 "OOD" 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 OOD has no bearing on the nonlearning-based baselines. The performance of nonlearning 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 12-DoF quadrotor and two-DoF double integrator, smoothness of the resulting controls. To compute smoothness we use

$$\mathcal{L}_{\text{smooth}}(U) = \sum_{t=1} ||u_t - u_{t-1}||^2. \tag{17}$$

A. Systems and 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 seven-DoF 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 found in Appendix B.

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 Fig. 5(a) and (b). The training environments consist disc-shaped obstacles, where the size, location, and number of obstacles is randomized. The OOD 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 B2.

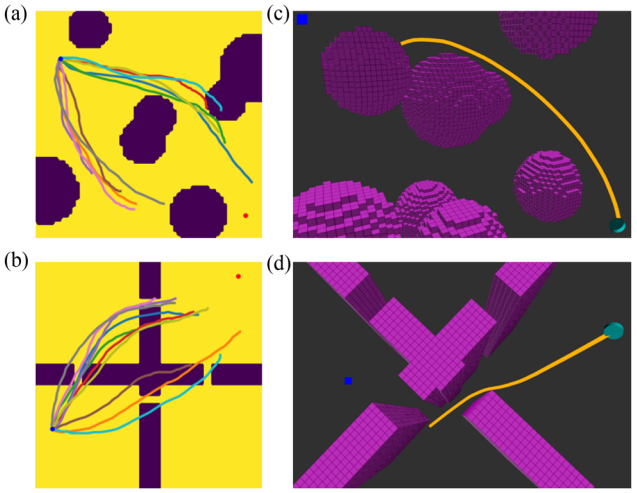

Fig. 5. Examples of our "in-distribution" environments (top) and "OOD" environments (bottom). (a) Sphere environment for the planar navigation task, showing sampled trajectories from the flow. (b) Narrow passages environment for planar navigation, we see that the samples from the flow are goal orientated and generally toward the passages, but most are generally not collision free. (c) Sphere environment for the 12-DoF quadrotor. (d) Narrow passages environment for the 12-DoF quadrotor.

2) 3-D 12-DoF Quadrotor: This system is a 3-D 12-DoF 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 Fig. 5(c) and (d). The training environment consists of spherical obstacles of random size, location, and number, and the OOD 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 3-D 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

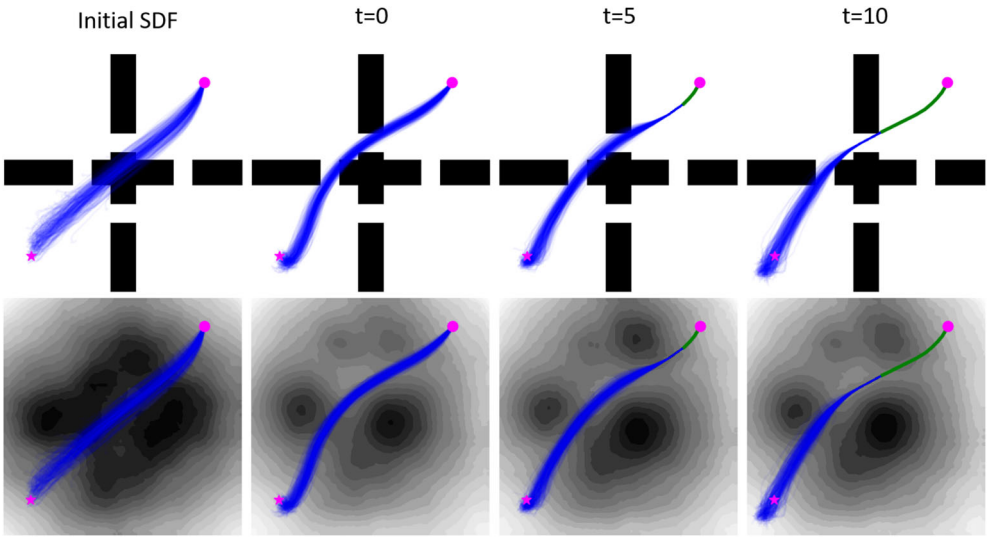

Fig. 6. Projection process visualized for the planar navigation task. We visualize the projected environment embedding using the VAE decoder on the bottom row. Note that decoding h is only used for training the VAE and visualization, it is not necessary for projection. The top shows the environment and 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.

