2024 IEEE International Conference on Robotics and Automation (ICRA 2024)
May 13-17, 2024. Yokohama, Japan

Deep Model Predictive Optimization

Jacob Sacks*, Rwik Rana*, Kevin Huang*, Alex Spitzer*, Guanya Shi†, and Byron Boots*

***Abstract*— A major challenge in robotics is to design robust policies which enable complex and agile behaviors in the real world. On one end of the spectrum, we have model-free reinforcement learning (MFRL), which is incredibly flexible and general but often results in brittle policies. In contrast, model predictive control (MPC) continually re-plans at each time step to remain robust to perturbations and model inaccuracies. However, despite its real-world successes, MPC often under-performs the optimal strategy. This is due to model quality, myopic behavior from short planning horizons, and approximations due to computational constraints. And even with a perfect model and enough compute, MPC can get stuck in bad local optima, depending heavily on the quality of the optimization algorithm. To this end, we propose Deep Model Predictive Optimization (DMPO), which learns the inner-loop of an MPC optimization algorithm directly via experience, specifically tailored to the needs of the control problem. We evaluate DMPO on a real quadrotor agile trajectory tracking task, on which it improves performance over a baseline MPC algorithm for a given computational budget. It can outperform the best MPC algorithm by up to 27% with fewer samples and an end-to-end policy trained with MFRL by 19%. Moreover, because DMPO requires fewer samples, it can also achieve these benefits with $4.3\times$ less memory. When we subject the quadrotor to turbulent wind fields with an attached drag plate, DMPO can adapt zero-shot while still outperforming all baselines. Additional results can be found at https://tinyurl.com/mr2ywmnw.**

I. Introduction

For robots to perform complex and agile behaviors in the real world, it is crucial to design control policies that remain robust while pushing the limits of the system. Model-free reinforcement learning (MFRL) is a general approach that makes minimal assumptions on the problem and has been successfully deployed in the real world [1]–[3]. However, policies trained with these methods are often brittle and do not generalize to out-of-distribution disturbances. For instance, consider an uninhabited aerial vehicle (UAV) following an aggressive trajectory in uncertain environments [4, 5]. If the UAV encounters unknown wind gusts that were not experienced in training, the policy will likely not be able to account for the change in dynamics and lead to a crash. Furthermore, due to the sample inefficiency of MFRL methods, they often train policies in simulation. This can create a sim-to-real gap due to the mismatch between the simulator and the true system. Even if the policies succeed in simulation, they often fail in the real world. We can partially remedy this issue using domain randomization (DR) [6]–[8]. However, its efficacy is dependant on the parameter distributions we select and the nature of the problem.

*University of Washington, Seattle WA 98105, USA. †Robotics Institute, Carnegie Mellon University, Pittsburgh PA 15213, USA. Email: jsacks6@cs.washington.edu

Alternatively, model predictive control (MPC) is a powerful framework which leverages a model to iteratively re-plan a finite-horizon control sequence at each time step [9]–[11]. The first control from this plan is applied to the system and the process repeats. Although the planned sequence is often open-loop, because we are updating it using the current state, MPC effectively yields a state-feedback policy. By re-solving this optimization problem online, we can improve the robustness of MPC to perturbations and model inaccuracies. However, this also leads to increased computational demands compared to MFRL. In practice, we approximate the solution at each time step to run in real time. This involves warm-starting with the solution from the previous time step, which works well when the two problems are similar [12]. But if our system encounters a large perturbation, this warm-starting procedure can bias us towards a poor solution [13]. The performance of MPC also depends on the model quality and prediction horizon length [11, 14]–[16]. And even with a perfect model and enough computation, the optimization algorithm may get stuck in bad local optima [17]. Altogether, these issues often lead to MPC under-performing the optimal policy.

To improve the performance of MPC while retaining its robustness, we propose Deep Model Predictive Optimization (DMPO), which *learns* an optimizer and warm-starting procedure directly via experience. That is, DMPO learns how to perform model-based planning more effectively while considering the computational demands for real-time deployment. Our key contributions are:

1) We develop DMPO, a general approach for learning the inner-loop of the MPC optimizer and warm-starting procedure via reinforcement learning by viewing MPC as a structured recurrent policy class;
2) On a real quadrotor platform (Crazyflie 2.1 with upgraded motors) tracking infeasible zig-zag trajectories, we show that DMPO can outperform an end-to-end (E2E) policy trained with MFRL by 19%;
3) Tracking zig-zag trajectories with alternating 180^o flips in the desired yaw, DMPO can improve error over a baseline MPC algorithm by up to 27% with $16\times$ fewer samples, saving $4.3\times$ memory requirements;
4) By exposing the quadrotor to turbulent wind fields with an attached cardboard drag plate, we show that DMPO can adapt zero-shot, matching the performance of the MPC baseline and outperforming E2E by 14%.

II. Preliminaries

A. Reinforcement Learning

We consider controlling a discrete-time stochastic dynamical system as an infinite-horizon discounted Markov decision

process (MDP) defined by the tuple $\mathcal{M} = (\mathcal{X}, \mathcal{U}, P, r, \rho_0, \gamma)$, where $\mathcal{X}$ is the state space, $\mathcal{U}$ is the control space, $x_{t+1} \sim P(\cdot|x_t, u_t)$ is the transition dynamics, $r(x, u)$ is the reward function, $\rho_0(x_0)$ is the initial state distribution, and $\gamma \in (0, 1)$ is the discount factor. Given a closed-loop policy, $u \sim \pi(\cdot|x)$, its value function is defined as

$$V^{\pi}(x) = \mathbb{E}_{P,\pi}\left[\sum_{t=0}^{\infty} \gamma^t r(x_t, u_t) \Big| x_0 = x\right]. \tag{1}$$

The goal of reinforcement learning (RL) is to find a policy which maximizes the expected discounted reward, which is equivalent to maximizing the value function:

$$\pi^* = \arg\max_{\pi} \mathbb{E}_{x_0 \sim \rho_0}\Big[V^{\pi}(x_0)\Big]. \tag{2}$$

A common approach is to directly find π with policy gradient methods, which perform gradient descent with a zeroth-order approximation of the gradient using a finite number of samples. State-of-the-art approaches include actor-critic algorithms [18, 19], which in addition to learning π (actor), learn an estimate of V_π (critic) and use it as a baseline to reduce the gradient estimator variance. If we wish to train our policy over a range of tasks, we can additionally condition both π and V_π on task parameters, such as a goal state.

B. Sampling-based Model Predictive Control

Rather than find a single, globally optimal policy, MPC re-optimizes a local policy at each time step. It accomplishes this by predicting the system's behavior over a finite horizon H using an approximate model $\hat{P}$. In sampling-based MPC, this local policy is often a distribution over open-loop control sequences, $\hat{\boldsymbol{u}}_t \sim \boldsymbol{\pi}_{\boldsymbol{\theta}_t}(\cdot)$, where $\hat{\boldsymbol{u}}_t \triangleq (\hat{u}_t, \hat{u}_{t+1}, \ldots, \hat{u}_{t+H-1})$ and $\boldsymbol{\theta}_t \in \Theta$ are some set of feasible parameters. At each time step, we solve $\boldsymbol{\theta}_t \leftarrow \arg\min_{\boldsymbol{\theta} \in \Theta} \hat{J}(\boldsymbol{\theta}; x_t)$, where $\hat{J}(\boldsymbol{\theta}; x_t)$ is a statistic defined on $C(\hat{\boldsymbol{x}}_t, \hat{\boldsymbol{u}}_t)$, the total cost of our predicted trajectory over the finite horizon H. After finding the solution, we sample a control sequence from our new policy and apply the first control, $u_t = \hat{u}_t$. Despite the plan being open-loop, the MPC procedure can be thought of as outlining a state-feedback policy. This is because we are updating the open-loop sequence using information about the current state.

