Variational Inference MPC for Robot Motion with Normalizing Flows

Thomas Power
Robotics Institute
University of Michigan
Michigan, MI 48109
tpower@umich.edu

Dmitry Berenson
Robotics Institute
University of Michigan
Michigan, MI 48109
dmitryb@umich.edu

Abstract

In this paper, we propose an MPC method for robot motion by formulating MPC as Bayesian Inference. We propose using amortized variational inference to approximate the posterior with a normalizing flow conditioned on the start, goal and environment. By using a normalizing flow to represent the posterior, we are able to model complex distributions. This is important for robotics, where real environments impose difficult constraints on trajectories. We also present an approach for generalizing the learned sampling distribution to novel environments outside the training distribution. We demonstrate that our approach generalizes to a difficult novel environment and outperform a baseline sampling-based MPC method on a navigation problem.

1 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 "stochastic" systems, sampling based approaches for MPC such as CEM and MPPI [4, 1] 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 sampling low cost action sequences is unlikely and can become stuck in local minima, for example when a robot must find a path through a cluttered environment. Previous work has investigated the duality between control and inference [5, 6] and considered both planning and control as inference problems [7–9]. Several recent papers have considered the finite-horizon stochastic optimal control problem as Bayesian inference, and proposed methods of performing variational inference to approximate the posterior [10–13]. In order to perform variational inference, we must specify a parameterized distribution which is tractable to optimize and sample while also being flexible enough to provide a good approximation of the true posterior over low cost trajectories, which may exhibit strong dependencies and multimodalities. SVI-MPC approximates the posterior as a set of particles [10], whereas VI-MPC uses a mixture of Gaussians [12]. We propose using a normalizing flow to approximate the true posterior over trajectories. Normalizing flows have previously been used to parameterize variational posteriors in [14].

Contributions: We propose a method for learning a start, goal and environment conditioned distribution over finite horizon action sequences for MPC applied to navigation problems. Our method is able to generate diverse action sequences which display goal driven behavior for use downstream in a sampling-based MPC controller and does not require differentiable cost or dynamics. We also propose a method for generalizing this action sequence distribution to novel environments. We attempt to find an in-distribution environment from which we can sample high-performing action sequences evaluated in the actual novel environment. We demonstrate that our approach can

NeurIPS 2021 Workshop on Robot Learning: Self-Supervised and Lifelong Learning, Virtual, Virtual

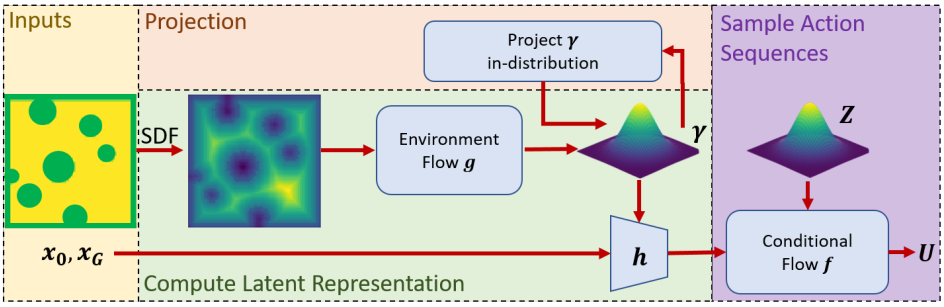

Figure 1: The overall process for sampling action sequences is shown above. The sampler takes as input initial & goal states x_0, x_G, and the environment, converted to a signed distance field (SDF). The SDF is processed by a normalizing flow g to produce a latent representation γ which is Gaussian-distributed. $p(\gamma)$ is used to measure how out-of distribution (OOD) an environment is and used to *project* γ in-distribution. (x_0, x_G, γ) are processed by a CNN h and passed to the conditional normalizing flow f. Finally, the conditional normalizing flow transforms samples from Gaussian distributed Z into sequences of actions.

generalize to difficult novel environments outside the training distribution where MPPI fails on a navigation problem. The outline of our approach is shown in Figure 1.

2 Methods

Finite-horizon Stochastic Optimal Control. We consider a discrete-time system with state $x \in \mathbb{R}^{d_x}$ and control $u \in \mathbb{R}^{d_u}$ and transition probability $p(x_{t+1}|x_t, u_t)$. We define finite horizon trajectories with horizon T as $\tau = (X, U)$, where $X = \{x_0, x_1, ...x_T\}$ and $U = \{u_0, u_1, ...u_{T-1}\}$. The goal is to find U which minimizes the expected cost $E_{p(X|U)}[J(\tau)]$ for 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 $J(x, u)$. For the purpose of the rest of the paper we will consider the case of collision-free navigation of an environment, where the cost function J is parameterized by (x_G, E), a goal state and an environment signed distance field (SDF) respectively.