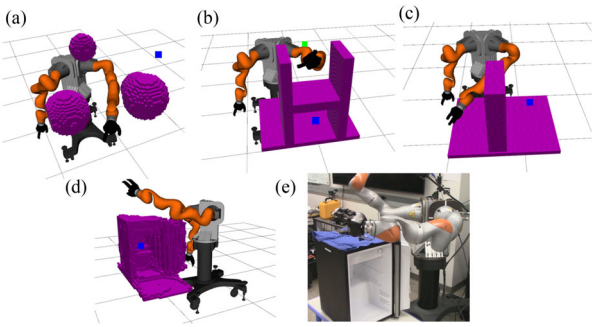

Fig. 7. We evaluate our approach on control of a kinematic seven-DoF manipulator on four environments in simulation (a)–(d). Tasks consist of: (a) navigating around spherical obstacles; (b) reaching into a shelf; (c) going from one side of a wall to another; (d) reaching inside a fridge; (e) real world 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.

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 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 B3.

3) Seven-DoF Manipulator: This system is a kinematic seven-DoF manipulator shown in different environments in Fig. 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 Fig. 7(a). The number and size of the spherical obstacles is randomized during training. The simulated novel environments are shown in Fig. 7(b)–(d). We additionally evaluate on one environment generated from real-world data, shown in Fig. 7(d) and (e). The environment is represented as a $64 \times 64 \times 64$ SDF. The goal region is specified as a 3-D 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 seven-DoF manipulator task is quasistatic, 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.

B. 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 previously. Fig. 4 shows that this score is clearly able to distinguish in-distribution environment embeddings from OOD ones.

C. 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 [39],

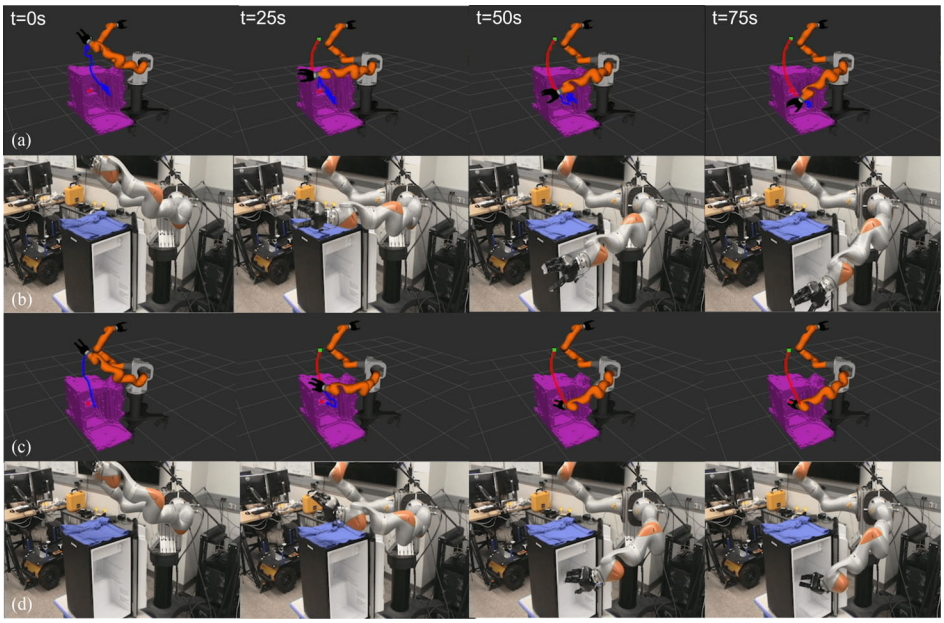

Fig. 8. (a) and (b) iCEM baseline performing a task where the goal is to navigate to to the inside of the fridge. The baseline fails to successfully navigate to the goal. (c) and (d) One of our proposed methods, FlowiCEMProject, successfully navigating to the inside of the fridge.