A popular sampling-based approach to MPC is Model Predictive Path Integral (`MPPI`) control [10, 20], which assumes that our policy is a factorized Gaussian of the form

$$\boldsymbol{\pi}_{\boldsymbol{\theta}}(\hat{\boldsymbol{u}}) = \prod_{h=0}^{H-1} \pi_{\theta_h}(\hat{u}_{t+h}) = \prod_{h=0}^{H-1} \mathcal{N}(\hat{u}_{t+h}; \mu_{t+h}, \Sigma_{t+h}). \tag{3}$$

MPPI also assumes that we optimize the exponential utility of our cost function, defined as:

$$\hat{J}(\boldsymbol{\theta}; x_t) = -\log \mathbb{E}_{\pi_{\boldsymbol{\theta}}, \hat{P}}\Big[\exp\Big(-\frac{1}{\beta} C(\hat{\boldsymbol{x}}_t, \hat{\boldsymbol{u}}_t)\Big)\Big], \tag{4}$$

where $\beta > 0$ is a scaling parameter, known as the temperature. We can approximate the gradient of this objective with samples and compute an update via dynamic mirror descent (DMD) [12] to arrive at the `MPPI` update rule:

$$\mu_{t+h} = (1 - \gamma_t^{\mu})\tilde{\mu}_{t+h} + \gamma_t^{\mu} \sum_{i=1}^{N} w_i \hat{u}_{t+h}^{(i)}, \tag{5}$$

$$\Sigma_{t+h} = (1 - \gamma_t^{\sigma})\tilde{\Sigma}_{t+h} + \gamma_t^{\sigma} \sum_{i=1}^{N} w_i m_{t+h}^{(i)} m_{t+h}^{(i)^T}, \tag{6}$$

where $\tilde{\mu}_{t+h}$ and $\tilde{\Sigma}_{t+h}$ are the current mean and covariance matrix for each time step, $m_{t+h} = u_{t+h} - \mu_{t+h}$, γ_t^{μ} and γ_t^{σ} are the corresponding step sizes, and the weights are:

$$w_i = \frac{e^{-\frac{1}{\beta} C(\hat{\boldsymbol{x}}_t^{(i)}, \hat{\boldsymbol{u}}_t^{(i)})}}{\sum_{j=1}^{N} e^{-\frac{1}{\beta} C(\hat{\boldsymbol{x}}_t^{(j)}, \hat{\boldsymbol{u}}_t^{(j)})}}. \tag{7}$$

Despite its robustness, MPC often requires a large number of sampled trajectories or multiple update iterations, which is infeasible due to real-time constraints. To improve convergence, we can initialize the current parameters of the optimization problem as a function of the previous approximate solution, $\tilde{\boldsymbol{\theta}}_{t+1} = \Phi(\boldsymbol{\theta}_t)$, where Φ is called the shift model. A common choice is to shift the parameter sequence forward by one time step. For instance, due to Equation (3), $\boldsymbol{\theta}_t = (\theta_t, \theta_{t+1}, \ldots, \theta_{t+H-1})$ and $\tilde{\boldsymbol{\theta}}_{t+1} = (\theta_{t+1}, \theta_{t+2}, \ldots, \theta_{t+H-1}, \bar{\theta})$, where $\bar{\theta}$ is a hyperparameter.

C. Learned Optimization for MPC

Given these approximations, the performance of MPC algorithms depends heavily on the computational budget available. And even with sufficient resources, MPC can be sub-optimal relative to a globally optimal policy due to how we optimize our policy at each time step. The update rule in Equation (5) and Equation (6) does not optimality make use of the information from the sampled trajectories. One reason is that the update at each time step and for each control input is independent and are only coupled together via the weights from Equation (7). Moreover, taking a weighted sum of the samples is a fairly simple operation and may potentially throw away useful information. The choice of a static step size may also be limiting, and how much we update our policy may be better treated in an adaptive way.

In general, there is structure in the problem which we can leverage to construct a better update rule. Rather than hand-design the optimization algorithm, we can *learn* it via experience by parameterizing the update rule. In the context of MPC, Sacks et al. [21] proposed an update of the form

$$\boldsymbol{\theta}_t = m_{\phi}(\tilde{\boldsymbol{\theta}}_t, C_t^{(1:N)}), \tag{8}$$

where $C_t^{(i)} = C(\hat{\boldsymbol{x}}_t^{(i)}, \hat{\boldsymbol{u}}_t^{(i)})$ and ϕ are the network parameters. Unlike the `MPPI` update, Equation (8) jointly optimizes all parameters across controls and time step. For this to work, they generate samples from a standard Gaussian and keep them fixed. They then use the reparameterization trick to shift and scale these samples by the current mean and standard deviation, respectively. This makes all samples a deterministic function of our current policy parameters, which is already

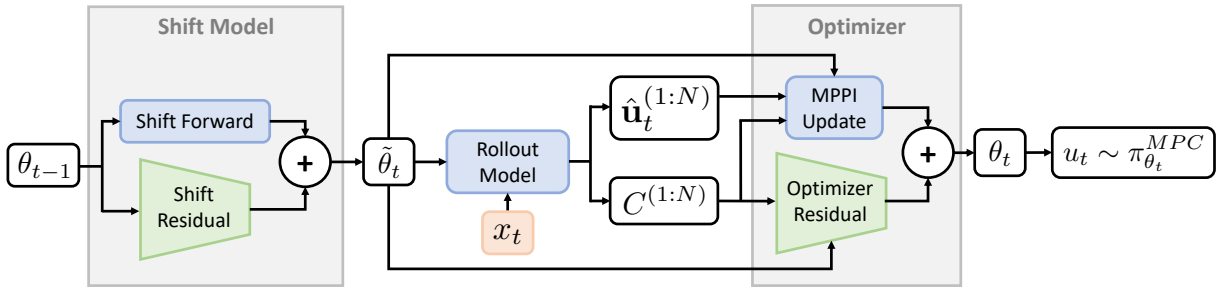

Fig. 1: The DMPO architecture consists of two learnable modules, the shift model and optimizer. The fixed rollout model module performs rollouts of the sampled control sequences.

provided to the network. This means we can remove the samples from the update altogether.

III. Deep Model Predictive Optimization

A. Problem Formulation

In DMPO, we learn an update rule of the form in Equation (8) with RL, treating MPC as a structured policy class. However, the common shift-forward operation used to warm-start the optimization only works well when problems at adjacent time steps are similar, which may not be true if there are substantial perturbations. This choice of warm-start is also independent of the value of each decision variable. Therefore, DMPO additionally learns a shift model of the form

$$\tilde{\boldsymbol{\theta}}_{t+1} = \Phi_\phi(\boldsymbol{\theta}_t). \tag{9}$$

Such a shift model is joint across time steps and control dimensions. It also provides us with a parameter-dependent shift operation, which can take advantage of structure and the types of perturbations known to effect the system.