Variational Inference for Stochastic Optimal Control. We can reformulate the above as an inference problem as in [9, 15, 12, 10]. First, we introduce a binary 'optimality' random variable o for a trajectory where $p(o = 1|\tau) \propto \exp(-J(\tau))$. We place a prior $p(U)$ on U, resulting in a prior on τ, $p(\tau) = p(X|U)p(U)$ and aim to find posterior distribution $p(\tau|o = 1) \propto p(o = 1|\tau)p(\tau)$. In general, this posterior is intractable, so we use variational inference to approximate it with a tractable distribution $q(\tau)$ which minimizes the KL-divergence $\mathcal{KL}(q(\tau)||p(\tau|o = 1))$ [16]. Since we define the trajectory by selecting the controls, the variational posterior factorizes as $p(X|U)q(U)$. Thus, we must compute an approximate posterior over action sequences. Minimizing $\mathcal{KL}(q(\tau)||p(\tau|o = 1))$ is equivalent to minimizing the variational free energy $\mathcal{F}$ (see Appendix A):

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

Where $\mathcal{H}(q(U)$ is the entropy of $q(U)$. Note that $\log p(U)$ imposes regularization on the control, and this can be appropriately combined with the cost, i.e. a Gaussian prior can be incorporated as a squared cost on the control, so will be omitted for the rest of the paper.

Variational Inference for Stochastic Optimal Control using Normalizing Flows. Normalizing flows can be used as a parameterization of the variational posterior [14]. 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 [14, 17, 18]. We use normalizing flows to define our variational posterior, i.e. $q(U) = f(Z)$ for $Z \sim p(Z) = \mathcal{N}(0, I)$. The variational free energy then becomes (see Appendix B.2):

$$\mathcal{F} = -\mathbb{E}_{q(\tau)}\left[\log p(o|\tau)\right] + \mathbb{E}_{p(Z)}\left[\log p(Z) - \log\left|\det \frac{\partial f}{\partial Z}\right|\right] \tag{2}$$

We can then optimize the flow parameters to minimize the free energy. However, this would mean optimizing a normalizing flow every time we would like to generate a trajectory, which is clearly too inefficient. Instead, we amortize by training a conditional normalizing flow [19] which is conditioned on the start, goal and an SDF of the environment, i.e. $q(U) = q(U|x_0, x_G, E)$ and the conditional normalizing flow is a function $U = f(Z, x_0, x_G, E)$ and is invertible with respect to Z, i.e. $Z = f^{-1}(U, x_0, x_G, E)$. We then use a dataset $\mathcal{D} = \{E, x_0, x_G\}^N$ and minimize the free energy via gradient descent. Further details on using normalizing flows for variational inference and conditional normalizing flows can be found in Appendices B and C, respectively.

Gradient Estimation. Minimizing eq. (2) via gradient descent requires the cost and dynamics to be differentiable. To avoid this, we estimate gradients, using the method in [12]. At each iteration, we sample from the normalizing flow and approximate $q(U)$ with K particles and weights $\{U_i, w_i\}$. The weights are initially uniform, but are updated according to the following update law:

$$w_i = \frac{q(U_i)^{-\beta} p(o = 1|\tau_i)^{\frac{1}{\alpha}}}{\frac{1}{K}\sum_{j=1}^{K} q(U_j)^{-\beta} p(o|\tau_j)^{\frac{1}{\alpha}}} \tag{3}$$

This update law is equivalent to performing mirror descent on the variational free energy, see [12] for further details. Here $\{\alpha, \beta\}$ are hyperparameters which control the trade-off between entropy and likelihood. We then train the normalizing flow by performing a single gradient step maximizing the log likelihood of this updated distribution, i.e. we maximize $\sum_{i=1}^{K} w_i \log q(U_i)$. Also, to avoid mode collapse in the flow, we apply a Gaussian perturbation to U_i after sampling from $q(U)$, this is demonstrated concretely in Algorithm 1 in Appendix E.

Generalizing to New Environments. A novel environment can be OOD for the conditional flow, and result in poor performance. This is because we learn over a dataset of environments; other methods that do not perform learning do not suffer from this issue. We present an approach where we *project* the OOD environment encoding in-distribution in order to produce low-cost trajectories when used as input to the flow. In order to do this, we first need some measure of OOD uncertainty to minimize. Our approach is to train a latent generative model over environments; in particular, we again use a normalizing flow to model $p(E)$ using a standard Normal latent distribution; $\gamma = g(E); \gamma \sim \mathcal{N}(0, I)$. We use a multiscale-prior for this flow such that $\gamma = [\gamma_1, \gamma_2]^T$ and γ_2 is transformed by additional flow layers [17]. We pass (γ_2, x_0, x_G) to be processed by a Convolutional Neural Network (CNN) h before being passed to the conditional flow, so that the conditional flow from above becomes $U = f\left(Z, h(x_0, x_g, \gamma_2)\right)$. Previous work has demonstrated that the likelihood from generative models is not necessarily indicative of an input being OOD. In fact, many generative models, including flows, can assign higher likelihood to OOD data [20–22]. However, it has been noted that representations which have been trained for a downstream task can yield more reliable OOD detection due to the presence of semantic information [22]. By training our entire system end-to-end, γ_2 is trained to contain an encoding of the environment relevant for trajectory generation. We thus define our OOD score as $C_{OOD} = p(\gamma_2) = \mathcal{N}(\gamma_2|0, I)$. This score measures whether or not the representation of the environment used for trajectory generation is in-distribution. We can perform gradient ascent on the above likelihood to find $\hat{\gamma}_2$, thus *projecting* the environment to be in-distribution. Note that this process will trivially drive toward the zero vector without regularization, for which we use the variational free energy loss from the previous section to ensure high trajectory quality. Using the normalizing flow, for a given $\hat{\gamma}_2$ we can sample γ_1 and reconstruct an SDF of the environment. Figures 2 (c,d) show an example of an original OOD SDF and its reconstructed projection.