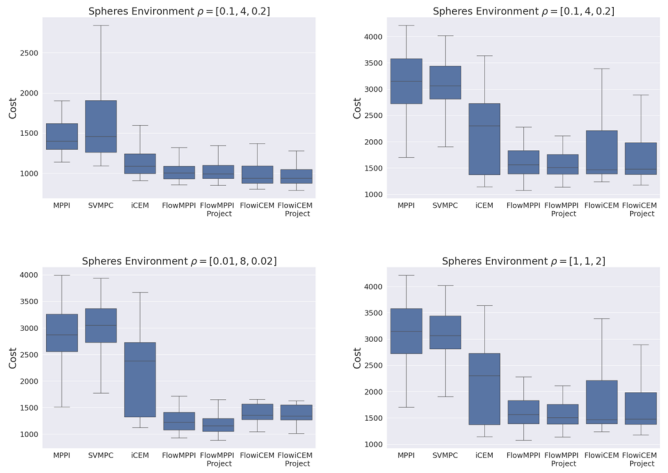

Fig. 9. Box plot of the costs for the double integrator experiments. We evaluate on 100 trials for the training environment consisting of randomly generated disc obstacles. In addition we evaluate for 100 trials with three different cost parameterizations in three different environments consisting of four walls with narrow passages between them.

shown in Fig. 11. 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 12-DoF quadrotor and two-DoF double integrator, and 20 for the seven-DoF manipulator. For the control sampling flow we use the conditional coupling layers from [41]. 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 3-D case we use 3-D 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 [49].

D. Training and Data

For training, we use 10 000 randomly generated environments for planar navigation task, and 20 000 for the 3-D 12-DoF quadrotor and seven-DoF 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 $\mathcal{L}_{\text{VAE}}$. 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 seven-DoF manipulator planning, we train with the noise. We found that without using the noise for the seven-DoF 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.

E. Baselines

For our baselines, we use several state-of-the-art sampling-based MPC methods: MPPI [1], Stein variational MPC (SVMPC) [11] and iCEM [15]. 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 OOD environments. We evaluate each

TABLE I
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 $\rho = [0.1, 25, 0.2]^T$			OOD environment $\rho = [0.1, 25, 0.2]^T$			$\rho = [0.1, 25, 2]^T$			$\rho = [1, 12, 0.2]^T$		
Controller	Success	Cost	$\mathcal{L}_{\text{smooth}}$	Success	Cost	$\mathcal{L}_{\text{smooth}}$	Success	Cost	$\mathcal{L}_{\text{smooth}}$	Success	Cost	$\mathcal{L}_{\text{smooth}}$
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

The bold numbers indicate the highest performance across the different controllers.

TABLE II
COMPARISON OF METHODS FOR THE 12-DOF QUADROTOR TASK. WE EVALUATE ON BOTH IN-DISTRIBUTION ENVIRONMENTS AND OOD ENVIRONMENTS ACROSS DIFFERENT COST FUNCTION PARAMETERS ρ

	In-distribution environment $\rho = [0.1, 25, 0.2]^T$			OOD environment $\rho = [0.1, 25, 0.2]^T$			$\rho = [0.1, 25, 2]^T$			$\rho = [1, 12, 0.2]^T$		
Controller	Success	Cost	$\mathcal{L}_{\text{smooth}}$	Success	Cost	$\mathcal{L}_{\text{smooth}}$	Success	Cost	$\mathcal{L}_{\text{smooth}}$	Success	Cost	$\mathcal{L}_{\text{smooth}}$
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

The bold numbers indicate the highest performance across the different controllers.

TABLE III
COMPARISON OF METHODS FOR THE 3-D 12-DOF QUADROTOR NAVIGATION TASK WITH TWO ENVIRONMENTS GENERATED FROM REAL-WORLD DATA. THE ROOMS ENVIRONMENT IS SHOWN IN FIG. 5(B) AND THE STAIRWAY ENVIRONMENT IS SHOWN IN FIG. 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	$\mathcal{L}_{\text{smooth}}$	Cost	Success	$\mathcal{L}_{\text{smooth}}$
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

The bold numbers indicate the highest performance across the different controllers.

TABLE IV
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
12-DoF quadrotor	0.084	0.124	0.087	0.076	0.136	0.134	0.195

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 4.

F. Results

The results comparing our MPC methods to baselines are shown in Tables I, II III, and V. In addition, box plots showing the distribution of costs are shown in Figs. 9 and 10, and the computational times for all methods are shown in Table IV. 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 OOD 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 versus 1.20 for iCEM while improving the cost from 1427 to 1008. The flow-based methods

TABLE V
RESULTS FOR ATTEMPTING TASK 100 TIMES FOR EACH ENVIRONMENT IN SIMULATION. THE ENVIRONMENTS ARE SHOWN IN FIG. 7. THE FRIDGE ENVIRONMENT IS GENERATED FROM REAL-WORLD DATA FROM THE FRIDGE SHOWN IN FIG. 7(E)

	In-distribution		OOD					
	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**

The bold numbers indicate the highest performance across the different controllers.

