Learning Sampling Distributions for Model Predictive Control

Anonymous Author(s)
Affiliation
Address
email

Abstract: Sampling-based methods have become a cornerstone of contemporary approaches to Model Predictive Control (MPC), as they make no restrictions on the differentiability of the dynamics or cost function and are straightforward to parallelize. However, their efficacy is highly dependent on the quality of the *sampling distribution itself*, which is often assumed to be simple, like a Gaussian. This restriction can result in samples which are far from optimal, leading to poor performance. Recent work has explored improving the performance of MPC by sampling in a learned latent space of controls. However, these methods ultimately perform all MPC parameter updates and warm-starting between time steps in the control space. This requires us to rely on a number of heuristics for generating samples and updating the distribution and may lead to sub-optimal performance. Instead, we propose to carry out all operations in the latent space, allowing us to take full advantage of the learned distribution. Specifically, we frame the learning problem as bi-level optimization and show how to train the controller with backpropagation-through-time. By using a normalizing flow parameterization of the distribution, we can leverage its tractable density to avoid requiring differentiability of the dynamics and cost function. Finally, we evaluate the proposed approach on simulated robotics tasks and demonstrate its ability to surpass the performance of prior methods and scale better with a reduced number of samples.

Keywords: Model Predictive Control, Normalizing Flows

1 Introduction

Sequential decision making under uncertainty is a fundamental problem in machine learning and robotics. Recently, model predictive control (MPC) has emerged as a powerful paradigm to tackle such problems on real-world robotic systems. In particular, MPC has been successfully applied to helicopter aerobatics [1], robot manipulation [2, 3, 4], humanoid robot locomotion [5], robot-assisted dressing [6], and aggressive off-road driving [7, 8, 9]. Sampling-based approaches to MPC are becoming particularly popular due to their simplicity and ability to handle non-differentiable dynamics and cost functions. These methods work by sampling controls from a policy distribution and rolling out an approximate system model using the sampled control sequences. They then use the resulting trajectories to compute an approximate gradient of the cost function and update the policy. The controller then samples an action from this distribution and applies it to the system, and it repeats the process from the resulting next state, creating a feedback controller. It warm-starts the optimization process using a modification of the solution at the previous time step.

An important design decision is the form of the sampling distribution, which is often simple, e.g. a Gaussian, such that we can efficiently sample and tractably update its parameters. However, this also has drawbacks: without much control over the distribution form, samples often lie in high-cost regions, hindering performance. This can be particularly problematic in complex environments with sparse costs or rewards, as a poorly parameterized distribution may hinder efficient exploration, leading the system into bad local optima. A side effect is that we often require many samples to accomplish the objective, increasing computational requirements. There have been extensions which

Submitted to the 6th Conference on Robot Learning (CoRL 2022). Do not distribute.

target more complex distributions, such as Gaussian mixture models [10] and a particle method based on Stein variational gradient descent (SVGD) [11]. However, there is a large amount of structure in the environment that these methods fail to exploit. Instead, an alternative approach is to *learn* a sampling distribution which can leverage the environmental structure to draw more optimal samples.

Prior work on learning MPC sampling distributions generally requires differentiability of the dynamics and cost function [12, 13]. Power and Berenson [14, 15] circumvent this issue by leveraging normalizing flows (NFs) [16, 17, 18], which have a tractable log-likelihood. This property allows them to learn flexible distributions by directly optimizing the MPC cost without requiring differentiability via the likelihood-ratio gradient. However, a limitation of their approach is that all online updates to the distribution and warm-starting between time steps occur entirely in the control space, leaving the latent space fixed. This forces them to apply heuristics to generate samples by combining those from the learned distribution with Gaussian perturbations to the current control-space mean. These restrictions prevent us from fully taking advantage of the learned distribution and potentially throws away useful information. Additionally, their approach does not allow for the incorporation of control constraints directly into the sampling distribution, which is important for real-world robots.

Instead, we propose to alter the optimization machinery to operate entirely in the latent space. As the NF latent space follows a simple distribution, it remains feasible to perform MPC updates in this learned space and update the latent distribution online. Specifically, during an episode, the parameters of the latent distribution are updated with MPC while those of the NF remain fixed. Then during training, after each episode, the parameters of the NF are updated. We can frame this setup as a bi-level optimization problem [19] and derive a method for computing an approximate gradient through the latent MPC update. This involves treating MPC as a recurrent network, where the control distribution acts as a form of memory, and unrolling the computation to train with backpropagation-through-time (BPTT). However, it is no longer clear how to warm-start between time steps because there is no clear delineation of time in the latent space. Moreover, the usual method of warm-starting, which simply shifts the current plan forward in time, may be sub-optimal. Therefore, we propose to learn a shift model, which performs all warm-starting operations in the latent space. Finally, we show how to alter the NF architecture to incorporate box constraints on the sampled controls.

Contributions: In this work, we build upon recent efforts to learn sampling distributions for MPC with NFs by moving all online parameter updates and warm-start operations into the the latent space. We accomplish this by framing the learning problem as bi-level optimization and derive an approximate gradient through the MPC update of the latent distribution in order to train the network with BPTT. Additionally, we show how to parameterize the flow architecture such that we can incorporate box constraints on the controls. Finally, we empirically evaluate our proposed approach on simulated robotics tasks, including both navigation and manipulation problems. In both cases, we demonstrate its ability to improve performance over all baselines by taking full advantage of the learned latent space. Furthermore, we find that the performance of the controllers with our learned sampling distributions scales more gracefully with a reduction in the number of samples.