FlowMPPI. We present a method for using the flow for a control task. The flow specifies an action sampling distribution $q(U)$ and can be used alongside many sampling based MPC approaches. We propose a way to use our method with MPPI [1]. MPPI iteratively perturbs a nominal trajectory with Gaussian noise and performs a weighted sum of the perturbations to find a new trajectory. We can run MPPI in the latent space of the flow, i.e. perturbing a nominal vector Z with Gaussian noise. Note that by perturbing a nominal vector we quickly leave the region of high density under $p(Z)$, so rather than sample perturbations to a nominal trajectory we instead sample trajectories directly from $p(Z)$. We then apply a perturbation cost on the distance of the sampled trajectory from the nominal in latent space. We found that the best results were produced by perturbing half of the samples in action space (as in standard MPPI), and half in the flow latent space. Additionally at each timestep we perform one gradient step to project the environment toward being in-distribution as described in the previous section. This is detailed in Algorithm 4 in Appendix E.

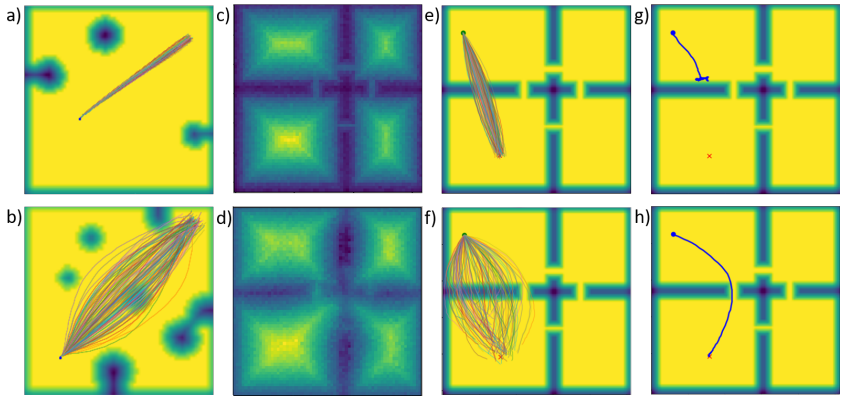

Figure 2: a,b) Rollouts of sampled action sequences. The entropy of the sampling distribution increases in the presence of obstacles. c, e) The SDF for a novel environment and the sampled trajectories. Despite the presence of an obstacle, the entropy of the sampling distribution is narrow and no collision free trajectories are generated. d,f) SDF after performing the projection. The entropy of the sampled trajectories has increased to account for the presence of the wall, and there are some collision-free sampled trajectories. g, h) Traversed trajectory of FlowMPPI in an OOD environment without (top) and with (bottom) using projection.

3 Evaluation

Environments We evaluate our proposed approach on a simple simulated experiment of controlling a double integrator system in the presence of obstacles (see Appendix G). Our training environment consists of spherical obstacles, where the size, location and number of obstacles is randomized. These environments and the outputs from the trained flow are shown in Fig. 2 (a,b). We define a testing environment which consists of four rooms with narrow entrances shown in Fig. 2 (e,f). This is a challenging environment for all sampling-based methods due to the narrow passages, and is also OOD for the proposed learned flow. From each environment we get a (64×64) SDF which we use to evaluate the cost. For each environment, the task is to successfully execute a trajectory from the start to the goal within 100 timesteps. The task is considered a failure if the point robot fails to get to a goal region defined by $\mathcal{X}_G = \{x : ||x - x_G||_2 < 0.1\}$, or if the point robot collides with an obstacle $(C(x_t) = 1)$ at any point during the executed trajectory. We define the trajectory cost as $J(\tau) = 100||x_T - x_G||_2 + \sum_{t=0}^{T-1} 10||x_t - x_G||_2 + 1000C(x_t)$, where T is the MPC horizon. The prior on actions is $p(U) = \mathcal{N}(0, I)$ which induces a cost on the squared magnitude of the actions.

Training & Data For training, we use 10000 randomly generated environments consisting of spherical obstacles. The algorithm for training is shown in Appendix E. At each epoch, for each environment, we randomly select one of 100 start and goal pairs. We train the Action flow f, the conditioning CNN h and the Environment Flow g end-to-end using Adam for 500 epochs with a learning rate of $1e^{-4}$.

	Spherical Obstacles In-Distribution		**Narrow Passages** Out-of Distribution	
Method	Success Rate	Av. Cost	Success Rate	Av. Cost
MPPI	0.89	1.0	0.26	1.0
FlowMPPI	0.96	**0.81**	0.52	0.86
FlowMPPI + proj.	**0.97**	0.84	**0.87**	**0.72**

Table 1: Comparison of methods on 100 randomly generated Spherical Obstacles and Narrow Passage environments with random starts and goals.