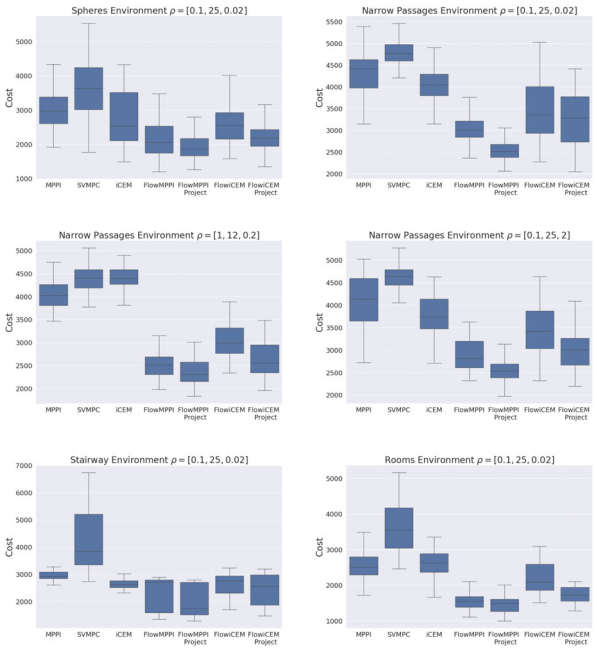

Fig. 10. Box plot of the costs for the 12-DoF quadrotor experiments. We evaluate on 50 trials for the training environment consisting of randomly generated disc obstacles. In addition, we evaluate 50 trials with three different cost parameterizations in three different environments consisting of four walls with narrow passages between them.

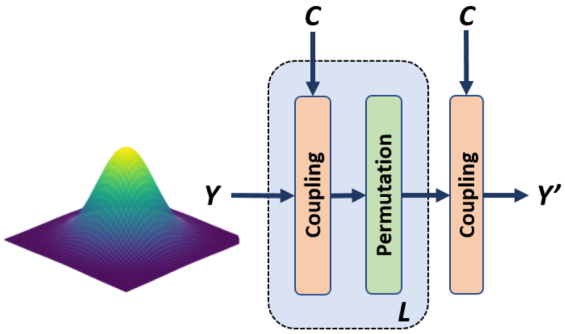

Fig. 11. Architecture for both the prior flow and the control sequence posterior flow, based on [39] and [41], showing a mapping from arbitrary Y to Y'. Each flow consists of L chained transformation blocks. A transformation block consists of a coupling layer and a random permutation. There is a final 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.

show stronger control action, trading off smoothness for rapidly moving to the goal. Since the overall cost for these methods is lower, this suggests that this tradeoff is desirable according to our given cost functions. The projection process for the planar navigation is shown in Fig. 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 suboptimal 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% versus 24% for iCEM and 8% for MPPI. Increasing the smoothness parameter in the cost does lead to a corresponding improvement in the $\mathcal{L}_{\text{smooth}}$ 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 nontrivial. 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 III shows the results when evaluating our method in simulation in two environments generated from real-world data. FlowMPPIProject outperforms all other methods in cost and 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 seven-DoF manipulator experiment are shown in Table V. 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,

TABLE VI
TRAINING AND ARCHITECTURE HYPERPARAMETERS

Variable	Planar navigation	3-D 12-DoF quadrotor	Seven-DoF 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

FlowiCEMProject achieved 97% success rate compared with 89% for iCEM and 44% for SVMPC. Fig. 8 shows FlowiCEMProject running on a seven-DoF 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.

VII. 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 was 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 environment dynamics during planning, by predicting the future environment SDFs as in [50]. 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 IV that incorporating projection requires significant computation time, and thus the current implementation cannot be used for real-time control. For both the 12-DoF quadrotor and the two-DoF 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 12-DoF 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 [49].

Fifth, training an effective control-sequence posterior requires tuning system-dependent hyper-parameters. While some hyper-parameters can be automatically selected (see Appendix A1 for details), tuning parameters α and β is necessary when considering a new system. These parameters control the tradeoff between diversity and low-cost control-sequence samples, and this tradeoff 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.