To learn these components with RL, we need to actually consider two different policies. There is the local policy output by MPC at each time step, $\hat{\boldsymbol{u}}_t \sim \boldsymbol{\pi}_{\boldsymbol{\theta}_t}^{MPC}(\cdot)$, which we call the optimizee. Additionally, the optimizer defines a policy over the parameters of the optimizee, $\boldsymbol{\theta}_t \sim \boldsymbol{\pi}_\phi(\tilde{\boldsymbol{\theta}}_t, C_t^{(1:N)})$. From the perspective of RL, $\boldsymbol{\pi}_\phi$ is the policy we wish to learn, while $\boldsymbol{\pi}_{\boldsymbol{\theta}_t}^{MPC}$ is part of the environment dynamics. This means we have to consider both the state of the system and optimizer, forming an auxiliary MDP $\hat{\mathcal{M}} = (\mathcal{H}, \Theta, \hat{P}, r, \hat{\rho}_0, \gamma)$. We have auxiliary states $h \in \mathcal{H}$, where $h_t = (x_t, \boldsymbol{\theta}_{t-1})$, actions $\boldsymbol{\theta}_t \in \Theta$, which are the optimizer outputs, and dynamics

$$h_{t+1} = \begin{bmatrix} x_{t+1} \\ \boldsymbol{\theta}_t \end{bmatrix} \sim \hat{P}(\cdot|h_t, \boldsymbol{\theta}_t) = \begin{bmatrix} P(\cdot|x_t, u_t) \\ \delta(\boldsymbol{\theta}_t) \end{bmatrix}, \tag{10}$$

where the control of the original system, u_t, is a function of the auxiliary action, $\boldsymbol{\theta}_t$, via

$$u_t = \hat{u}_t, \quad \hat{\boldsymbol{u}}_t \sim \boldsymbol{\pi}_{\boldsymbol{\theta}_t}(\cdot). \tag{11}$$

From the perspective of the original MDP, the current parameters of the MPC policy actually form a recurrent state. Therefore, even if DMPO is parameterized with feedforward networks, we are still effectively forming a recurrent policy. The initial state distribution $\hat{\rho}_0$ now samples both an initial system state and set of optimizer parameters. And the reward function is still defined on the original state-control space, $r(x_t, u_t)$, but now u_t is a function of the optimizer action $\boldsymbol{\theta}_t$ due to Equation (11). Finally, in actor-critic algorithms, we also simultaneously learn the value function of the policy, which in this case is the optimizer, $\boldsymbol{\pi}_\phi$. This means the critic should be defined on the auxiliary state, $V^\pi(h_t)$, making it a function of both the state of the system and optimizer. Note that the optimizer never directly receives the system state x_t. While we could condition on state as well, we found that this actually hurts generalization performance, especially for sim-to-real transfer. Intuitively, there may be multiple states for which we have the same cost distribution. This means that the optimizer may encounter out-of-distribution states during testing which have in-distribution trajectory costs.

B. Algorithm Overview

A valid instantiation of DMPO is to learn both the optimizer and shift model from scratch. Instead, we learn residuals on the MPPI update and shift-forward operation. While the closed-form MPPI update may be sub-optimal, it still can be fairly robust and generalize well to different tasks. By learning each component as a residual operation, it can alter the reward landscape in a way that can simplify exploration. If the MPPI controller is already good, it allows us to potentially inherit its robustness and generalization capabilities, with the residual providing small corrections. But even if the proposed MPPI update is far from optimal, such as in the case when the controller has access to few samples, it can still provide a hint about a good direction to search. We illustrate each module of DMPO in Figure 1, which we describe below:

Shift Model. Let us define $\tilde{\boldsymbol{\theta}}_t = (\tilde{\boldsymbol{\mu}}_t, \tilde{\boldsymbol{\Sigma}}_t)$ as the shifted parameters of our optimizee policy. Then we implement the shift model, Φ_ϕ, as a residual update:

$$\begin{aligned} \hat{\boldsymbol{\mu}}_t, \hat{\boldsymbol{\Sigma}}_t &= \Phi_\phi(\boldsymbol{\theta}_t) \\ \tilde{\boldsymbol{\mu}}_t = \boldsymbol{\mu}_t^{SHIFT} + \hat{\boldsymbol{\mu}}_t, &\quad \tilde{\boldsymbol{\Sigma}}_t = \boldsymbol{\Sigma}_t^{SHIFT} \odot \hat{\boldsymbol{\Sigma}}_t, \end{aligned} \tag{12}$$

where $\boldsymbol{\mu}_t^{SHIFT}$ and $\boldsymbol{\Sigma}_t^{SHIFT}$ are the mean and covariance following the normal shift forward operation and $\odot$ is the Hadamard product. While both updates could be additive, we found that the multiplicative update for the covariance worked better in practice.

Rollout Model. We use fixed samples from a standard Gaussian, scaling and shifting them by $\tilde{\boldsymbol{\Sigma}}_t$ and $\tilde{\boldsymbol{\mu}}_t$, respectively, to get $\hat{\boldsymbol{u}}_t^{(1:N)}$. Additionally, we always include the

current mean as one of the samples. After this reparameterization, we roll out the open-loop control sequences with our dynamics model to get a set of N scalar total trajectory costs concatenated into a vector $C^{(1:N)}$. Since we also run the MPPI update, we compute weights using Equation (7). Importantly, this is the only module which makes use of the current system state x_t.

Optimizer. We compute the normal MPPI update using Equation (5) and Equation (6) to get $\boldsymbol{\theta}_t^{MPC} = (\boldsymbol{\mu}_t^{MPC}, \boldsymbol{\Sigma}_t^{MPC})$. The DMPO update for the mean is then

$$\begin{aligned} \hat{\boldsymbol{\mu}}_t, \boldsymbol{g}_t, \boldsymbol{\sigma}_t^{\mu} &= m_{\phi}^{\mu}(\tilde{\boldsymbol{\theta}}_t, C_t^{1:N}) \\ \boldsymbol{\mu}_t &= (1 - \boldsymbol{g}_t) \odot \boldsymbol{\mu}_t^{MPC} + \boldsymbol{g}_t \odot \hat{\boldsymbol{\mu}}_t, \end{aligned} \tag{13}$$

where $\boldsymbol{g}_t$ is a gating term, which has the same dimension as the mean and is passed through a sigmoid to ensure it is between zero and one. It allows the network to modulate how much it relies on MPC versus the network output. Since we are using PPO [18] to train the components of DMPO, we actually need a distribution over proposed mean and covariance updates. Therefore, the network also outputs a standard deviation $\boldsymbol{\sigma}_t^{\mu}$, which then defines our optimizer policy for the mean, $\boldsymbol{\pi}_{\phi}^{\mu} = \mathcal{N}(\boldsymbol{\mu}_t, \boldsymbol{\sigma}_t^{\mu})$. During training, we sample the mean update from this policy, but simply use the mean at test time. Similarly, for the covariance matrix:

$$\hat{\boldsymbol{\Sigma}}_t, \boldsymbol{\sigma}_t^{\Sigma} = m_{\phi}^{\Sigma}(\tilde{\boldsymbol{\theta}}_t, C_t^{1:N}), \quad \boldsymbol{\Sigma}_t = \boldsymbol{\Sigma}_t^{MPC} \odot \hat{\boldsymbol{\Sigma}}_t, \tag{14}$$

where the optimizer scales the covariance matrix proposed by MPC. Our optimizer policy for the covariance is then $\boldsymbol{\pi}_{\phi}^{\Sigma} = \mathcal{N}(\boldsymbol{\Sigma}_t, \boldsymbol{\sigma}_t^{\Sigma})$. It is also important to note that these mean and covariance updates jointly depend on all the current parameters values, $\tilde{\boldsymbol{\theta}}_t$. This means that we consider the current covariance while updating the mean, and vice versa.