Results Our results in Table 1 demonstrate that our approach marginally outperforms MPPI in the training environments, but drastically outperforms MPPI in the challenging OOD test environments. Our proposed method of projecting the environment in-distribution leads to large further improvement. Further experimental and network architecture details are shown in Appendices G and F respectively. For future experiments we intend to evaluate our approach with more complex dynamics, higher dimensional state and action spaces, and with a 3D representation of the environment.

References

[1] Grady Williams, Paul Drews, Brian Goldfain, James M. Rehg, and Evangelos A. Theodorou. Information-Theoretic Model Predictive Control: Theory and Applications to Autonomous Driving. *IEEE Trans. Robot.*, 2018.

[2] Camille Brasseur, Alexander Sherikov, Cyrille Collette, Dimitar Dimitrov, and Pierre-Brice Wieber. A robust linear mpc approach to online generation of 3d biped walking motion. In *Humanoids*, 2015.

[3] Thomas Power and Dmitry Berenson. Keep it simple: Data-efficient learning for controlling complex systems with simple models. *IEEE RA-L*, 2021.

[4] Marin Kobilarov. Cross-entropy motion planning. *IJRR*, 2012.

[5] Emanuel Todorov. General duality between optimal control and estimation. In *CDC*, 2008.

[6] Evangelos A. Theodorou and Emanuel Todorov. Relative entropy and free energy dualities: Connections to Path Integral and KL control. In *CDC*, 2012.

[7] Hagai Attias. Planning by Probabilistic Inference. In *AISTATS*, 2003.

[8] Marc Toussaint and Amos Storkey. Probabilistic inference for solving discrete and continuous state Markov Decision Processes. In *ICML*, 2006.

[9] Konrad Rawlik, Marc Toussaint, and Sethu Vijayakumar. On stochastic optimal control and reinforcement learning by approximate inference. In *IJCAI*, 2013.

[10] Alexander Lambert, Adam Fishman, Dieter Fox, Byron Boots, and Fabio Ramos. Stein Variational Model Predictive Control. In *CoRL*, 2020.

[11] Ziyi Wang, Oswin So, Jason Gibson, Bogdan Vlahov, Manan Gandhi, Guan-Horng Liu, and Evangelos Theodorou. Variational Inference MPC using Tsallis Divergence. In *RSS*, 2021.

[12] Masashi Okada and Tadahiro Taniguchi. Variational Inference MPC for Bayesian Model-based Reinforcement Learning. In *CoRL*, 2020.

[13] Lucas Barcelos, Alexander Lambert, Rafael Oliveira, Paulo Borges, Byron Boots, and Fabio Ramos. Dual Online Stein Variational Inference for Control and Dynamics. In *RSS*, 2021.

[14] Danilo Rezende and Shakir Mohamed. Variational inference with normalizing flows. In *ICML*, 2015.

[15] Marc Toussaint. Robot trajectory optimization using approximate inference. In *ICML*, 2009.

[16] David M. Blei, Alp Kucukelbir, and Jon D. McAuliffe. Variational inference: A review for statisticians. *Journal of the American Statistical Association*, 2017.

[17] Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. Density estimation using real NVP. In *ICLR*, 2017.

[18] Durk P Kingma and Prafulla Dhariwal. Glow: Generative flow with invertible 1x1 convolutions. In *NeurIPS*, 2018.

[19] Christina Winkler, Daniel Worrall, Emiel Hoogeboom, and Max Welling. Learning likelihoods with conditional normalizing flows, 2019.

[20] Eric T. Nalisnick, Akihiro Matsukawa, Yee Whye Teh, Dilan Görür, and Balaji Lakshminarayanan. Do deep generative models know what they don't know? In *ICLR*, 2019.

[21] Eric T. Nalisnick, Akihiro Matsukawa, Yee Whye Teh, and Balaji Lakshminarayanan. Detecting out-of-distribution inputs to deep generative models using a test for typicality. *CoRR*, abs/1906.02994, 2019. URL `http://arxiv.org/abs/1906.02994`.

[22] Polina Kirichenko, Pavel Izmailov, and Andrew Gordon Wilson. Why normalizing flows fail to detect out-of-distribution data. In *NeurIPS*, 2020.

[23] Normalizing flows for image modeling. `https://uvadlc-notebooks.readthedocs.io/en/latest/tutorial_notebooks/tutorial11/NF_image_modeling.html`, 2020. Accessed: 2021-09-23.

[24] Jonathan Ho, Xi Chen, Aravind Srinivas, Yan Duan, and Pieter Abbeel. Flow++: Improving flow-based generative models with variational dequantization and architecture design. In *ICML*, 2019.

[25] Wenling Shang, Kihyuk Sohn, Diogo Almeida, and Honglak Lee. Understanding and improving convolutional neural networks via concatenated rectified linear units. In *ICML*, 2016.