2 Sampling-Based Model Predictive Control

We consider the problem of controlling a discrete-time stochastic dynamical system, which is in state $x_t \in \mathbb{R}^N$ at time step t. Upon the application of control $u_t \in \mathbb{R}^M$, the system incurs the instantaneous cost $c(x_t, u_t)$ and transitions to the next state x_{t+1} according to the dynamics $x_{t+1} \sim f(x_t, u_t)$. We wish to design a state-feedback policy $u_t \sim \pi(\cdot|x_t)$ such that the system achieves good performance over T steps. Instead of finding a single, globally optimal policy, MPC re-optimizes a local policy at each time step by predicting the system's behavior over a finite horizon $H < T$ using an approximate model $\hat{f}$. Specifically, in sampling-based MPC, we generate a sequence of controls by sampling $\hat{\boldsymbol{u}}_t \sim \pi_\theta(\cdot)$, where $\hat{\boldsymbol{u}}_t \triangleq (\hat{u}_t, \hat{u}_{t+1}, \cdots, \hat{u}_{t+H-1})$ and our policy is parameterized by $\theta \in \Theta$. We rollout our model starting at our current state x_t using these sampled controls to get our predicted state sequence $\hat{\boldsymbol{x}}_t \triangleq (\hat{x}_t, \hat{x}_{t+1}, \cdots, \hat{x}_{t+H})$, with $\hat{x}_t = x_t$. The total trajectory cost is

$$C(\hat{\boldsymbol{x}}_t, \hat{\boldsymbol{u}}_t) = \sum_{h=0}^{H-1} c(\hat{x}_{t+h}, \hat{u}_{t+h}) + c_{term}(\hat{x}_{t+H}), \tag{1}$$

where $c_{term}(\cdot)$ is a terminal cost function. We then define a statistic $\hat{J}(\theta; x_t)$ defined on cost $C(\hat{\boldsymbol{x}}_t, \hat{\boldsymbol{u}}_t)$ and use the rollouts to solve $\theta_t \leftarrow \arg\min_{\theta \in \Theta} \hat{J}(\theta; x_t)$. After optimizing our parameters, we sample the control sequence $\hat{\boldsymbol{u}}_t \sim \pi_{\theta_t}(\cdot)$, apply the first control to the real system (i.e. $u_t = \hat{u}_t$), and repeat the process. Because each parameter θ_t depends on the current state, MPC effectively yields a state-feedback policy, even though the individual distributions give us an open-loop sequence.

In this paper, we consider a popular sampling-based MPC algorithm known as Model Predictive Path Integral (MPPI) control [7, 8]. MPPI optimizes the exponential utility or risk-seeking objective:

$$\hat{J}(\theta; x_t) = -\log \mathbb{E}_{\pi_\theta, \hat{f}}\Big[\exp\Big(-\frac{1}{\beta} C(\hat{\boldsymbol{x}}_t, \hat{\boldsymbol{u}}_t)\Big)\Big], \tag{2}$$

where $\beta > 0$ is a scaling parameter, known as the temperature. As we do not assume that the dynamics or cost function are differentiable, we compute the gradients via the likelihood-ratio derivative:

$$\nabla \hat{J}(\theta; x_t) = -\frac{\mathbb{E}_{\pi_\theta, \hat{f}}\Big[e^{-\frac{1}{\beta} C(\hat{\boldsymbol{x}}_t, \hat{\boldsymbol{u}}_t)} \nabla_\theta \log \pi_\theta(\hat{\boldsymbol{u}}_t)\Big]}{\mathbb{E}_{\pi_\theta, \hat{f}}\Big[e^{-\frac{1}{\beta} C(\hat{\boldsymbol{x}}_t, \hat{\boldsymbol{u}}_t)}\Big]}. \tag{3}$$

In MPPI, the policy is assumed to be a factorized Gaussian of the form

$$\pi_\theta(\hat{\boldsymbol{u}}) = \prod_{h=0}^{H-1} \pi_{\theta_h}(\hat{u}_{t+h}) = \prod_{h=0}^{H-1} \mathcal{N}(\hat{u}_{t+h}; \mu_{t+h}, \Sigma_{t+h}). \tag{4}$$

Previous work by Wagener et al. [9] has shown that optimizing this objective with dynamic mirror descent (DMD) [20] and approximating with Monte Carlo estimates gives us the MPPI update rule:

$$\mu_{t+h} = (1 - \gamma_t^\mu)\tilde{\mu}_{t+h} + \gamma_t^\mu \sum_{i=1}^{N} w_i \hat{u}_{t+h}^{(i)}, \quad w_i = \frac{e^{-\frac{1}{\beta} C(\hat{\boldsymbol{x}}_t^{(i)}, \hat{\boldsymbol{u}}_t^{(i)})}}{\sum_{j=1}^{N} e^{-\frac{1}{\beta} C(\hat{\boldsymbol{x}}_t^{(j)}, \hat{\boldsymbol{u}}_t^{(j)})}} \tag{5}$$

where $\tilde{\mu}_{t+h}$ is the current mean for each time step and γ_t^μ is the step size. Between time steps of DMD, we get $\tilde{\mu}_{t+h}$ from our previous solution μ_{t+h} by using a shift model $\tilde{\mu}_{t+h} = \Phi(\mu_{t+h})$. This shift model aims to predict the optimal decision at the next time step given the previous solution. In the context of MPC, it allows us to warm-start the optimization problem to speed up convergence, as we can only approximately solve the optimization problem due to real-time constraints.

