DeePC-Hunt: Data-enabled Predictive Control Hyperparameter Tuning via Differentiable Optimization

Michael Cummins[1] M.CUMMINS24@IMPERIAL.AC.UK
Alberto Padoan[2] APADOAN@ECE.UBC.CA
Keith Moffat[3] KMOFFAT@ETHZ.CH
Florian Dörfler[3] DORFLER@CONTROL.EE.ETHZ.CH
John Lygeros[3] JLYGEROS@ETHZ.CH
[1]*Department of Electrical and Electronic Engineering, Imperial College London*
[2]*Department of Electrical and Computer Engineering, University of British Columbia*
[3]*Department of Electrical Engineering and Information Technology, ETH Zürich*

Editors: N. Ozay, L. Balzano, D. Panagou, A. Abate

Abstract

This paper introduces **D**ata-**e**nabl**e**d **P**redictive **C**ontrol **H**yperparameter T**un**ing via Differen**t**iable Optimization (DeePC-Hunt), a backpropagation-based method for automatic hyperparameter tuning of the DeePC algorithm. The necessity for such a method arises from the importance of hyperparameter selection to achieve satisfactory closed-loop DeePC performance. The standard methods for hyperparameter selection are to either optimize the open-loop performance, or use manual guess-and-check. Optimizing the open-loop performance can result in unacceptable closed-loop behavior, while manual guess-and-check can pose safety challenges. DeePC-Hunt provides an alternative method for hyperparameter tuning which uses an approximate model of the system dynamics and backpropagation to directly optimize hyperparameters for the closed-loop DeePC performance. Numerical simulations demonstrate the effectiveness of DeePC in combination with DeePC-Hunt in a complex stabilization task for a nonlinear system and its superiority over model-based control strategies in terms of robustness to model misspecifications.[1]

Keywords: Data-driven Control, Differentiable Optimization, Hyperparameter Tuning

1. Introduction

Over the past decade, direct data-driven control methods have experienced a surge of interest (Coulson et al., 2019a; van Waarde et al., 2020; Waarde et al., 2020; Xue and Matni, 2021), primarily fueled by the increasing adoption of machine learning techniques and the availability of vast datasets. Unlike traditional control design paradigms that rely on system identification followed by model-based control, these methods compute control actions directly from data. Leveraging tools from behavioral system theory, uncertainty quantification and convex optimization, data-driven methods have shown promise in simplifying the control design process, while achieving satisfactory control performance with reduced implementation effort.

Among the methods in this expanding literature, Data-enabled Predictive Control (DeePC) (Coulson et al., 2019a) stands out as an effective direct data-driven control algorithm. Operating similarly to Model Predictive Control (MPC) in a receding-horizon manner (Borrelli et al., 2017), DeePC circumvents the need for an accurate state-space model, relying instead on a Hankel matrix derived from offline input/output raw data. Although stability guarantees have been established

1. GitHub repo for reproducing experiments: https://github.com/michael-cummins/DeePC-HUNT

only under specific assumptions (Coulson et al., 2019a), the DeePC algorithm has demonstrated remarkable performance in controlling nonlinear systems affected by noise. This performance is mainly attributed to regularization techniques (Coulson et al., 2019b) and is theoretically justified by tools from distributionally robust optimization (Coulson et al., 2021). However, the performance of DeePC is often sensitive to the regularization parameters (Dörfler et al., 2023), presenting challenges when experiments are difficult, costly, or unsafe.

Today, the two standard methods for regularization tuning are analytical methods for the open-loop and manual guess-and-check. Open-loop methods, such as Final Control Error (Chiuso et al., 2023), rely on structural assumptions of the objective function and often yield overly conservative performance, as they do not account for DeePC's ability to replan at each timestep. Manual guess-and-check methods, on the other hand, optimize the closed-loop performance but require experiments on the real system, which may be costly or even impossible. In contrast, DeePC-Hunt directly optimizes hyperparameters for receding-horizon closed-loop performance via offline backpropagation, without requiring strong modeling assumptions or direct interaction with the system.