VIII. CONCLUSION

In this article, we have presented a framework for using a CNF to learn a control sequence sampling distribution for MPC based on the formulation of MPC as VI. 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.

ACKNOWLEDGMENT

The authors would like to thank the other members of the Autonomous Robotic Manipulation Lab at the University of Michigan for their insightful discussions and feedback.

APPENDIX A
TRAINING AND ARCHITECTURE DETAILS

1) Hyperparameter Tuning

There are several hyperparameters to tune in our approach. The scalar a in (13) was tuned so that $a\mathcal{L}_{\text{VAE}}$ and $\mathcal{L}_{\text{flow}}$ were of approximately similar magnitude. The scalar b in (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 tradeoff 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.

APPENDIX B
ENVIRONMENT DETAILS

The environments are $4\,\text{m} \times 4\,\text{m}$ for the planar navigation task, $4\,\text{m} \times 4\,\text{m} \times 4\,\text{m}$ for the 12-DoF quadrotor, and $1.5\,\text{m} \times 1.5\,\text{m} \times 1.5\,\text{m}$ for the seven-DoF 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 and 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 OOD 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.

TABLE VII
CONTROLLER AGNOSTIC PARAMETERS USED FOR THE EVALUATIONS

Variable	Planar navigation	12DoF quadrotor	Seven-DoF manipulator
Control horizon H	40	40	40
Trial length T	100	100	100
Dynamics Δt	0.05	0.025	0.025

To ensure the navigation problem is nontrivial, we sample starts and goals that are at least 4 m away.

1) Real-World Environments

The two real-world environments are taken from area 3 from the 2-D–3-D–S dataset [5]. To generate the two environments, we used the 3-D 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.

2) Planar Navigation

The dynamics for the planar navigation system are

$$\begin{bmatrix} x \\ y \\ \dot{x} \\ \dot{y} \end{bmatrix}_{t+1} = \begin{bmatrix} 1 & 0 & \Delta t & 0 \\ 0 & 1 & 0 & \Delta t \\ 0 & 0 & 0.95 & 0 \\ 0 & 0 & 0 & 0.95 \end{bmatrix} \begin{bmatrix} x \\ y \\ \dot{x} \\ \dot{y} \end{bmatrix}_{t} + \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ \Delta t & 0 \\ 0 & \Delta t \end{bmatrix} \mathbf{u}. \tag{18}$$

3) 12-DoF Quadrotor

The dynamics for the 12-DoF quadrotor are from [51] and are given by

$$\begin{bmatrix} x \\ y \\ z \\ p \\ q \\ r \\ \dot{x} \\ \dot{y} \\ \dot{z} \\ \dot{p} \\ \dot{q} \\ \dot{r} \end{bmatrix}_{t+1} = \begin{bmatrix} x \\ y \\ z \\ p \\ q \\ r \\ \dot{x} \\ \dot{y} \\ \dot{z} \\ \dot{p} \\ \dot{q} \\ \dot{r} \end{bmatrix}_{t} + \Delta t \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \\ \dot{p} + \dot{q}s(p)t(q) + \dot{r}c(p)t(q) \\ \dot{q}c(p) - \dot{r}s\dot{p} \\ \dot{q}\frac{s(p)}{c(q)} + \dot{r}\frac{c(p))}{c(q)} \\ -(s(p)s(r) + c(r)c(p)s(q))K\frac{u_1}{m} \\ -(c(r)s(p) - c(p)s(r)s(q))K\frac{u_1}{m} \\ g - c(p)s(q))K\frac{u_1}{m} \\ \frac{(I_y - I_z)\dot{q}\dot{r} + Ku_2}{I_x} \\ \frac{(I_z - I_x)\dot{p}\dot{r} + Ku_3}{I_y} \\ \frac{(I_x - I_y)\dot{p}\dot{q} + Ku_4}{I_z} \end{bmatrix}_{t} \tag{19}$$

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 0.1 m and height 0.05 m.