3 Learning the Sampling Distribution of MPC

3.1 Representation of the Learned Distribution

Instead of using uninformed sampling distributions, learned distributions can potentially exploit structure in the environment to draw samples which are more likely to be collision-free and close to optimal. However, such learned distributions must be sufficiently expressive in order to better capture near-optimal, potentially multimodal, behavior. They must also be parameterized such that it is tractable to sample from and update online. If the distribution has a large number of parameters, the number of samples required to efficiently update them online may be computationally infeasible. And ideally, the form of our distribution would be such that we could find a closed-form update.

One path towards meeting these criteria is to maintain a simple latent distribution from which we can sample, and then learn a transformation of the samples which maps them to a more complex distribution. During training, we learn the parameters of this transformation, which can be conditioned on problem-specific information, such as the starting and goal configurations of the robot and obstacle placements. However, when executing the policy during an episode, the parameters of this learned transformation remain fixed, and instead, we update the parameters of the latent distribution. Concretely, we consider learning a distribution $\pi_{\theta,\lambda}$ which is defined implicitly as follows:

$$\hat{\boldsymbol{z}}_t \sim p_\theta(\cdot), \quad \hat{\boldsymbol{u}}_t = h_\lambda(\hat{\boldsymbol{z}}_t; c) \tag{6}$$

where $\hat{\boldsymbol{z}}_t \triangleq (\hat{z}_t, \hat{z}_{t+1}, \cdots, \hat{z}_{t+H-1})$, c is a context variable describing the relevant information of the environment, p_θ is the latent distribution with parameters θ, and h_λ is the learned conditional transformation with parameters λ. Moving forward, we assume that both $\hat{\boldsymbol{z}}_t$ and $\hat{\boldsymbol{u}}_t$ are stacked as vectors in $\mathbb{R}^{MH}$. If p_θ is a Gaussian factorized as in Equation (4) and we assume that h_λ is invertible, we prove in Appendix A.5 that the corresponding DMD update to the latent mean is simply Equation (5), except that we replace the controls in the weighted sum with the latent samples.

3.2 Formulating the Learning Problem

Learning the distribution $\pi_{\theta,\lambda}$ amounts to solving a bi-level optimization problem [19], in which one optimization problem is nested in another. The lower-level optimization problem involves updating the latent distribution parameters at each time step, θ_t, by minimizing the expected cost with DMD. The upper-level optimization problem consists of learning λ such that MPC performs well across a number of different environments. To formalize this, first consider that we have some distribution of environments $c \sim \mathcal{C}(\cdot)$ over which we wish MPC to perform well. For each environment, our system has some conditional initial state distribution $x_0 \sim \rho(\cdot|c)$. The objective we wish to minimize is then

$$\ell(\boldsymbol{\theta}, \lambda; c) = \mathbb{E}_{\boldsymbol{\pi}_{\boldsymbol{\theta},\lambda},\rho,f}\left[\sum_{t=0}^{T-1} \hat{J}(\theta_t, \lambda; x_t, c)\right] \tag{7}$$

where $\boldsymbol{\theta} = (\theta_0, \theta_1, \cdots, \theta_{T-1})$ and our cost statistic, $\hat{J}$, now depends on λ and c as well. This objective measures the expected performance of the intermediate plans produced by MPC along the T steps of the episode. Our desired bi-level optimization problem can be formulated as:

$$\min_{\lambda} \ \mathbb{E}_{\mathcal{C}}\Big[\ell(\boldsymbol{\theta}(\lambda), \lambda; c)\Big] \quad \text{s.t. } \boldsymbol{\theta}(\lambda) \approx_{\lambda} \arg\min_{\boldsymbol{\theta}} \ell(\boldsymbol{\theta}, \lambda; c) \tag{8}$$

where $\approx_\lambda$ indicates that we approximate the solution of the optimization problem with an iterative algorithm that may also be parameterized by λ, as the exact minimizer is not available in closed form. Moreover, the notation $\boldsymbol{\theta}(\lambda)$ indicates the dependence of the lower-level solution on the upper-level parameters. In our case, we solve the lower-level problem with DMD, where we also parameterize the shift model, $\Phi_\lambda(\cdot; c)$, making it a learnable component and conditioned on c.

The normal shift model in MPC simply shifts the control sequence forward one time step and appends a zero or random control at the end. However, because we are performing this update in the latent space, there is no clear delineation between time steps of the latent controls, as they are coupled according to the learned transformation. Therefore, there is no way to easily perform the equivalent shift operation in the latent space. As such, we instead learn this shift model along with the transformation. Besides, the standard approach described above may not be optimal. By learning it, we may be able to further improve performance. This is especially true because the performance hinges greatly on the quality of the shift model since we only run one iteration of the DMD update.