This paper introduces **D**ata-**e**nabl**e**d **P**redictive **C**ontrol **H**yperparameter T**un**ing via Differen**t**iable Optimization (DeePC-Hunt), a backpropagation-based method designed to automate hyperparameter tuning for DeePC. *DeePC-Hunt interprets DeePC, i.e., the solution of the DeePC optimization, as a control policy, and interprets the DeePC regularization hyperparameters as the parameters of the policy*. Leveraging this DeePC-as-policy interpretation and an approximate model of the system dynamics, DeePC-Hunt optimizes the hyperparameters (to local optimality) by backpropagating (Rumelhart et al., 1986) closed-loop performance. Specifically, DeePC-Hunt implements a (constrained) variant of the resilient backpropagation algorithm (Riedmiller and Braun, 1993) and widely-used automatic differentiation tools (Paszke et al., 2019) to directly optimize the regularization hyperparameters of a DeePC policy based on the closed-loop performance of the policy deployed on the approximate model in offline experiments. Our findings suggest that, given a reasonably approximated model, DeePC-Hunt yields performant hyperparameters for the true system. Moreover, DeePC-Hunt demonstrates superior robustness to model inaccuracies over traditional model-based control methods.

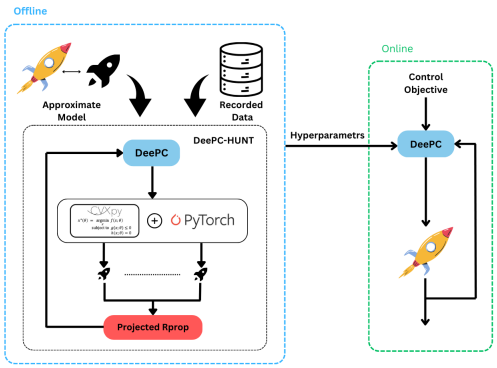

Figure 1: The DeePC policy is instantiated using CvxpyLayers and PyTorch to enable automatic differentiation. Simulations are carried out on an approximate model of the system and projected resilient backpropagation is used to update the hyperparameters.

Related work: DeePC-Hunt is motivated and inspired by policy optimization algorithms (Hu et al., 2023), which iteratively refine control policy parameters to minimize cumulative cost using different versions of the (projected) gradient descent algorithm. Unlike MPC or DeePC, DeePC-Hunt yields policies expressed as explicitly differentiable functions mapping states to actions. The implicit function theorem (Amos and Kolter, 2017) can be then used to differentiate the solution map of a Convex Optimization Control Policy (COCP), where Karush-Kuhn-Tucker (KKT) conditions are differentiated to compute the gradient of the control input with respect to the hyperparameters. The work closest to ours are (Amos et al., 2018; Zuliani et al., 2023), which extend the theory of differentiating KKT conditions theory to MPC policies for automatic parameter tuning via gradient

descent methods. However, (Amos et al., 2018; Zuliani et al., 2023) explicitly rely on state-space models, which can lead to unacceptable performance if the model is inaccurate. A broader approach to policy optimization for COCPs is presented by (Agrawal et al., 2020), with experimental validation demonstrating the effectiveness of differentiable optimization for tuning a wide range of COCPs. The use of an approximate model in DeePC-Hunt is further motivated by results in sample-efficient policy optimization (Song et al., 2024; Wiedemann et al., 2023), where policy gradients are computed directly rather than estimated. Unlike previous approaches, DeePC-Hunt optimizes DeePC policies by integrating data from a system with data obtained from numerical simulations of an approximate model of the system, resulting in improved closed-loop performance.

Contributions: The main contributions of the paper are threefold.

i) We formulate the problem of hyperparameter tuning for DeePC policies, ensuring compatibility with backpropagation, and propose the DeePC-Hunt algorithm as an effective solution.

ii) To efficiently implement DeePC-Hunt, we introduce a variant of the resilient backpropagation algorithm (Riedmiller and Braun, 1993) that supports the use of constraints (e.g. box constraints) using off-the-shelf automatic differentiation tools (Paszke et al., 2019).

iii) We validate DeePC-Hunt on a challenging benchmark task: landing a Vertical Takeoff Vertical Landing (VTVL) vehicle on an oceanic platform using a high-fidelity Gym simulation environment (Brockman et al., 2016).

Paper organization: The paper is organized as follows. Section 2 presents a concise overview of differentiable convex optimization layers, DeePC, and a version of Resilient Backpropagation (Riedmiller and Braun, 1993). Section 3 introduces the DeePC-Hunt algorithm. Section 4 demonstrates the robustness and performance of DeePC-Hunt on the VTVL landing task, comparing its performance against MPC under model mismatch. Section 5 concludes the paper with a summary of our findings.