A Variational Inference for Finite-Horizon Stochastic Optimal Control

The variational posterior over trajectories is defined by the dynamics and the variational posterior over actions:

$$q(\tau|x_0) = q(X, U|x_0) = p(X|U, x_0)q(U) = \prod_{t=0}^{T} p(x_{t+1}|x_t, u_t)q(u_t) \tag{4}$$

We will omit the dependence on the initial state x_0 for convenience.

$$\mathcal{KL}\left(q(\tau)||p(\tau|o=1)\right) = \int q(\tau) \log \frac{q(\tau)}{p(\tau|o=1)} d\tau \tag{5}$$

$$= \int q(X, U) \log \frac{p(X|U)q(U)p(o=1)}{p(o=1|X, U)p(X|U)p(U)} dX\, dU \tag{6}$$

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

$$\mathcal{F} = \int q(X, U) \log \frac{q(U)}{p(o=1|X, U)p(U)} dX\, dU \tag{7}$$

$$= -\mathbb{E}_{q(X,U)} \left[\log p(o=1|X, U) + \log p(U) - \log q(U)\right] \tag{8}$$

$$= \mathbb{E}_{q(X,U)} \left[J(X, U)\right] + \mathcal{KL}(q(U)||p(U)) \tag{9}$$

$$= \mathbb{E}_{q(X,U)} \left[\hat{J}(X, U) + \log q(U)\right] \tag{10}$$

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

B Variational Inference with Normalizing Flows

B.1 Normalizing flows

We closely follow the exposition given in [14]. Consider a random variable $z \in \mathbb{R}^d$ and with known pdf $p(z)$. If we define a bijective function $f : \mathbb{R}^d \to \mathbb{R}^d$ and define a random variable x such that $x = f(z)$ and $x = f^{-1}(x)$. According to the change of variable formula, we can define $p(x)$ in terms of $p(z)$ as follows:

$$p(x) = p(z) \left|\det \frac{\partial f}{\partial z}\right|^{-1} \tag{11}$$

$$\log p(x) = \log p(z) - \log \left|\det \frac{\partial f}{\partial z}\right| \tag{12}$$

$p(z)$ can be chosen to be a simple distribution such as a Normal or Uniform distribution. By choosing an appropriate f we can represent complex and multi-modal densities $p(x)$.

B.2 VI using normalizing flows

By selecting a base pdf $p(z)$ and a family of functions parameterized f_θ, we specify a potentially complex set of possible densities $q_\theta(x)$. Suppose that we want to approximate some distribution $p(x)$ with some distribution $q_\theta(x)$. The variational objective is to minimize $\mathcal{KL}(q_\theta(x)||p(x))$. This is equivalent to:

$$\mathcal{KL}\left(q_\theta(x)||p(x)\right) = \int q_\theta(x) \log \frac{q_\theta(x)}{p(x)} dx \tag{13}$$

$$= \mathbb{E}_{q_\theta(x)}[\log q_\theta(x) - \log p(x)] \tag{14}$$

$$= \mathbb{E}_{p(z)} \left[\log p(z) - \log \left|\det \frac{\partial f_\theta}{\partial z}\right| - \log p(x)\right] \tag{15}$$

Thus we can optimize parameters of θ of the bijective transform f in order to minimize the variational objective.

C Conditional Normalizing Flows

We use a modification of the conditional normalizing flow detailed in [19]. We first note that we use the invertible transforms described in [17] to define the flow. For defining invertible flow $x = f(z)$ $(x, z \in \mathbb{R}^d)$ we define:

$$x_{1:\frac{d}{2}} = z_{1:\frac{d}{2}} \tag{16}$$

$$x_{\frac{d}{2}:d} = z_{\frac{d}{2}:d} \circ s(z_{1:\frac{d}{2}}) + t(z_{1:\frac{d}{2}}) \tag{17}$$

Where $x_{1:\frac{d}{2}}$ denotes the first $\frac{d}{2}$ elements of x and $\circ$ denotes the hadamard product, and s, t are neural networks. This defines an invertible transform on half of z. By chaining these transforms together with suitable permutation or masking we can develop complex invertible transforms on the full vector. These transforms can be made conditional on context vector $C \in R^n$ by the following:

$$x_{1:\frac{d}{2}} = z_{1:\frac{d}{2}} \tag{18}$$

$$x_{\frac{d}{2}:d} = z_{\frac{d}{2}:d} \circ s(z_{1:\frac{d}{2}}, C) + t(z_{1:\frac{d}{2}}, C) \tag{19}$$

Which preserves the invertibility while allowing the addition of contextual information. While the original paper [19] makes use of conditional priors where the latent $z \sim p(z|C)$. We do not do this as we found that this prior becomes extremely peaked, and this presents difficulty for searching in the latent space for action sequences during control.