3.3 Parameterizing with Normalizing Flows

In order to optimize the upper-level objective in Equation (8) with respect to λ, we need to be able to compute the density $\pi_{\theta,\lambda}$ directly. Therefore, we choose to represent h_λ with a normalizing flow (NF) [16, 18, 17], which explicitly learns the density by defining an invertible transformation that maps latent variables to observed data. Generally, we compose a series of component flows together, i.e. $h_\lambda = h_{\lambda_K} \circ h_{\lambda_{K-1}} \circ \cdots \circ h_{\lambda_1}$, which define a series of intermediate variables $\hat{\boldsymbol{y}}_0, \ldots, \hat{\boldsymbol{y}}_{K-1}, \hat{\boldsymbol{y}}_K$, with $\hat{\boldsymbol{y}}_0 = \hat{\boldsymbol{z}}$ and $\hat{\boldsymbol{y}}_K = \hat{\boldsymbol{u}}$. The log-likelihood of the composed flow is given by:

$$\log \pi_{\theta,\lambda}(\hat{\boldsymbol{u}}|c) = \log p_\theta(\hat{\boldsymbol{z}}) - \sum_{i=1}^{K} \log \left|\det \frac{\partial \hat{\boldsymbol{y}}_i}{\partial \hat{\boldsymbol{y}}_{i-1}}\right|. \tag{9}$$

In this work, we make use of the affine coupling layer proposed by Dinh et al. [18] as part of the real non-volume-preserving (RealNVP) flow. The core idea is to split the input $\hat{\boldsymbol{u}}$ into two partitions $\hat{\boldsymbol{u}} = (\hat{\boldsymbol{u}}_{I_1}, \hat{\boldsymbol{u}}_{I_2})$, where I_1 and I_2 are a partition of $[1, MH]$, and apply

$$\hat{\boldsymbol{y}}_{I_1} = \hat{\boldsymbol{u}}_{I_1}, \quad \hat{\boldsymbol{y}}_{I_2} = \hat{\boldsymbol{u}}_{I_2} \odot \exp s_\lambda(\hat{\boldsymbol{u}}_{I_1}, c) + t_\lambda(\hat{\boldsymbol{u}}_{I_1}, c), \tag{10}$$

where s_λ and t_λ are the scale and translation terms, which are represented with arbitrary neural networks, and $\odot$ is the Hadamard product. This makes computing the log-determinant term in Equation (9) and inverting the flow fast and efficient. Now, in robotics, we often have lower and upper limits on the controls. These are usually enforced in sampling-based MPC by either clamping the control samples or passing them through a scaled sigmoid. However, instead of enforcing the constraints heuristically after sampling, we learn a constrained sampling distribution directly. Since the sigmoid function is invertible and has a tractable log-determinant (shown in Appendix A.7), we can simply append one after h_{λ_K} in the NF and scale it by the control limits. This ensures that control constraints are satisfied by design and taken into account while learning the distribution.

3.4 Training the Sampling Distribution

Computing gradients through the upper-level objective is not straightforward, as both the expectation and the inner terms of Equation (7) depend on λ. Therefore, the state distribution depends on the NF and latent shift model. One way around this issue is to consider a modified objective at each batch d:

$$\ell_d(\boldsymbol{\theta}, \lambda; c) = \mathbb{E}_{\boldsymbol{\pi}_{\boldsymbol{\theta},\lambda_d},\rho,f}\left[\sum_{t=0}^{T-1} \hat{J}(\theta_t, \lambda; x_t, c)\right], \tag{11}$$

which fixes the outer expectation to be with respect to the current policy. Intuitively, this choice trains the NF to optimize the MPC cost function under the state distribution resulting from the current policy $\boldsymbol{\pi}_{\boldsymbol{\theta},\lambda_d}$. We then update the outer expectation distribution at each episode, overcoming the covariate shift problem that would otherwise arise.

Now, we only have to focus on computing the gradient $\nabla_\lambda \hat{J}(\theta_t(\lambda), \lambda; x_t, c)|_{\lambda=\lambda_d}$ for each time step, which can be computed similar to Equation (3) and approximated with Monte Carlo sampling:

$$\nabla \hat{J}(\theta_t(\lambda), \lambda; x_t, c) \approx -\sum_{i=1}^{N} w_i \nabla_\lambda \log \pi_{\theta_t(\lambda),\lambda}(\hat{\boldsymbol{u}}_t^{(i)}|c), \tag{12}$$

where the weights w_i are defined according to Equation (5). The log-likelihood is given by Equation (9), the gradient of which involves computing the backwards pass through the network h_λ. However, we also have to consider the dependence of the latent distribution parameters $\boldsymbol{\theta}(\lambda)$ on λ. Therefore, we must backpropagate through the MPC update:

$$\boldsymbol{\mu}_t(\lambda) = (1-\gamma_t^\mu)\tilde{\boldsymbol{\mu}}_t(\lambda) + \gamma_t^\mu \Delta\boldsymbol{\mu}_t, \quad \Delta\boldsymbol{\mu}_t = \frac{\mathbb{E}_{\pi_{\tilde{\theta}_t(\lambda),\lambda},\hat{f}}\left[e^{-\frac{1}{\beta}C(\hat{\boldsymbol{x}}_t,\hat{\boldsymbol{u}}_t)} h_\lambda(\hat{\boldsymbol{u}}_t; c)\right]}{\mathbb{E}_{\pi_{\tilde{\theta}_t(\lambda),\lambda},\hat{f}}\left[e^{-\frac{1}{\beta}C(\hat{\boldsymbol{x}}_t,\hat{\boldsymbol{u}}_t)}\right]} \tag{13}$$