Notation: The set of real numbers is denoted by $\mathbb{R}$. The set of real n-dimensional vectors is denoted by $\mathbb{R}^n$. The set of real $n \times m$-dimensional matrices is denoted by $\mathbb{R}^{n\times m}$. The set of positive integers is denoted by $\mathbb{N}$. The set of non-negative real numbers is denoted by $\mathbb{R}_+$. The set of $n \times n$-dimensional symmetric positive definite matrices is denoted by $\mathbb{S}_+^{n\times n}$. The set of closed, convex cones in $\mathbb{R}^m$ is denoted by $\mathcal{C}^m$. The transpose of the matrix $M \in \mathbb{R}^{p\times m}$ is denoted by $M^\intercal$. The p-norm of the vector $x \in \mathbb{R}^n$ is denoted by $|x|_p$. The weighted norm of the vector $x \in \mathbb{R}^n$ induced by the matrix $Q \in \mathbb{S}_+^{n\times n}$ is denoted by $|x|_Q$ and defined as $|x|_Q = (x^\intercal Q x)^{1/2}$. The i^{th} entry of a vector $v^{\mathrm{d}} \in \mathbb{R}^n$ is denoted by $v^{\mathrm{d}}|_i$. The expectation operator is denoted by $\mathbb{E}$. The function $\mathrm{sign} : \mathbb{R} \to \{-1, 0, 1\}$ is defined as -1 if the argument is negative, 1 if the argument is positive, and 0 if the argument is zero, respectively. Given two vectors $v \in \mathbb{R}^n$ and $w \in \mathbb{R}^m$, we define $\mathrm{col}(v, w) := (v^\intercal, w^\intercal)^\intercal$. Given $T \in \mathbb{N}$, the Hankel matrix of depth $L \in \mathbb{N}$, with $L \leq T$, associated with the vector $w \in \mathbb{R}^{qT}$ is denoted by $H_L(w)$ (see, e.g., (Coulson et al., 2019a) for the definition).

2. Background

2.1. Differentiable Convex Optimization Layers

Consider the parameterized convex optimization problem

$$\min_{x\in\mathbb{R}^n} \; f(x|\theta) \;\text{ s.t. }\; g(x|\theta) \leq 0, \;\; h(x|\theta) = 0, \tag{1}$$

where $f : \mathbb{R}^n \to \mathbb{R}$ and $g : \mathbb{R}^n \to \mathbb{R}^{m_1}$ are convex functions, $h : \mathbb{R}^n \to \mathbb{R}^{m_2}$ is affine and $\theta \in \mathbb{R}^\ell$ represents the vector of parameters defininng f, g and h, respectively. The parameter θ is assumed to

belong to a given set $\Theta \subseteq \mathbb{R}^\ell$, which represents the problem parameters. The solution to the class of convex optimization problems of the form (1), known as *disciplined parameterized programs* (Agrawal et al., 2019a), may be defined as a mapping $\theta \mapsto x^*(\theta)$ from the parameter θ to the (global) solution $x^*(\theta)$ of (1). Formally, we define the solution map of (1) as

$$s(\theta) := \theta \mapsto x^*(\theta). \tag{2}$$

Differentiable convex optimization layers (Agrawal et al., 2019a) provides a method for differentiating the solution-map (2) with respect to the parameter θ. To this end, (1) is first transformed into a convex cone problem defined as

$$\min_{(x,\nu)\in\mathbb{R}^n\times\mathbb{R}^m} \quad c^\mathsf{T} x \ \text{ s.t. } \ Ax + \nu = b, \ \ (x,\nu) \in \mathbb{R}^n \times \mathcal{K}, \tag{3}$$

where $A \in \mathbb{R}^{m\times n}$, $b \in \mathbb{R}^m$, $c \in \mathbb{R}^n$ and $\mathcal{K} \in \mathcal{C}^m$ are transformations of the problem parameter θ. Such a transformation is typically performed through domain-specific languages for convex programming, such as CVXPY (Diamond and Boyd, 2016), where the subset of parameters (A, b, c) that affect the gradient are determined by a sparse (linear) projection $L(\theta) = (A, b, c)$ (Agrawal et al., 2018). Thus, the solution map (2) is decomposed as