TABLE VIII
CONTROLLER HYPERPARAMETERS USED FOR THE EXPERIMENTS FOR BOTH OUR PROPOSED METHOD AND THE BASELINES

Controller	Variable	Planar navigation	12-DoF quadrotor	Seven-DoF 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}	2×10^{-3}	1×10^{-2}

4) *Seven-DoF Manipulator*

We use a kinematic model of the seven-DoF manipulator

$$q = q + u\Delta t \tag{20}$$

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

APPENDIX C ALGORITHMS

Algorithm 4: Sample from $q(U|C)$.

function SampleUC, K
  for i $\in \{k, \ldots, K\}$ **do**
    $Z_k \sim \mathcal{N}(0, I)$
    $U_k \leftarrow f_\zeta(Z_k, C)$
    $q_\zeta(U_k|C) \leftarrow$ from$\hat{Z}_k via (7)$
  return $\{U_k, q_\zeta(U_k|C)\}_{k=1}^K$

Algorithm 5: Sample from $q(U|C)$ with Perturbation.

function SamplePertUC, Σ_ϵ, K
  for i $\in \{k, \ldots, K\}$ **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$\hat{Z}_k via (7)$
  return $\{U_k, q_\zeta(U_k|C)\}_{k=1}^K$

Algorithm 6: 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 $n \in \{1, \ldots, N\}$ **do**
  $h \leftarrow q_\theta(h|E)$
  $\hat{E} \leftarrow p_\psi(E|h)$
  Compute $\log p_\phi(h)$ via (7)
  Compute $\mathcal{L}_{VAE}$
  $C \leftarrow g_\omega(x_0, x_G, \rho, h)$
  $\{U_k, q_\zeta(U_k|C)\}_{k=1}^K \leftarrow$ SAMPLEPERTU(C, Σ_ϵ, K)
  $\mathcal{L} \leftarrow \mathcal{L}_{VAE}$
  for k $\in \{1, \ldots, K\}$ **do**
    $w_k \leftarrow$ from $(\{U_i, \log q_\zeta(U_i|C)\}_{i=1}^K, \alpha, \beta\})$ via (11)
    $\mathcal{L} \leftarrow \mathcal{L} - w_k \cdot \log q_\zeta(U_k|C)$
  $\Theta^{n+1} \leftarrow \Theta^n - \eta \frac{\partial \mathcal{L}}{\partial \Theta}$

REFERENCES

[1] G. Williams, P. Drews, B. Goldfain, J. M. Rehg, and E. A. Theodorou, "Information-theoretic model predictive control: Theory and applications to autonomous driving," *IEEE Trans. Robot.*, vol. 34, no. 6, pp. 1603–1622, Dec. 2018.

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

[3] T. Power and D. Berenson, "Keep it simple: Data-efficient learning for controlling complex systems with simple models," *IEEE Robot. Automat. Lett.*, vol. 6, no. 2, pp. 1184–1191, Apr. 2021.

[4] M. Kobilarov, "Cross-entropy motion planning," *Int. J. Robot. Res.*, vol. 31, no. 7, pp. 855–871, 2012.

[5] I. Armeni, S. Sax, A. R. Zamir, and S. Savarese, "Joint 2D-3D-semantic data for indoor scene understanding," 2017, *arXiv:1702.01105*.

[6] E. Todorov, "General duality between optimal control and estimation," in *Proc. IEEE Conf. Decis. Control*, 2008, pp. 4286–4292.