IV. Related Work

Learned Optimization. There is a large body of work on L2O in the context of training neural networks [22]–[34]. Another thrust learns how to perform reinforcement learning more efficiently [35, 36, 36]–[40]. These approaches still ultimately use standard stochastic gradient descent optimizers to update the policy and value functions.

Combining MPC with Learning. Common strategies to boosting the performance of MPC involve learning a dynamics model [10, 41]–[53], terminal value functions [54]–[58], cost-shaping terms [15], the entire controller end-to-end [15, 59]–[64], or improving the sampling distribution [65]–[74]. However, these methods all leverage the structure of the optimization solver, learning components of the model or objective, rather than training a new update rule. Sacks et al. [21] explored learned optimization in the context of MPC. However, they do not learn a shift model or a residual, and their optimizer is trained with imitation learning. This limits their performance to the quality of the expert, which was an MPC controller with many samples.

Residual Policy Learning. Prior work has explored learning residual policies on a hand-designed controller with RL [75]–[79]. However, unlike DMPO, these methods do not leverage structure in the policy class. Additionally, there is nothing specific in DMPO necessitating a residual update.

MPC and RL for Quadrotor Control. Researchers have successfully deployed sampling-based [2, 80, 81] and gradient-based [82]–[85] MPC on quadrotors. A growing body of literature has applied RL for quadrotor stabilization [86]–[88], trajectory tracking [2, 89], and high-speed drone racing [3]. More closely related to our work, Romero et al. [90] embed differentiable MPC [62] into an actor-critic pipeline. Song et al. [91] use RL to tune the hyperparameters of MPC in an adaptive, state-dependent fashion. However, both treat the MPC optimizer as a fixed component, instead learning how to tweak its hyperparameters and objective.

V. Experiments

A. Task and Implementation Details.

Quadrotor Trajectory Tracking. We perform all evaluations on a quadrotor trajectory tracking problem in which the desired trajectories are infeasible zig-zags with and without yaw flips. These zig-zags linearly connect a series of random waypoints, while the yaw flips are a 180^o change to the desired yaw at each waypoint. For the real hardware experiments, we use the Bitcraze Crazyflie 2.1 equipped with the longer 20 mm motors from the thrust upgrade bundle. State estimation for position and velocity is provided by an OptiTrack motion capture system, while the Crazyflie provides orientation estimates via a 2.4 GHz radio. An offboard computer receives the state estimates and runs all controllers at a rate of 50 Hz. All controllers operate on desired body thrust f_{des} and angular velocity $\boldsymbol{\omega}_{des}$ commands. We convert these commands to motor thrusts using a low-level controller.

For training and the MPC dynamics model, we use the following dynamics with a dt $= 0.02$:

$$\dot{\boldsymbol{p}} = \boldsymbol{v}, \quad m\dot{\boldsymbol{v}} = m\boldsymbol{g} + \boldsymbol{R}\boldsymbol{e}_3 f, \quad \dot{\boldsymbol{R}} = \boldsymbol{R}S(\omega), \tag{15}$$

where $\boldsymbol{p}, \boldsymbol{v}, \boldsymbol{g} \in \mathbb{R}^3$ are the position, velocity, and gravity vectors in the world frame, $\boldsymbol{R} \in \mathrm{SO}(3)$ is the attitude rotation matrix, $\boldsymbol{\omega} \in \mathbb{R}^3$ is the angular velocity in the body frame, $S(\cdot) : \mathbb{R}^3 \to \mathrm{so}(3)$ maps a vector to its skew-symmetric matrix form, $\boldsymbol{e}_3$ is a unit vector in the Z direction, and m is the mass. The state of the system is then $\boldsymbol{x} = (\boldsymbol{p}, \boldsymbol{v}, \boldsymbol{q}, \boldsymbol{\omega})$, where $\boldsymbol{q}$ is the quaternion representation of $\boldsymbol{R}$, and $\boldsymbol{u} = (f_{des}, \boldsymbol{\omega}_{des})$. Rather than explicitly model the angular velocity dynamics and low-level controller, we convert $\boldsymbol{u}$ to the actual thrust f and angular velocity $\boldsymbol{\omega}$ using a first-order time delay model:

$$\begin{aligned} \boldsymbol{\omega}_t &= \boldsymbol{\omega}_{t-1} + k(\boldsymbol{\omega}_{des} - \boldsymbol{\omega}_{t-1}) \\ f_t &= f_{t-1} + k(f_{des} - f_{t-1}). \end{aligned} \tag{16}$$

The cost function includes terms defined on position and orientation tracking performance. Additionally, a control penalty was necessary for sim-to-real transfer. Without it, DMPO would exploit the simulator and learn aggressive commands which are difficult to perform on the real system.

Hyperparameters and Training. The optimizer, shift model, and value function were all parameterized with multi-layer perceptrons (MLPs) with a single hidden layer of 256

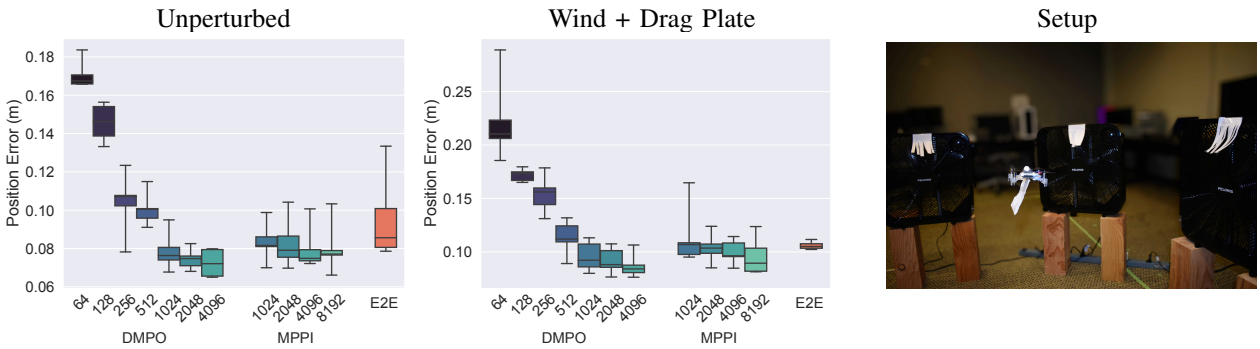

Fig. 2: Position tracking error of DMPO versus MPPI and E2E on tracking random infeasible zig-zag trajectories without any environmental disturbances (left) and with an attached plate and wind (middle), with the setup shown on the right.