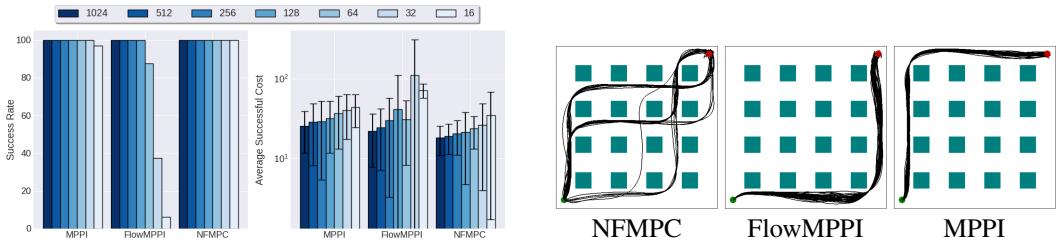
where the previous shifted mean $\tilde{\boldsymbol{\mu}}_t(\lambda)$ is given by the learned latent shift model. To compute the gradient of Equation (13), we must approximate the gradient of $\Delta\boldsymbol{\mu}_t$ with respect to λ, which we can compute as $\frac{\partial \Delta\boldsymbol{\mu}_t}{\partial \lambda} \approx M_1 - M_2 M_3$, where we define:

$$\begin{aligned} M_1 &= \sum_{i=1}^{N} w_i \Big[\nabla_\lambda h_\lambda(\hat{\boldsymbol{u}}_t^{(i)}; c) + h_\lambda(\hat{\boldsymbol{u}}_t^{(i)}; c)\nabla_\lambda \log \pi_{\tilde{\theta}(\lambda),\lambda}(\hat{\boldsymbol{u}}_t^{(i)}|c)\Big], \\ M_2 &= \sum_{i=1}^{N} w_i h_\lambda(\hat{\boldsymbol{u}}_t^{(i)}; c), \quad M_3 = \sum_{i=1}^{N} w_i \nabla_\lambda \log \pi_{\tilde{\theta}(\lambda),\lambda}(\hat{\boldsymbol{u}}_t^{(i)}|c). \end{aligned} \tag{14}$$

The derivation of this approximate gradient can be found in Appendix A.6. Note that computing the gradients $\nabla_\lambda \log \pi_{\tilde{\theta}(\lambda),\lambda}(\hat{\boldsymbol{u}}_t^{(i)}|c)$ will also require us to backpropagate through the shift model due to the dependence of $\tilde{\theta}$ on λ. Therefore, even when the step size is set to one, i.e. $\gamma_t^\mu = 1$, we introduce a form of recurrence between time steps. Please see Appendix A.1 for additional visualizations of the computational graph generated by an episode and further descriptions of the overall algorithm.

4 Related Work

Multiple works have considered sampling distributions beyond simple Gaussians, such as Gaussian mixture models [10] and a particle method based on Stein variational gradient descent (SVGD) [11]. Additionally, most implementations use heuristics to modify samples and squeeze out additional performance gains [4, 21]. In terms of learning the distribution, Amos and Yarats [12] learn a latent action space for their proposed differentiable cross-entropy method (CEM) controller. However, they require all components of the pipeline to be differentiable and do not consider learning a shift model. Agarwal et al. [13] learn a normalizing flow in the latent space of a variational autoencoder (VAE). Yet, they also require differentiability, use expert demonstrations to learn the latent space of the VAE, and have no means of warm-starting between time steps. Wang and Ba [22] propose to use a learned feedback policy to warm-start MPC, but still rely on a Gaussian perturbations to the



(a) Quantitative Results

(b) Trajectory Visualizations

Figure 1: (a) Success rate and average cost of successful trajectories on the PNGRID task across different samples. (b) Visualization of trajectories in the PNGRID task across multiple random seeds for a fixed environmental layout.

proposed action sequence. The authors also explore performing online planning in the space of the network's parameters, which results in a massive action space that may be hard to scale.

Power and Berenson [14, 15] also train a normalizing flow to use as the sampling distribution for MPC, but they do not learn a latent shift model and perform all operations in the control space. Moreover, they mix the latent samples with Gaussian perturbations to the current control-space mean, as they do not update the latent distribution directly. This prevents them from fully taking advantage of the learned distribution and throws away useful information which could potentially improve performance. In fact, we show in Appendix A.4 that the learned shift model contributes significantly to the performance gains. Additionally, a primary focus of their work is on how to handle out-of-distribution (OOD) environments by learning a posterior over environment context variables. We could combine their approach with ours by conditioning the learned shift model on the inferred environment context for improved generalization. Finally, normalizing flows have also been used to improve exploration in RL [23, 24, 25] by providing a more flexible, and potentially multimodal, distribution. They have also been employed in sampling-based motion planning [26, 27] to provide good proposal configurations to speed up convergence.