D Gradient-free updates

Minimizing eq. (10) via gradient descent requires gradients of the cost and dynamics functions with respect to the state and actions. This may not be always available, particularly when the cost includes checking for collisions. Instead we follow the derivation in [12] to optimize the action flow without these gradients. The full derivation can be found in the original paper, but performing mirror descent on the variational objective gives an update law for $q(U)$ as:

$$\hat{q}(U) = \frac{q(U)^{1-\beta} \cdot p(o=1|\tau)^{\frac{1}{\alpha}}}{\int q(U)^{1-\beta} \cdot p(o=1|\tau)^{\frac{1}{\alpha}} dU} = \frac{q(U)^{1-\beta} \cdot p(o=1|\tau)^{\frac{1}{\alpha}}}{\mathbb{E}_{q(U)}\left[q(U)^{-\beta} \cdot p(o=1|\tau)^{\frac{1}{\alpha}}\right]} \tag{20}$$

For a distribution q(U) represented as a set of K weighted particles with weight $w_i{}_{i=1}^K$ we can formulate this as an update formula, where we assume the initial weights are uniform:

$$w_i = \frac{q(U_i)^{-\beta} \cdot p(o=1|\tau_i)^{\frac{1}{\alpha}}}{\frac{1}{K}\sum_{j=1}^K q(U_j)^{-\beta} \cdot p(o|\tau_j)^{\frac{1}{\alpha}}} \tag{21}$$

E Algorithm Details: FlowMPPI

Algorithm 1 Sample from Action Flow with Perturbation

function SAMPLEPERTURBEDACTIONS($x_0, x_G, \gamma_2, f, h, \Sigma_\epsilon, K$)
  for i $\in \{k, ..., K\}$ **do**
    $Z_k \sim \mathcal{N}(0, I)$
    $\epsilon_k \sim \mathcal{N}(0, \Sigma_\epsilon)$
    $U_k \leftarrow f(Z_k, h\,(x_0, x_G, \gamma_2)) + \epsilon_k$
    $\hat{Z}_k \leftarrow f^{-1}(U_k, h\,(x_0, x_G, \gamma_2))$
    $q(U_k) \leftarrow$ from $\hat{Z}_k$ via eq. (12)
  return $\{U_k, q(U_k)\}_{k=1}^K$

Algorithm 2 Flow Training

Inputs: N iterations, K samples, θ^1 initial parameters for (f, g, h), (x_0, x_G, E), action perturbation sigma Σ_ϵ, learning rate η, loss hyperparameters (α, β)

for n $\in \{1, ..., N\}$ **do**
  $\gamma_1, \gamma_2 \leftarrow g(E)$
  $\log p(E) \leftarrow$ from (γ_1, γ_2) via eq. (12)
  $\{U_k, q(U_k)\}_{k=1}^K \leftarrow$ SAMPLEPERTURBEDACTIONS$(x_0, x_G, \gamma_2, f, h, \Sigma_\epsilon, K)$
  $\mathcal{L} \leftarrow -\log p(E)$
  for k $\in \{1, ..., K\}$ **do**
    $w_k \leftarrow$ from $(\{U_i, q(U_i)\}_{i=1}^K, \alpha, \beta\})$ via eq. (3)
    $\mathcal{L} \leftarrow \mathcal{L} - w_k \cdot \log q(U_k)$
  $\theta^{n+1} \leftarrow \theta^n - \eta \frac{\partial \mathcal{L}}{\partial \theta}$

Algorithm 3 Projection

function PROJECT$(x_0, x_G, E, f, g, h, \alpha, \beta, \eta, \Sigma_\epsilon, K)$
  $\gamma_1, \gamma_2 \leftarrow g(E)$
  $\hat{\gamma}_2 \leftarrow \gamma_2$
  $\{U_k, q(U_k)\}_{k=1}^K \leftarrow$ SAMPLEPERTURBEDACTIONS$(x_0, x_G, \hat{\gamma}_2, f, h, \Sigma_\epsilon, K)$
  $\mathcal{L} \leftarrow -\log p(\hat{\gamma}_2)$
  for k $\in \{1, ..., K\}$ **do**
    $w_k \leftarrow$ from $(\{U_i, q(U_i)\}_{i=1}^K, \alpha, \beta\})$ via eq. (3)
    $\mathcal{L} \leftarrow \mathcal{L} - w_k \cdot \log q(U_k)$
  $\hat{\gamma} \leftarrow \hat{\gamma} - \eta \frac{\partial \mathcal{L}}{\partial \hat{\gamma}}$
  return $\hat{\gamma}_2$

Algorithm 4 A single step of FlowMPPI, this will run every timestep until task is completed or failure is reached.

Inputs: Loss function J, previous nominal trajectory U, Start x_0, Goal x_G, Environment SDF E, Action Flow f, Environment flow g, Context CNN h, MPPI hyperparameters (λ, Σ), Horizon T, Samples K, Flow loss hyperparameters (α, β), Learning rate η