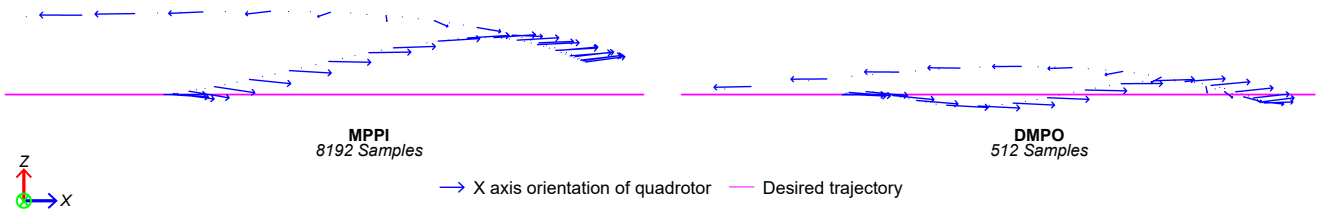

Fig. 3: Example zig-zag trajectory with a 180^o yaw flip performed by MPPI (8192 samples) and DMPO (512 samples).

and ReLU activation functions implemented in PyTorch [92]. Since the value function operates on the full auxiliary state, we needed to make it aware of the reference trajectory. Therefore, we give it the desired trajectory for the next 32 time steps with a stride of 4, forming a 56-dimensional conditioning vector. However, we note that this is not necessary for the optimizer or shift model, as they do not operate on states. For the network initialization, we set the last layer of each MLP to have a weight distribution of $\mathcal{N}(0, 0.001)$. This made each residual term effectively zero, allowing us to start with the MPPI update and shift-forward operation for warm-starting.

We trained the optimizers with PPO [18] and Generalized Advantage Estimation (GAE) [93] on an NVIDIA RTX 3080 GPU with a $\gamma = 0.99$ and $\lambda = 0.95$. To update DMPO, we used Adam [94] with a learning rate of 10^{-6} and 10^{-4} for the actor and critic, respectively. Learning the DMPO residual optimizers only took up to 1000 iterations of PPO to achieve good performance. To improve performance, we used domain randomization (DR) [6]–[8] on the mass, randomly scaling it by a factor in $[0.7\times, 1.3\times]$, and the delay coefficient k, selecting it in $[0.2, 0.6]$. We also applied a constant force perturbation with a randomized direction and magnitude at the beginning of each episode in $[-3.5\ N, 3.5\ N]$.

Baselines. Our two baselines are MPPI and an end-to-end (E2E) 3-layer MLP policy operating on states, conditioned on desired trajectories. The desired trajectories consist of the 10 desired positions up to 0.6 seconds in the future, evenly spaced in time. E2E was also trained with PPO using DR, but with a learning rate of 3×10^{-4} and took about 10^7 iterations to converge. We used a custom implementation of MPPI in PyTorch [92], which used Halton sequences [95] for generating the fixed samples from a standard Gaussian. We tune all hyperparameters of the MPPI controller using a grid search in simulation on a fixed set of desired trajectories.

B. Tracking Performance on Infeasible Zig-Zags

We begin by evaluating the performance of DMPO compared to MPPI and E2E on random zig-zag trajectories without the presence of additional disturbances. For each controller, we evaluate the performance across 5 different fixed trajectory seeds. Figure 2 (left) reports the position tracking error of each controller, varying the number of samples for DMPO and MPPI. The box plots represent the median and quartiles of the total episode costs. Given 512 or less samples, MPPI would consistently crash, while DMPO is able to successfully complete the task with as few as 64 samples. We found that increasing the number of samples for MPPI only helps to a point, after which performance can suffer. However, increasing the number of samples generally improves the median performance of DMPO. And DMPO with 1024 samples outperforms the best MPPI controller (with 4096 samples) by 7% and E2E by 19% in terms of the median error.

In order to gauge whether DMPO retains the robustness of MPC, we test the Crazyflie in a scenario with an unknown wind field generated by three fans. Additionally, we attached a soft cardboard plate hanging below the chassis, which creates drag and adds additional mass (see the right of Figure 2). The combination of the fans and cardboard plate creates highly dynamic and state-dependent disturbances which are not encountered during training. We report the results of these perturbations on the right of Figure 2. The performance of all three controllers got worse, although DMPO can still remain in the air with as few as 64 samples. DMPO with 1024 samples nearly matches the performance of the best MPPI controller with 8192 samples. And it still surpasses the performance of E2E by 14%, with the improvement growing with samples.

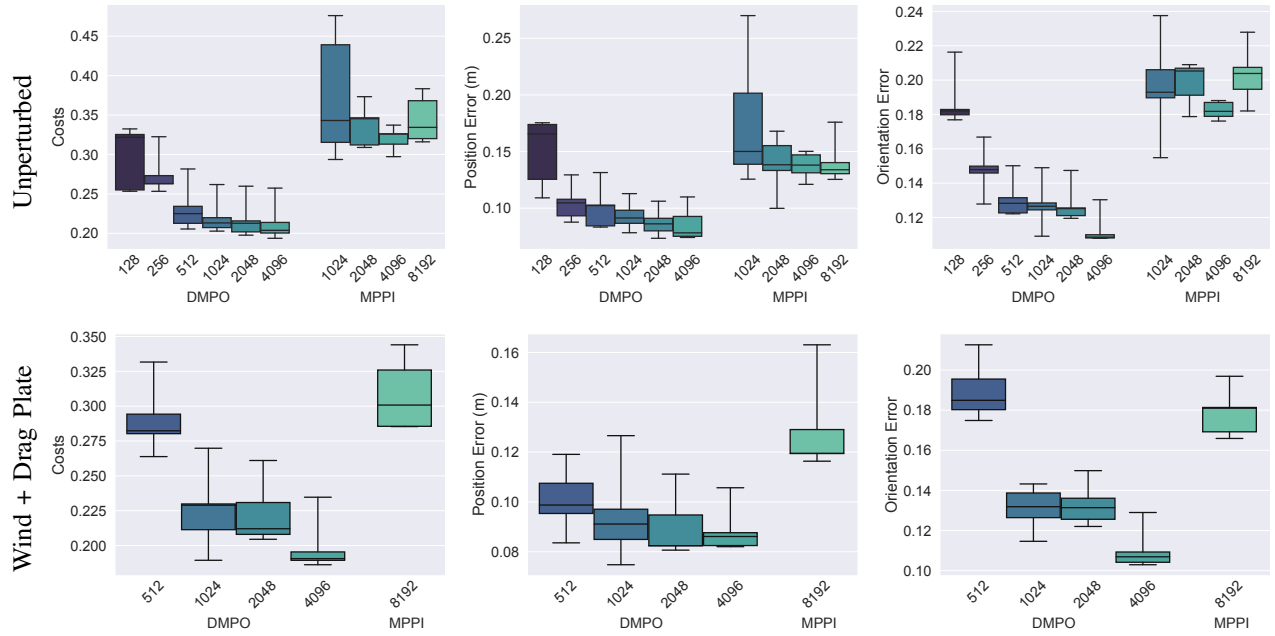

Fig. 4: Total cost (left), position error in meters (middle), and orientation error (right) of DMPO versus MPPI on tracking random yaw flip trajectories without any environmental disturbances (top) and with an attached plate and wind (bottom).

TABLE I: Relative memory usage between MPPI with 4096 samples and DMPO.

# DMPO Samples	256	512	1024	2048
Memory Reduction	4.3×	3.1×	1.9×	1.1×

C. Tracking Performance on Yaw Flips

Next, we evaluate performance on zig-zags with yaw flips. The E2E baseline could not successfully transfer to the real system on this task. The top row of Figure 4 reports the performance of DMPO and MPPI without the presence of additional disturbances. Again, with 512 samples or fewer, MPPI would consistently crash, while DMPO with as few as 128 can successfully stay in the air. At 256 samples, DMPO outperforms the best MPPI controller with 4096 samples by over 27%. And in this much harder scenario, DMPO with 4096 samples is 64% better than MPPI. Breaking down the cost, we see that DMPO with 256 or more samples does a much better job in tracking both desired position and orientation. Figure 3 illustrates an example trajectory and how DMPO has much better position tracking (especially in the Z direction) and rotates more rapidly to improving orientation tracking.