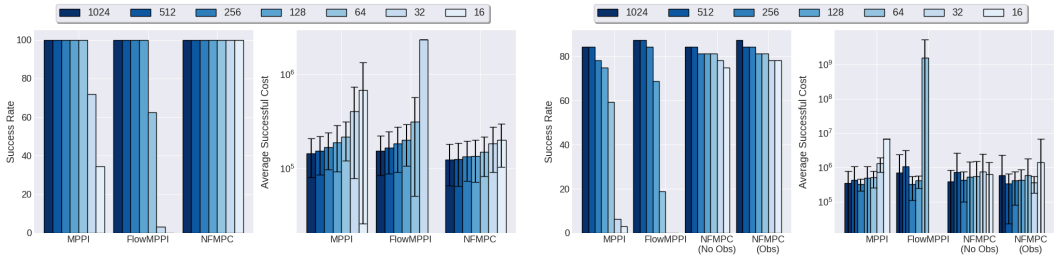
5 Experimental Results

In all experiments, we denote our proposed approach as **NFMPC**, the baseline MPPI implementation as **MPPI**, and the method by Power and Berenson [14, 15] as **FlowMPPI**. Details about the hyper-parameters, implementation, tasks, and training can be found in Appendix A.2. We evaluate on a fixed set of environments, which includes start states, goal locations, and obstacle placements, and run 32 rollouts for each sample amount. Our primary metrics for comparison are the success rate, defined as the percentage of times the task goal was achieved, and the average cost of trajectories which successfully completed the task.

5.1 Planar Robot Navigation

We begin by applying **NFMPC** to a planar robot navigation task in which a 2D holonomic point-robot must reach a goal position while avoiding obstacles, which are arranged in a grid (PNGRID). The point-robot has double integrator dynamics with stochasticity on the acceleration commands to create a mismatch between the predictive model used by MPC and the true environment. The starting and goal locations of the robot are randomized in each episode, which lasts for 200 time steps. An episode is considered successful if the agent reaches the goal without colliding into any obstacles. We wish to explore whether **NFMPC** can surpass the performance of **MPPI** and **FlowMPPI** when given access to the same number of samples. Finally, we would like to evaluate how the performance of the learned controllers scales with number of samples.

We quantitatively compare all controllers in Figure 1a and find that **NFMPC** consistently matches or outperforms both **MPPI** and **FlowMPPI** at each sample quantity in terms of both success rate and average trajectory cost of successful trajectories. At 1024 samples, **NFMPC** achieves a 29% and 17% reduction in cost over **MPPI** and **FlowMPPI**, respectively. We also find that **NFMPC** scales more gracefully than **MPPI** as the number of samples is reduced. For instance, **NFMPC** is able



(a) FRANKA Task

(b) FRANKAOBSTACLES Task

Figure 2: Success rate and average cost of successful trajectories on the (a) FRANKA and (b) FRANKAOBSTACLES environments across different samples.

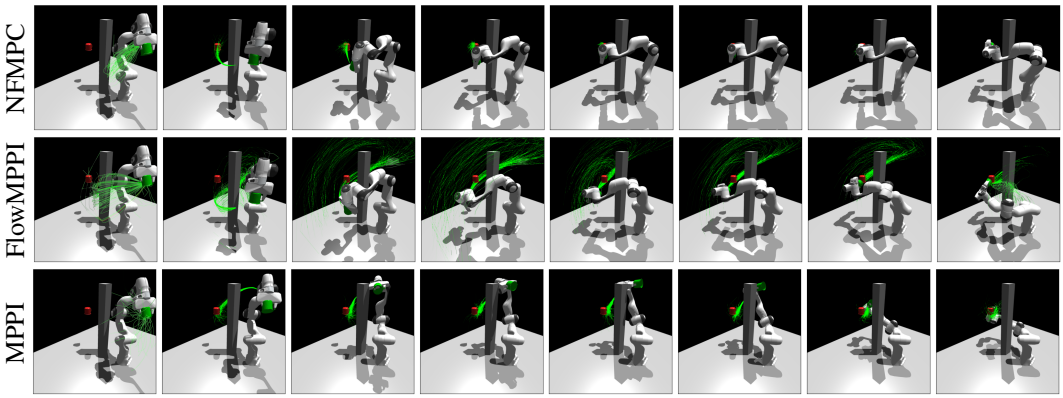


Figure 3: Visualization of a trajectory and top samples from (top) **NFMPC**, (middle) **FlowMPPI**, and (bottom) **MPPI** on the FRANKAOBSTACLES task.

to withstand a $64\times$ decrease in the number of samples (1024 to 16) while still achieving a 100% success rate, although the average trajectory cost increases by 91%. Meanwhile, **MPPI** at 16 samples has a 97% success rate and a 26% increase in average trajectory cost over **NFMPC** with the same number of samples. We found that while **FlowMPPI** improves over **MPPI** at higher sample counts, it actually performs significantly worse with fewer samples. At 16 samples, **FlowMPPI** achieves only a 6% success rate and has over a $2\times$ worse average trajectory cost than **NFMPC**. To help understand what **NFMPC** is doing differently, we superimpose 32 different trajectories with fixed start and goal positions using each controller in Figure 1b. We find that both **MPPI** and **FlowMPPI** always select the same path through the environment. Meanwhile, **NFMPC** is able to discover different paths through the environment, allowing it to better react to the stochastic perturbations that knock it off the current plan and improve performance. Additional visualizations of the resulting trajectories and top samples drawn from the distributions for all controllers can be found in Appendix A.3.