[7] E. A. Theodorou and E. Todorov, "Relative entropy and free energy dualities: Connections to path integral and KL control," in *Proc. IEEE 51st Conf. Decis. Control*, 2012, pp. 1466–1473.
[8] H. Attias, "Planning by probabilistic inference," in *Proc. 9th Int. Workshop Artif. Intell. Statist.*, 2003, pp. 9–16.
[9] M. Toussaint and A. Storkey, "Probabilistic inference for solving discrete and continuous state Markov decision processes," in *Proc. Int. Conf. Mach. Learn.*, 2006, pp. 945–952.
[10] K. Rawlik, M. Toussaint, and S. Vijayakumar, "On stochastic optimal control and reinforcement learning by approximate inference," in *Proc. Conf. Robot.: Sci. Syst.*, 2013.
[11] A. Lambert, A. Fishman, D. Fox, B. Boots, and F. Ramos, "Stein variational model predictive control," in *Proc. Conf. Robot Learn.*, 2020, pp. 1278–1297.
[12] Z. Wang et al., "Variational inference MPC using Tsallis divergence," in *Proc. Conf. Robot.: Sci. Syst.*, 2021.
[13] M. Okada and T. Taniguchi, "Variational inference MPC for Bayesian model-based reinforcement learning," in *Proc. Conf. Robot Learn.*, 2020, pp. 258–272.
[14] L. Barcelos, A. Lambert, R. Oliveira, P. Borges, B. Boots, and F. Ramos, "Dual online Stein variational inference for control and dynamics," in *Proc. Conf. Robot.: Sci. Syst.*, 2021.
[15] C. Pinneri et al., "Sample-efficient cross-entropy method for real-time planning," in *Proc. Conf. Robot Learn.*, 2021, vol. 155, pp. 1049–1065.
[16] T. Power and D. Berenson, "Variational inference MPC using normalizing flows and out-of-distribution projection," in *Proc. Conf. Robot.: Sci. Syst.*, 2022.
[17] R. E. Kalman, "A new approach to linear filtering and prediction problems," *J. Basic Eng.*, vol. 82, no. 1, pp. 35–45, 1960.
[18] J. Watson, H. Abdulsamad, and J. Peters, "Stochastic optimal control as approximate input inference," in *Proc. Conf. Robot Learn.*, 2020, vol. 100, pp. 697–716.
[19] K. C. Rawlik, "On probabilistic inference approaches to stochastic optimal control," Ph.D. dissertation, The Univ. of Edinburgh, Edinburgh, U.K., 2013.
[20] W. H. Fleming and S. K. Mitter, "Optimal control and nonlinear filtering for nondegenerate diffusion processes," *Stochastics*, vol. 8, no. 1, pp. 63–77, 1982.
[21] H. J. Kappen, "Linear theory for control of nonlinear stochastic systems," *Phys. Rev. Lett.*, vol. 95, Nov. 2005, Art. no. 200201.
[22] E. Todorov, "Linearly-solvable Markov decision problems," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2006, vol. 19, pp. 1369–1376.
[23] H. J. Kappen, V. Gómez, and M. Opper, "Optimal control as a graphical model inference problem," *Mach. Learn.*, vol. 87, no. 2, pp. 159–182, May 2012.
[24] E. A. Theodorou, J. Buchli, and S. Schaal, "Path integral-based stochastic optimal control for rigid body dynamics," in *Proc. IEEE Symp. Adaptive Dyn. Program. Reinforcement Learn.*, 2009, pp. 219–225.
[25] J. Watson and J. Peters, "Inferring smooth control: Monte Carlo posterior policy iteration with gaussian processes," in *Proc. 6th Conf. Robot Learn.*, 2023, vol. 205, pp. 67–79.
[26] D. M. Blei, A. Kucukelbir, and J. D. McAuliffe, "Variational inference: A review for statisticians," *J. Amer. Statist. Assoc.*, vol. 112, no. 518, pp. 859–877, 2017.
[27] Q. Liu and D. Wang, "Stein variational gradient descent: A general purpose Bayesian inference algorithm," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2016, pp. 2378–2386.
[28] C. Zhang, J. Huh, and D. D. Lee, "Learning implicit sampling distributions for motion planning," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2018, pp. 3654–3661.
[29] B. Ichter, J. Harrison, and M. Pavone, "Learning sampling distributions for robot motion planning," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2018, pp. 7087–7094.
[30] A. H. Qureshi and M. C. Yip, "Deeply informed neural sampling for robot motion planning," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2018, pp. 6582–6588.
[31] L. Li, Y. Miao, A. H. Qureshi, and M. C. Yip, "MPC-MPNet: Model-predictive motion planning networks for fast, near-optimal planning under kinodynamic constraints," *IEEE Robot. Autom. Lett.*, vol. 6, no. 3, pp. 4496–4503, 2021, doi: 10.1109/LRA.2021.3067847.
[32] T. Lai, W. Zhi, T. Hermans, and F. Ramos, "Parallelised diffeomorphic sampling-based motion planning," in *Proc. Conf. Robot. Learn.*, 2021, pp. 81–90.