We report the perturbation results in the bottom row of Figure 4. In this scenario, MPPI needed 8192 samples to avoid crashing, while DMPO remained robust at 512 samples, outperforming MPPI by 7%. And given 4096 samples, DMPO is over 57% better than MPPI. We again see that position error is substantially lower for DMPO. In contrast, DMPO at 512 samples was slightly worse than MPPI for orientation error. Yet, DMPO quickly got better at tracking orientation with 1024 or more samples. Comparing DMPO in this case to the unperturbed scenario, we see that its performance is markedly similar. In fact, it only got worse by about 7% on average, while MPPI incurred about 11% more cost. The median position and orientation errors are only slightly larger with disturbances, except for the 512-sample DMPO. Together, these results indicate the zero-shot generalization capability of DMPO in the presence of unknown disturbances. Additional results can be found at https://tinyurl.com/mr2ywmnw.

D. Compute Requirements

DMPO with 256 samples is 1.2× faster than MPPI with 4096 samples on our offboard computer while outperforming it on the yaw flip task. And the savings may be even greater for on-board compute which is more constrained. We report the memory usage of DMPO for various samples compared to MPPI with 4096 samples in Table I. With 256 samples, DMPO requires 4.3× less memory. Altogether, this means that we can achieve better performance while using less compute and memory compared to MPPI.

VI. Conclusion

We devised DMPO, a method for jointly learning the optimizer and warm-starting procedure for MPC. By framing the optimizer as a policy in an auxiliary MDP, we showed how MPC could be treated as a structured policy class and learned via MFRL. We evaluated DMPO on a real quadrotor platform tracking infeasible zig-zag trajectories and showed it can outperform E2E and MPPI controllers with far fewer samples. And DMPO is even more robust than MPPI to unseen disturbances, such as unknown wind fields and an attached cardboard drag plate. Moreover, since DMPO can accomplish this level of performance with fewer samples, it can save up to 4.3× memory and reduce runtime by 1.2× compared to MPPI. This indicates DMPO is a viable strategy to leverage the robustness of MPC while improving upon these hand-designed controllers and better match the optimal policy.

REFERENCES

[1] M. A. OpenAI, B. Baker, M. Chociej, R. Józefowicz, B. McGrew, J. W. Pachocki, J. Pachocki, A. Petron, M. Plappert, G. Powell, *et al.*, "Learning Dexterous In-Hand Manipulation," *arXiv preprint arXiv:1808.00177*, 2018.

[2] K. Huang, R. Rana, G. Shi, A. Spitzer, and B. Boots, "DATT: Deep Adaptive Trajectory Tracking for Quadrotor Control," in *Conference on Robot Learning (CoRL)*, 2023.

[3] E. Kaufmann, L. Bauersfeld, A. Loquercio, M. Müller, V. Koltun, and D. Scaramuzza, "Champion-Level Drone Racing Using Deep Reinforcement Learning," *Nature*, vol. 620, no. 7976, pp. 982–987, 2023.

[4] M. O'Connell, G. Shi, X. Shi, K. Azizzadenesheli, A. Anandkumar, Y. Yue, and S.-J. Chung, "Neural-fly enables rapid learning for agile flight in strong winds," *Science Robotics*, vol. 7, no. 66, p. eabm6597, 2022.

[5] G. Shi, X. Shi, M. O'Connell, R. Yu, K. Azizzadenesheli, A. Anandkumar, Y. Yue, and S.-J. Chung, "Neural lander: Stable drone landing control using learned dynamics," in *2019 international conference on robotics and automation (icra)*. IEEE, 2019, pp. 9784–9790.

[6] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel, "Domain Randomization for Transferring Deep Neural Networks from Simulation to the Real World," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2017, pp. 23–30.

[7] X. B. Peng, M. Andrychowicz, W. Zaremba, and P. Abbeel, "Sim-to-Real Transfer of Robotic Control with Dynamics Randomization," in *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2018, pp. 3803–3810.

[8] X. Chen, J. Hu, C. Jin, L. Li, and L. Wang, "Understanding Domain Randomization for Sim-to-Real Transfer," *arXiv preprint arXiv:2110.03239*, 2021.

[9] M. Morari and J. H. Lee, "Model predictive control: past, present and future," *Computers & chemical engineering*, vol. 23, no. 4-5, pp. 667–682, 1999.

[10] 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)*. IEEE, 2017, pp. 1714–1721.

[11] C. Yu, G. Shi, S.-J. Chung, Y. Yue, and A. Wierman, "The power of predictions in online control," *Advances in Neural Information Processing Systems*, vol. 33, pp. 1994–2004, 2020.

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

[13] T. Erez, Y. Tassa, and E. Todorov, "Infinite-Horizon Model Predictive Control for Periodic Tasks with Contacts," *Robotics: Science and systems VII*, p. 73, 2012.

[14] N. Jiang, A. Kulesza, S. Singh, and R. Lewis, "The Dependence of Effective Planning Horizon on Model Accuracy," in *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems*, 2015, pp. 1181–1189.

[15] A. Tamar, G. Thomas, T. Zhang, S. Levine, and P. Abbeel, "Learning from the Hindsight Plan — Episodic MPC Improvement," in *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2017, pp. 336–343.

[16] T. Li, R. Yang, G. Qu, G. Shi, C. Yu, A. Wierman, and S. Low, "Robustness and consistency in linear quadratic control with untrusted predictions," *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, vol. 6, no. 1, pp. 1–35, 2022.

[17] A. Jain, L. Chan, D. S. Brown, and A. D. Dragan, "Optimal Cost Design for Model Predictive Control," in *Learning for Dynamics and Control*. PMLR, 2021, pp. 1205–1217.

[18] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal Policy Optimization Algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[19] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust Region Policy Optimization," in *International Conference on Machine Learning (ICML)*. PMLR, 2015, pp. 1889–1897.

[20] 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)*. IEEE, 2016, pp. 1433–1440.

[21] J. Sacks and B. Boots, "Learning to Optimize in Model Predictive Control," in *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2022, pp. 10 549–10 556.

[22] M. Andrychowicz, M. Denil, S. Gomez, M. W. Hoffman, D. Pfau, T. Schaul, B. Shillingford, and N. De Freitas, "Learning to learn by gradient descent by gradient descent," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2016, pp. 3981–3989.

[23] S. Ravi and H. Larochelle, "Optimization as a Model for Few-Shot Learning," *International Conference on Learning Representations (ICLR)*, 2016.

[24] K. Lv, S. Jiang, and J. Li, "Learning Gradient Descent: Better Generalization and Longer Horizons," in *International Conference on Machine Learning (ICML)*. PMLR, 2017, pp. 2247–2255.

[25] T. Chen, W. Zhang, Z. Jingyang, S. Chang, S. Liu, L. Amini, and Z. Wang, "Training Stronger Baselines for Learning to Optimize," *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 33, 2020.

[26] S. Flennerhag, Y. Schroecker, T. Zahavy, H. van Hasselt, D. Silver, and S. Singh, "Bootstrapped Meta-Learning," *arXiv preprint arXiv:2109.04504*, 2021.