5.2 Franka Panda Arm

Next, we apply **NFMPC** to the FRANKA task, which involves controlling a 7 degree-of-freedom (DOF) Franka Panda robot arm and steering it towards a target goal from a fixed starting pose. The goal is randomly selected at the beginning of each episode, which lasts for 600 time steps. An episode is considered successful if the end effector reaches the target position under the time constraints. We plot our quantitative results in Figure 2a and find that **NFMPC** again consistently matches or outperforms **MPPI** and **FlowMPPI** at each sample amount. At 1024 samples, **NFMPC** achieves a 16% and 20% reduction in cost over **MPPI** and **FlowMPPI**, respectively. In fact, **FlowMPPI** consistently performs worse than **MPPI**, indicates that conditioning on just goal location does not help much in this more complex scenario. As before, we find that **NFMPC** scales more gracefully with reduced samples, achieving a 100% success rate with a $64\times$ decrease in the number of samples (1024 to 16) while incurring only a 63% increase in average trajectory cost. For comparison, **MPPI**

at 16 samples has only a 34.38% success rate and increases in average trajectory cost by over $4\times$. And at 16 samples, **FlowMPPI** always fails to complete the task.

To increase the difficulty of the problem, we consider the FRANKAOBSTACLES task, which adds a single pole obstacle that the arm must avoid. The obstacle and goal positions are randomized at the beginning of each episode, and the episode is successful if the end effector reaches the target while the arm avoids collisions. In addition to our normal baselines, we also compare to a controller trained only in environments without obstacles (**NFMPC (No Obs)**). We compare all four models in Figure 2b. It is also important to note that no controller achieves a 100% success rate, as not every randomly generated environment is feasible. At 1024 samples, both **NFMPC (Obs)** and **FlowMPPI** achieve a success rate of 87.50%, compared to the 84.38% success rate of **MPPI** and **NFMPC (No Obs)**. However, **NFMPC (Obs)** achieves a 16% reduction in cost over **FlowMPPI**, illustrating its potential benefits over all the baselines. We again find that both variants of **NFMPC** scale better with reduced number of samples. In fact, **FlowMPPI** scales even worse than **MPPI**, as it fails to succeed in every test environment with 32 or less samples. Meanwhile, with a $64\times$ decrease in the number of samples (1024 to 16), the success rate of **NFMPC (Obs)** only drops by 11% to 78.12%. Next, we note that **NFMPC (No Obs)** at 1024 or 512 samples actually performs worse than **MPPI** when obstacles are present. This illustrates that the gain in performance is *specific* to the environments on which the controller is trained. However, this appears to only be true with more samples, as **NFMPC (No Obs)** still achieves a 75.00% success rate at 16 samples, which is only 4% worse than **NFMPC (Obs)**. Finally, in Appendix A.4, we present a breakdown of the individual cost terms for all models to gain insight into the learned sampling distributions. We also perform an ablation which removes the learned shift model from **NFMPC** to illustrate that it is a crucial component.

6 Limitations

A major limitation of our approach is that the learned distribution and shift model are only valid for a fixed horizon length and control dimensionality. Therefore, these components cannot be directly transferred to new robots or for alternate choices of horizon without being retrained. However, this is, in general, true of all prior work which learns the sampling distribution for MPC. This could potentially be remedied in future work by novel architectural innovations and training distributions across both environments and robots. Moreover, the learned sampling distribution is specific to the environmental distributions on which it was trained. As such, we find that it can perform worse than MPPI when transferred to out-of-distribution environments. Prior work by Power and Berenson [14, 15] remedied this by learning a generative model of environments and using this learned distribution to perform a projection step on the conditional normalizing flow. However, we found in our evaluations that conditioning on environmental information consistently hurt performance. Therefore, we cannot directly apply this technique to our method, and future work needs to be done to improve generalization capabilities. Finally, while our approach does perform better overall on in-distribution environments, figuring out how to successfully incorporate environment information into the NF may possibly open up the door to further improvements.

7 Conclusion

In this paper, we presented a method for learning sampling distributions for MPC with normalizing flows (NFs), which moves all online parameter updates and warm-starting operations into the latent space. We show how to train these sampling distributions by framing the problem as bi-level optimization and deriving an approximate gradient through the MPC update. Additionally, we illustrate how to incorporate box control constraints directly into the NF architecture. Through our empirical evaluations in both simulated navigation and manipulation problems, we demonstrate that our approach is able to surpass the performance of all baselines. Moreover, we find that controllers which move all operations into the latent space are able to scale more gracefully with a reduction in the number of samples. These results indicate the importance of leveraging the latent space in learned sampling distributions for MPC. Finally, because we learn all components of the controller through episodic interactions with the environment, they can potentially be trained to account for the modeling errors in the MPC controller.

References

[1] P. Abbeel, A. Coates, and A. Y. Ng. Autonomous Helicopter Aerobatics through Apprenticeship Learning. *The International Journal of Robotics Research (IJRR)*, 29(13):1608–1639, 2010.

[2] V. Kumar, E. Todorov, and S. Levine. Optimal Control with Learned Local Models: Application to Dexterous Manipulation. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 378–383. IEEE, 2016.

[3] C. Finn and S. Levine. Deep Visual Foresight for Planning Robot Motion. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 2786–2793. IEEE, 2017.

[4] M. Bhardwaj, B. Sundaralingam, A. Mousavian, N. Ratliff, D. Fox, F. Ramos, and B. Boots. STORM: An Integrated Framework for Fast Joint-Space Model-Predictive Control for Reactive Manipulation. *arXiv preprint arXiv:2104.13542*, 2021.