[33] T. Loew, T. Bandyopadhyay, J. Williams, and P. Borges, "PROMPT: Probabilistic motion primitives based trajectory planning," in *Proc. Conf. Robot.: Sci. Syst.*, 2021.
[34] H. Kappen and H.-C. Euler, "Adaptive importance sampling for control and inference," *J. Statist. Phys.*, vol. 162, pp. 1244–1266, 2016.
[35] J. Carius, R. Ranftl, F. Farshidian, and M. Hutter, "Constrained stochastic optimal control with learned importance sampling: A path integral approach," *Int. J. Robot. Res.*, vol. 41, no. 2, pp. 189–209, 2022.
[36] J. Sacks and B. Boots, "Learning sampling distributions for model predictive control," in *Proc. 6th Conf. Robot. Learn.*, vol. 205, 2023, pp. 1733–1742.
[37] M. Toussaint, "Robot trajectory optimization using approximate inference," in *Proc. Int. Conf. Mach. Learn.*, 2009, pp. 1049–1056.
[38] D. J. Rezende and S. Mohamed, "Variational inference with normalizing flows," in *Proc. Int. Conf. Mach. Learn.*, 2015, pp. 1530–1538.
[39] L. Dinh, J. Sohl-Dickstein, and S. Bengio, "Density estimation using real NVP," in *Proc. Int. Conf. Learn. Representations*, 2017.
[40] D. P. Kingma and P. Dhariwal, "Glow: Generative flow with invertible 1x1 convolutions," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2018, pp. 10215–10224.
[41] C. Winkler, D. Worrall, E. Hoogeboom, and M. Welling, "Learning likelihoods with conditional normalizing flows," 2019, *arXiv:1912.00042*.
[42] D. P. Kingma and M. Welling, "Auto-encoding variational Bayes," in *Proc. Int. Conf. Learn. Representations*, 2014.
[43] X. Chen et al., "Variational lossy autoencoder," in *Proc. Int. Conf. Learn. Representations*, 2017.
[44] N. Wagener, C. Cheng, J. Sacks, and B. Boots, "An online learning approach to model predictive control," in *Proc. Conf. Robot.: Sci. Syst.*, 2019.
[45] Y. Feng, D. J. X. Ng, and A. Easwaran, "Improving variational autoencoder based out-of-distribution detection for embedded real-time applications," *ACM Trans. Embedded Comput. Syst.*, vol. 20, no. 5s, Sep. 2021, Art. no. 95.
[46] Z. Xiao, Q. Yan, and Y. Amit, "Likelihood regret: An out-of-distribution detection score for variational auto-encoder," in *Proc. Int. Conf. Adv. Neural Inf. Process. Syst.*, 2020, pp. 20685–20696.
[47] E. Nalisnick, A. Matsukawa, Y. W. Teh, and B. Lakshminarayanan, "Detecting out-of-distribution inputs to deep generative models using typicality," 2019, *arXiv:1906.02994*.
[48] P. Kirichenko, P. Izmailov, and A. G. Wilson, "Why normalizing flows fail to detect out-of-distribution data," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2020, pp. 20578–20589.
[49] A. Paszke et al., "PyTorch: An imperative style, high-performance deep learning library," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2019, pp. 8026–8037.
[50] M. N. Finean, W. Merkt, and I. Havoutis, "Predicted composite signed-distance fields for real-time motion planning in dynamic environments," in *Proc. Int. Conf. Automated Plan. Scheduling*, 2021, vol. 31, pp. 616–624.
[51] F. Sabatino, "Quadrotor control: Modeling, nonlinear control design, and simulation," M.S. thesis, KTH Royal Inst. of Technol., Stockholm, Sweden, 2015.

Thomas Power received the M.Eng. degree in mechanical engineering from Imperial College London, London, U.K., in 2016, and the M.S. degree in robotics from the University of Michigan, Ann Arbor, MI, USA, in 2020. He is currently working toward the Ph.D. degree in robotics with the Autonomous Robotic Manipulation Laboratory, Department of Robotics, University of Michigan.

His research interests include trajectory optimization and machine learning applied to robotic manipulation.

Dmitry Berenson received the B.S. degree in electrical engineering from Cornell University, Ithaca, NY, USA, in 2005, and the Ph.D. degree in robotics from the Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, USA, in 2011.

Since 2016, he has been an Associate Professor with Robotics Department, University of Michigan, Ann Arbor, MI, USA. From 2011 to 2012, he was a Postdoc with UC Berkeley, Berkeley, CA, USA. From 2012 to 2016, he was an Assistant Professor with WPI. His research interests include robotic manipulation, robot learning, and motion planning.

Dr. Berenson was the recipient of the IEEE RAS Early Career Award and the NSF CAREER award.