[27] J. Yang, T. Chen, M. Zhu, F. He, D. Tao, Y. Liang, and Z. Wang, "Learning to Generalize Provably in Learning to Optimize," in *International Conference on Artificial Intelligence and Statistics*. PMLR, 2023, pp. 9807–9825.

[28] X. Chen, T. Chen, Y. Cheng, W. Chen, A. Awadallah, and Z. Wang, "Scalable Learning to Optimize: A Learned Optimizer Can Train Big Models," in *European Conference on Computer Vision*. Springer, 2022, pp. 389–405.

[29] L. Metz, C. D. Freeman, N. Maheswaranathan, and J. Sohl-Dickstein, "Training Learned Optimizers with Randomly Initialized Learned Optimizers," *arXiv preprint arXiv:2101.07367*, 2021.

[30] L. Metz, N. Maheswaranathan, C. D. Freeman, B. Poole, and J. Sohl-Dickstein, "Tasks, Stability, Architecture, and Compute: Training More Effective Learned Optimizers, and Using Them to Train Themselves," *arXiv preprint arXiv:2009.11243*, 2020.

[31] O. Wichrowska, N. Maheswaranathan, M. W. Hoffman, S. G. Colmenarejo, M. Denil, N. Freitas, and J. Sohl-Dickstein, "Learned Optimizers that Scale and Generalize," in *International Conference on Machine Learning (ICML)*. PMLR, 2017, pp. 3751–3760.

[32] L. Metz, N. Maheswaranathan, J. Nixon, D. Freeman, and J. Sohl-Dickstein, "Understanding and Correcting Pathologies in the Training of Learned Optimizers," in *International Conference on Machine Learning (ICML)*. PMLR, 2019, pp. 4556–4565.

[33] K. Li and J. Malik, "Learning to Optimize," *arXiv preprint arXiv:1606.01885*, 2016.

[34] ——, "Learning to Optimize Neural Nets," *arXiv preprint arXiv:1703.00441*, 2017.

[35] C. Lu, J. Kuba, A. Letcher, L. Metz, C. Schroeder de Witt, and J. Foerster, "Discovered Policy Optimisation," *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 35, pp. 16 455–16 468, 2022.

[36] J. Oh, M. Hessel, W. M. Czarnecki, Z. Xu, H. P. van Hasselt, S. Singh, and D. Silver, "Discovering Reinforcement Learning Algorithms," *Advances in Neural Information Processing Systems*, vol. 33, pp. 1060–1070, 2020.

[37] L. Kirsch, S. van Steenkiste, and J. Schmidhuber, "Improving Generalization in Meta Reinforcement Learning Using Learned Objectives," *arXiv preprint arXiv:1910.04098*, 2019.

[38] R. Houthooft, Y. Chen, P. Isola, B. Stadie, F. Wolski, O. Jonathan Ho, and P. Abbeel, "Evolved Policy Gradients," *Advances in Neural Information Processing Systems*, vol. 31, 2018.

[39] J. D. Co-Reyes, Y. Miao, D. Peng, E. Real, S. Levine, Q. V. Le, H. Lee, and A. Faust, "Evolving Reinforcement Learning Algorithms," *arXiv preprint arXiv:2101.03958*, 2021.

[40] J. X. Wang, Z. Kurth-Nelson, D. Tirumala, H. Soyer, J. Z. Leibo, R. Munos, C. Blundell, D. Kumaran, and M. Botvinick, "Learning to reinforcement learn," *arXiv preprint arXiv:1611.05763*, 2016.

[41] 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)*. IEEE, 2018, pp. 4437–4444.

[42] I. Lenz, R. A. Knepper, and A. Saxena, "DeepMPC: Learning Deep Latent Features for Model Predictive Control," in *Robotics: Science and Systems (R:SS)*. Rome, Italy, 2015.

[43] J. Kocijan, R. Murray-Smith, C. E. Rasmussen, and A. Girard, "Gaussian process model based predictive control," in *American Control Conference (ACC)*, vol. 3. IEEE, 2004, pp. 2214–2219.

[44] J. Fu, S. Levine, and P. Abbeel, "One-Shot Learning of Manipulation Skills with Online Dynamics Adaptation and Neural Network Priors," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2016, pp. 4019–4026.

[45] K. Chua, R. Calandra, R. McAllister, and S. Levine, "Deep Reinforcement Learning in a Handful of Trials using Probabilistic Dynamics Models," *arXiv preprint arXiv:1805.12114*, 2018.

[46] A. Nagabandi, G. Kahn, R. S. Fearing, and S. Levine, "Neural Network Dynamics for Model-Based Deep Reinforcement Learning with Model-Free Fine-Tuning," in *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2018, pp. 7559–7566.

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

[48] N. Wahlström, T. B. Schön, and M. P. Deisenroth, "From Pixels to Torques: Policy Learning with Deep Dynamical Models," *arXiv preprint arXiv:1502.02251*, 2015.

[49] M. Watter, J. T. Springenberg, J. Boedecker, and M. Riedmiller, "Embed to Control: A Locally Linear Latent Dynamics Model for Control from Raw Images," *arXiv preprint arXiv:1506.07365*, 2015.

[50] E. Banijamali, R. Shu, H. Bui, and A. Ghodsi, "Robust Locally-Linear Controllable Embedding," in *International Conference on Artificial Intelligence and Statistics (AISTATS)*. PMLR, 2018, pp. 1751–1759.

[51] F. Ebert, C. Finn, S. Dasari, A. Xie, A. Lee, and S. Levine, "Visual Foresight: Model-Based Deep Reinforcement Learning for Vision-Based Robotic Control," *arXiv preprint arXiv:1812.00568*, 2018.

[52] J.-S. Ha, Y.-J. Park, H.-J. Chae, S.-S. Park, and H.-L. Choi, "Adaptive Path-Integral Autoencoder: Representation Learning and Planning for Dynamical Systems," *Advances in Neural Information Processing Systems (NeurIPS)*, 2018.

[53] D. Hafner, T. Lillicrap, I. Fischer, R. Villegas, D. Ha, H. Lee, and J. Davidson, "Learning Latent Dynamics for Planning from Pixels," in *International Conference on Machine Learning (ICML)*. PMLR, 2019, pp. 2555–2565.

[54] M. Zhong, M. Johnson, Y. Tassa, T. Erez, and E. Todorov, "Value Function Approximation and Model Predictive Control," in *IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning (ADPRL)*. IEEE, 2013, pp. 100–107.

[55] U. Rosolia and F. Borrelli, "Learning Model Predictive Control for Iterative Tasks. A Data-Driven Control Framework," *IEEE Transactions on Automatic Control*, vol. 63, no. 7, pp. 1883–1896, 2017.

[56] K. Lowrey, A. Rajeswaran, S. Kakade, E. Todorov, and I. Mordatch, "Plan Online, Learn Offline: Efficient Learning and Exploration via Model-Based Control," *arXiv preprint arXiv:1811.01848*, 2018.

[57] M. Bhardwaj, S. Choudhury, and B. Boots, "Blending MPC & Value Function Approximation for Efficient Reinforcement Learning," *arXiv preprint arXiv:2012.05909*, 2020.

[58] M. Bhardwaj, A. Handa, D. Fox, and B. Boots, "Information Theoretic Model Predictive Q-Learning," in *Learning for Dynamics & Control (L4DC)*. PMLR, 2020, pp. 840–850.