[5] T. Erez, K. Lowrey, Y. Tassa, V. Kumar, S. Kolev, and E. Todorov. An integrated system for real-time Model Predictive Control of humanoid robots. In *IEEE-RAS International Conference on Humanoid Robots (Humanoids)*, pages 292–299. IEEE, 2013.

[6] Z. Erickson, H. M. Clever, G. Turk, C. K. Liu, and C. C. Kemp. Deep Haptic Model Predictive Control for Robot-Assisted Dressing. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 4437–4444. IEEE, 2018.

[7] G. Williams, P. Drews, B. Goldfain, J. M. Rehg, and E. A. Theodorou. Aggressive driving with model predictive path integral control. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 1433–1440. IEEE, 2016.

[8] G. Williams, N. Wagener, B. Goldfain, P. Drews, J. M. Rehg, B. Boots, and E. A. Theodorou. Information theoretic MPC for model-based reinforcement learning. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 1714–1721. IEEE, 2017.

[9] N. Wagener, C.-A. Cheng, J. Sacks, and B. Boots. An Online Learning Approach to Model Predictive Control. *arXiv preprint arXiv:1902.08967*, 2019.

[10] M. Okada and T. Taniguchi. Variational Inference MPC for Bayesian Model-based Reinforcement Learning. In *Conference on Robot Learning (CoRL)*, pages 258–272. PMLR, 2020.

[11] A. Lambert, A. Fishman, D. Fox, B. Boots, and F. Ramos. Stein Variational Model Predictive Control. *arXiv preprint arXiv:2011.07641*, 2020.

[12] B. Amos and D. Yarats. The Differentiable Cross-Entropy Method. In *International Conference on Machine Learning (ICML)*, pages 291–302. PMLR, 2020.

[13] S. Agarwal, H. Sikchi, C. Gulino, and E. Wilkinson. Imitative Planning using Conditional Normalizing Flow. *arXiv preprint arXiv:2007.16162*, 2020.

[14] T. Power and D. Berenson. Variational Inference MPC for Robot Motion with Normalizing Flows. *Advances in Neural Information Processing Systems (NeurIPS) Workshop on Robot Learning: Self-Supervised and Lifelong Learning*, 2021.

[15] T. Power and D. Berenson. Variational Inference MPC using Normalizing Flows and Out-of-Distribution Projection. *arXiv preprint arXiv:2205.04667*, 2022.

[16] L. Dinh, D. Krueger, and Y. Bengio. Nice: Non-linear Independent Components Estimation. *arXiv preprint arXiv:1410.8516*, 2014.

[17] D. Rezende and S. Mohamed. Variational Inference with Normalizing Flows. In *International Conference on Machine Learning (ICML)*, pages 1530–1538. PMLR, 2015.

[18] L. Dinh, J. Sohl-Dickstein, and S. Bengio. Density estimation using Real NVP. *arXiv preprint arXiv:1605.08803*, 2016.

[19] J. F. Bard. *Practical Bilevel Optimization: Algorithms and Applications*, volume 30. Springer Science & Business Media, 2013.

[20] E. Hall and R. Willett. Dynamical Models and Tracking Regret in Online Convex Programming. In *International Conference on Machine Learning (ICML)*, pages 579–587. PMLR, 2013.

[21] C. Pinneri, S. Sawant, S. Blaes, J. Achterhold, J. Stueckler, M. Rolinek, and G. Martius. Sample-efficient Cross-Entropy Method for Real-time Planning. *arXiv preprint arXiv:2008.06389*, 2020.

[22] T. Wang and J. Ba. Exploring Model-Based Planning with Policy Networks. *arXiv preprint arXiv:1906.08649*, 2019.

[23] Y. Tang and S. Agrawal. Boosting Trust Region Policy Optimization by Normalizing Flows Policy. *arXiv preprint arXiv:1809.10326*, 2018.

[24] Y. Tang and S. Agrawal. Implicit Policy for Reinforcement Learning. *arXiv preprint arXiv:1806.06798*, 2018.

[25] B. Mazoure, T. Doan, A. Durand, J. Pineau, and R. D. Hjelm. Leveraging exploration in off-policy algorithms via normalizing flows. In *Conference on Robot Learning (CoRL)*, pages 430–444. PMLR, 2020.

[26] T. Lai and F. Ramos. Learning to Plan Optimally with Flow-based Motion Planner. *arXiv preprint arXiv:2010.11323*, 2020.

[27] T. Lai, W. Zhi, T. Hermans, and F. Ramos. Parallelised Diffeomorphic Sampling-based Motion Planning. *arXiv preprint arXiv:2108.11775*, 2021.

[28] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, et al. PyTorch: An Imperative Style, High-Performance Deep Learning Library. *Advances in Neural Information Processing Systems (NeurIPS)*, 32:8026–8037, 2019.

[29] J. H. Halton. Algorithm 247: Radical-inverse quasi-random point sequence. *Communications of the ACM*, 7(12):701–702, 1964.

[30] J. L. Ba, J. R. Kiros, and G. E. Hinton. Layer Normalization. *arXiv preprint arXiv:1607.06450*, 2016.

[31] D. P. Kingma and J. Ba. Adam: A Method for Stochastic Optimization. *arXiv preprint arXiv:1412.6980*, 2014.