function FLOWMPPISTEP$(x_0, x_G, E, f, g, h, J, \alpha, \beta, \eta, \lambda, \Sigma, \Sigma_\epsilon, K)$
  $\hat{\gamma}_2 \leftarrow$ PROJECT$(x_0, x_G, E, U, f, g, h, \alpha, \beta, \eta, \Sigma_\epsilon, K)$
  for t $\in \{1, ..., T-1\}$ **do**
    $U_{t-1} \leftarrow U_t$
  $U_{T-1} \sim \mathcal{N}(0, \Sigma)$
  $Z \leftarrow f^{-1}(U, h(x_0, x_G, \hat{\gamma_2}))$
  for k $\in \{1, ..., \frac{K}{2}\}$ **do**
    $\epsilon_U \sim \mathcal{N}(0, \Sigma)$
    $U_k \leftarrow U + \epsilon_U$
    $\tau_k \sim p(\tau | U_k)$
    $C_k \leftarrow J(\tau_k) + \lambda U_k \Sigma^{-1} \epsilon_U$
  for k $\in \{\frac{K}{2} + 1, ..., K\}$ **do**
    $\epsilon_Z \sim \mathcal{N}(0, I)$
    $U_k \leftarrow f(\epsilon_Z, h(x_0, x_g, \hat{\gamma}_2))$
    $\tau_k \sim p(\tau | U_k)$
    $C_k \leftarrow J(\tau_k) + \lambda \epsilon_Z (Z - \epsilon_Z)$
  $\beta \leftarrow \min_k C_k$
  $\eta = \sum_{k=1}^K \exp(-\frac{1}{\lambda}(C_k - \beta))$
  for k $\in \{1, ..., K\}$ **do**
    $w_k \leftarrow \frac{1}{\eta} \exp(-\frac{1}{\lambda}(C_k - \beta))$
  $U \leftarrow \sum_{k=1}^K w_k U_k$
  return U

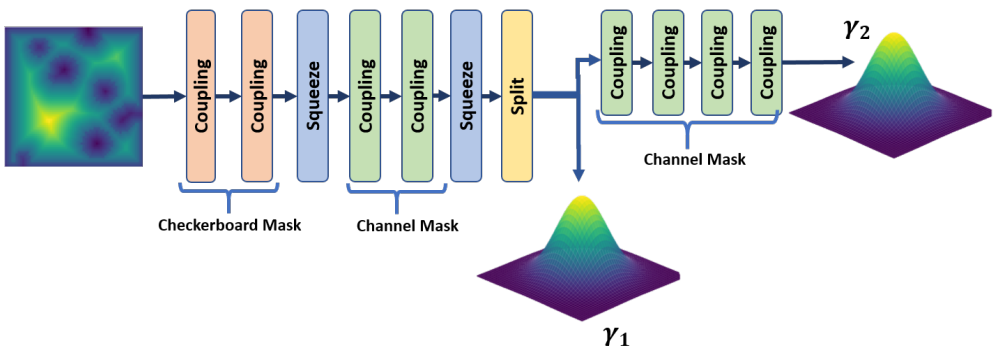

Figure 3: Architecture for Environment flow g

F Network Architectures

The proposed system contains three networks. The action flow f, the environment flow g and the conditioning CNN h.

Environment flow g Recall that the environment flow maps $\gamma = g(E)$, with $p(\gamma) = \mathcal{N}(0, I)$. We use the flow architecture outlined in [23], which is inspired by [24] and [17]. The input is a 64×64 SDF. This SDF is calculated from a 64×64 occupancy grid. As such the SDF takes on discrete values (due to the distances being calculated pixel-wise). Therefore we must perform dequantization to make the distribution over SDF values continuous. Thus we add small Gaussian noise with $\sigma = 0.02$ to the SDF.

We use coupling layers as described in Appendix C. The nonlinear transformations s, t are defined by Gated Convolutional ResNets as described in [24] with (3×3) kernels and 2 layers. We use the `ConcatElu` activation function [25]. Each transformation operating on half of the image at a time, the half that is operated on is defined by applying a mask to the image. These masks are either a checkerboard mask, or a channel-wise mask. We also use a `Squeeze` operation which squeezes an image from $c \times w \times h$ to $4 \cdot c \times \frac{w}{2} \times \frac{h}{2}$. We use a hierarchical prior which splits an imagine into two equal parts channel wise `Split` : $(c \times w \times h) \rightarrow (\frac{c}{2} \times w \times h), (\frac{c}{2} \times w \times h)$. After splitting one the two half-images can be evaluated on the prior.

The flow transforms $E \in \mathbb{R}^{64 \times 64}$ to $\gamma_1, \gamma_2 \in \mathbb{R}^{8 \times 16 \times 16}$. The full architecture is shown in Figure 3.

Conditioning CNN h The conditioning CNN h takes as input x_0, x_G, γ_2 and outputs a context vector $c \in \mathbb{R}^{d_c}$ which is then used in the action flow. First we use three Convolutional layers with ReLu activation functions and 32 channels and (3×3) kernels to process γ_2, which is a multi-channel image, followed by a single linear to map the environment encoding to a vector of dimension 256. This is then concatenated with x_0, x_G and processed with a 2 layer neural network with ReLU activations and a hidden size of 256 to produce c. For our experiment $d_c = 256$.

Action flow f The action flow represents distribution $q(U|x_0, x_G, E)$ as a function f transforming $Z \sim \mathcal{N}(0, I)$. (x_0, x_G, E) are represented by the context vector c described in the previous section, thus $U = f(Z, c)$ and the inverse w.r.t Z is $Z = f^{-1}(X, c)$. The context vector is used in the coupling layer as described in Appendix C. The action sequence $U \in \mathbb{R}^{T \times d_u}$, and we use a flattened $Z \in \mathbb{R}^{T \cdot du}$. Rather than defining masks as in the previous section, we use the invertible 1×1 convolution introduced in [18] which includes a random permutation. The action flow f is defined as 10 coupling layers separated by a 1×1 convolution permutation layer.