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

[60] P. Karkus, D. Hsu, and W. S. Lee, "QMDP-Net: Deep Learning for Planning under Partial Observability," *arXiv preprint arXiv:1703.06692*, 2017.

[61] M. Okada, L. Rigazio, and T. Aoshima, "Path Integral Networks: End-to-End Differentiable Optimal Control," *arXiv preprint arXiv:1706.09597*, 2017.

[62] B. Amos, I. D. J. Rodriguez, J. Sacks, B. Boots, and J. Z. Kolter, "Differentiable MPC for End-to-end Planning and Control," *arXiv preprint arXiv:1810.13400*, 2018.

[63] M. Okada and T. Taniguchi, "Acceleration of Gradient-based Path Integral Method for Efficient Optimal and Inverse Optimal Control," in *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2018, pp. 3013–3020.

[64] M. Pereira, D. D. Fan, G. N. An, and E. Theodorou, "MPC-Inspired Neural Network Policies for Sequential Decision Making," *arXiv preprint arXiv:1802.05803*, 2018.

[65] 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.

[66] ——, "Variational Inference MPC using Normalizing Flows and Out-of-Distribution Projection," *arXiv preprint arXiv:2205.04667*, 2022.

[67] J. Sacks and B. Boots, "Learning Sampling Distributions for Model Predictive Control," in *Conference on Robot Learning (CoRL)*. PMLR, 2023, pp. 1733–1742.

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

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

[70] T. Power and D. Berenson, "Constrained Stein Variational Trajectory Optimization," *arXiv preprint arXiv:2308.12110*, 2023.

[71] D. M. Asmar, R. Senanayake, S. Manuel, and M. J. Kochenderfer, "Model Predictive Optimized Path Integral Strategies," in *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2023, pp. 3182–3188.

[72] J. Yin, Z. Zhang, E. Theodorou, and P. Tsiotras, "Trajectory Distribution Control for Model Predictive Path Integral Control Using Covariance Steering," in *2022 International Conference on Robotics and Automation (ICRA)*. IEEE, 2022, pp. 1478–1484.

[73] I. S. Mohamed, J. Xu, G. Sukhatme, and L. Liu, "Towards Efficient MPPI Trajectory Generation with Unscented Guidance: U-MPPI Control Strategy," *arXiv preprint arXiv:2306.12369*, 2023.

[74] K. Honda, N. Akai, K. Suzuki, M. Aoki, H. Hosogaya, H. Okuda, and T. Suzuki, "Stein Variational Guided Model Predictive Path Integral Control: Proposal and Experiments with Fast Maneuvering Vehicles," *arXiv preprint arXiv:2309.11040*, 2023.

[75] G. Lee, B. Hou, S. Choudhury, and S. S. Srinivasa, "Bayesian Residual Policy Optimization:: Scalable Bayesian Reinforcement Learning with Clairvoyant Experts," in *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2021, pp. 5611–5618.

[76] T. Johannink, S. Bahl, A. Nair, J. Luo, A. Kumar, M. Loskyll, J. A. Ojea, E. Solowjow, and S. Levine, "Residual Reinforcement Learning for Robot Ccontrol," in *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, 2019, pp. 6023–6029.

[77] T. Silver, K. Allen, J. Tenenbaum, and L. Kaelbling, "Residual Policy Learning," *arXiv preprint arXiv:1812.06298*, 2018.

[78] Y. Yang, X. Meng, W. Yu, T. Zhang, J. Tan, and B. Boots, "Continuous Versatile Jumping Using Learned Action Residuals," in *Learning for Dynamics & Control (L4DC)*. PMLR, 2023, pp. 770–782.

[79] Y. Yang, G. Shi, X. Meng, W. Yu, T. Zhang, J. Tan, and B. Boots, "Cajun: Continuous adaptive jumping using a learned centroidal controller," *arXiv preprint arXiv:2306.09557*, 2023.

[80] J. Pravitra, K. A. Ackerman, C. Cao, N. Hovakimyan, and E. A. Theodorou, "L1-Adaptive MPPI Architecture for Robust and Agile Control of Multirotors," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2020, pp. 7661–7666.

[81] K. Lee, J. Gibson, and E. A. Theodorou, "Aggressive Perception-Aware Navigation Using Deep Optical Flow Dynamics and PixelMPC," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 1207–1214, 2020.

[82] D. Hanover, P. Foehn, S. Sun, E. Kaufmann, and D. Scaramuzza, "Performance, Precision, and Payloads: Adaptive Nonlinear MPC for Quadrotors," *IEEE Robotics and Automation Letters*, vol. 7, no. 2, pp. 690–697, 2021.

[83] S. Sun, A. Romero, P. Foehn, E. Kaufmann, and D. Scaramuzza, "A Comparative Study of Nonlinear MPC and Differential-Flatness-Based Control for Quadrotor Agile Flight," *IEEE Transactions on Robotics*, vol. 38, no. 6, pp. 3357–3373, 2022.

[84] E. Kaufmann, A. Loquercio, R. Ranftl, M. Müller, V. Koltun, and D. Scaramuzza, "Deep Drone Acrobatics," *arXiv preprint arXiv:2006.05768*, 2020.

[85] Y. Zhang, W. Wang, P. Huang, and Z. Jiang, "Monocular Vision-Based Sense and Avoid of UAV Using Nonlinear Model Predictive Control," *Robotica*, vol. 37, no. 9, pp. 1582–1594, 2019.

[86] A. Molchanov, T. Chen, W. Hönig, J. A. Preiss, N. Ayanian, and G. S. Sukhatme, "Sim-to-(Multi)-Real: Transfer of Low-Level Robust Control Policies to Multiple Quadrotors," in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2019, pp. 59–66.

[87] D. Zhang, A. Loquercio, X. Wu, A. Kumar, J. Malik, and M. W. Mueller, "Learning a Single Near-hover Position Controller for Vastly Different Quadcopters," in *2023 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2023, pp. 1263–1269.

[88] J. Hwangbo, I. Sa, R. Siegwart, and M. Hutter, "Control of a Quadrotor with Reinforcement Learning," *IEEE Robotics and Automation Letters*, vol. 2, no. 4, pp. 2096–2103, 2017.

[89] E. Kaufmann, L. Bauersfeld, and D. Scaramuzza, "A Benchmark Comparison of Learned Control Policies for Agile Quadrotor Flight," in *2022 International Conference on Robotics and Automation (ICRA)*. IEEE, 2022, pp. 10 504–10 510.
[90] A. Romero, Y. Song, and D. Scaramuzza, "Actor-Critic Model Predictive Control," *arXiv preprint arXiv:2306.09852*, 2023.
[91] Y. Song and D. Scaramuzza, "Policy Search for Model Predictive Control with Application to Agile Drone Flight," *IEEE Transactions on Robotics*, vol. 38, no. 4, pp. 2114–2130, 2022.
[92] 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)*, vol. 32, pp. 8026–8037, 2019.
[93] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel, "High-Dimensional Continuous Control Using Generalized Advantage Estimation," *arXiv preprint arXiv:1506.02438*, 2015.
[94] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," *arXiv preprint arXiv:1412.6980*, 2014.
[95] J. H. Halton, "Algorithm 247: Radical-inverse quasi-random point sequence," *Communications of the ACM*, vol. 7, no. 12, pp. 701–702, 1964.