G Experimental Details

System Dynamics For our test environment we use a 2D double integrator with damping, with the following state and dynamics, control $u \in \mathbb{R}^2$ and $\Delta t = 0.05s$. For our experiments we use a horizon of $T = 40$.

$$\begin{bmatrix} x_{t+1} \\ y_{t+1} \\ \dot{x}_{t+1} \\ \dot{y}_{t+1} \end{bmatrix} = \begin{bmatrix} 1 & 0 & \Delta t & 0 \\ 0 & 1 & 0 & \Delta t \\ 0 & 0 & 0.95 & 0 \\ 0 & 0 & 0 & 0.95 \end{bmatrix} \begin{bmatrix} x_t \\ y_t \\ \dot{x}_t \\ \dot{y}_t \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ \Delta t & 0 \\ 0 & \Delta t \end{bmatrix} \mathbf{u} \tag{22}$$

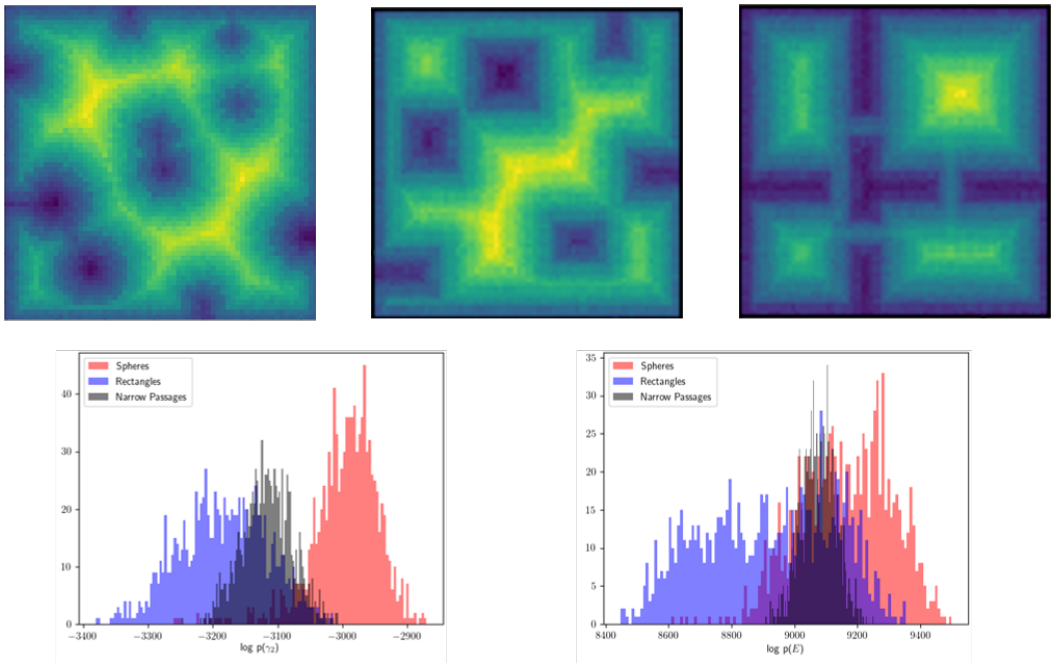

Figure 4: Top row: Three example environments. Left to right, Spherical obstacles, Rectangular obstacles, Environment with Narrow passages. Bottom row: Measuring OOD of generative flow trained on environment with spherical obstacles. Left: We see that using only γ_2, the environment representation used for planning, we get substantially better separation of in-distribution vs out of distribution data vs using the overall likelihood of the environment on the right.

Training We train with 10000 64×64 environment SDFs and 100 start, goal pairs per environment. At each iteration when training on one environment, we randomly sample a single start and goal pair. We train with Adam and learning rate of $1e^{-4}$, and $L2$ regularization with strength $1e^{-5}$. During training when evaluating $J(\tau_k) - \log p(U_k)$ we normalize it to be in the range $[0, 1]$, we also normalize $q(U_k)$ to be in the range $[-1, 0]$. We use hyperparameters $\alpha = 0.01, \beta = 1$ when training the flow. Additionally, we anneal the perturbation applied to samples of the flow, decaying their magnitude to zero, i.e. $\Sigma_\epsilon = I \cdot \max(\min(1.0, (1.0 - \frac{\text{epoch}}{\#\text{epochs}})), 0.0)$.

Projection When performing projection inside FLOWMPPISTEP we perform one gradient step of projection at each timestep. The learning rate when performing projection is $1e^{-2}$. Figure 4 shows the OOD score for different datasets. Since for our dataset, the OOD score is a maximum of around -2900, we clamp the OOD score during projection to be no higher than this. This is to prevent the OOD score from driving the projected environment further towards the zero vector and away from the in-distribution data. During projection, we again use $\alpha = 0.01, \beta = 1.0$, and we use an action perturbation of $\Sigma_\epsilon = 1$.