$$s(\theta) = (R \circ S \circ L)(\theta), \tag{4}$$

where $L : \mathbb{R}^\ell \to \mathbb{R}^{m\times n} \times \mathbb{R}^m \times \mathbb{R}^n$ maps the problem data of (1) to the problem data of (3), $S : \mathbb{R}^{m\times n} \times \mathbb{R}^m \times \mathbb{R}^n \to \mathbb{R}^n \times \mathcal{K}$ is the solution map of (3) and $R : \mathbb{R}^n \times \mathcal{K} \to \mathbb{R}^n$ maps the solution of (3) to the solution of (1). Specifically, $S(A, b, c) = (x^*, \nu^*)$ and $R(x^*, \nu^*) = x^*$. The gradient of the solution map is then $\nabla s(\theta) = \nabla L(\theta)\nabla S(A, b, c)\nabla R(x^*, \nu^*)$. Since R and L are linear mappings by design, calculating their gradient is elementary. Furthermore, the gradient $\nabla S(A, b, c)$ may be obtained by differentiating the KKT conditions of (3), as detailed in (Agrawal et al., 2019b) (see also (Agrawal et al., 2018, 2019a)).

The CvxpyLayers (Agrawal et al., 2019a) Python package integrates CVXPY and PyTorch (Paszke et al., 2019) to automate the transformation (1) $\mapsto$ (3) and the differentiation of (4) with respect to θ. DeePC, explained in the following section, can be interpreted as a convex optimization problem of the form (1). CvxpyLayers will be later used in the implementation of DeePC-Hunt to instantiate the DeePC policy (6), allowing us to differentiate the control inputs with respect to the regularization parameters.

2.2. The DeePC Algorithm

Consider a discrete-time dynamical system described by the equations

$$x_{k+1} = f\left(x_k, u_k\right), \ \ y_k = h\left(x_k, u_k\right), \tag{5}$$

where $u \in \mathbb{R}^m$, $x \in \mathbb{R}^n$ and $y \in \mathbb{R}^p$ represent the input, state, and output of system (5), respectively. The DeePC algorithm leverages raw data matrices derived from offline input/output measurements of system (5) as a predictive model. Assuming the availability of input/output data recorded offline from system (5), let $u^\mathrm{d} = \mathrm{col}(u^{\mathrm{d}|1}, \ldots, u^{\mathrm{d}|T}) \in \mathbb{R}^{mT}$ and $y^\mathrm{d} = \mathrm{col}(y^{\mathrm{d}|1}, \ldots, y^{\mathrm{d}|T}) \in \mathbb{R}^{pT}$ be vectors containing an input sequence of length $T \in \mathbb{N}$ applied to system (5) and the corresponding output sequence, respectively. Let $q = m + p$ and define the data vector $w^\mathrm{d} = \mathrm{col}(u^\mathrm{d}, y^\mathrm{d}) \in \mathbb{R}^{qT}$. For given

initial and future time horizons $T_{\text{ini}} \in \mathbb{N}$ and $T_{\text{f}} \in \mathbb{N}$, we define the input and output data matrices for the past and future timesteps as $\begin{pmatrix} U_{\text{p}} \\ U_{\text{f}} \end{pmatrix} = H_{T_{\text{ini}}+T_{\text{f}}}(u^{\text{d}})$ and $\begin{pmatrix} Y_{\text{p}} \\ Y_{\text{f}} \end{pmatrix} = H_{T_{\text{ini}}+T_{\text{f}}}(y^{\text{d}})$, respectively. Furthermore, we define the past and future combined input-and-output data matrices as $W_{\text{p}} = \begin{pmatrix} U_{\text{p}} \\ Y_{\text{p}} \end{pmatrix}$ and $W_{\text{f}} = \begin{pmatrix} U_{\text{f}} \\ Y_{\text{f}} \end{pmatrix}$, respectively.

Given a reference trajectory $w^{\text{ref}} = \text{col}(u^{\text{ref}|1}, \dots, u^{\text{ref}|T_{\text{f}}}, y^{\text{ref}|1}, \dots, y^{\text{ref}|T_{\text{f}}}) \in \mathbb{R}^{qT_{\text{f}}}$, an input constraint set $\mathcal{U} \subseteq \mathbb{R}^{mT_{\text{f}}}$, an output constraint set $\mathcal{Y} \subseteq \mathbb{R}^{pT_{\text{f}}}$, an input cost matrix $R \in \mathbb{S}_+^{m\times m}$, an output cost matrix $Q \in \mathbb{S}_+^{p\times p}$, a regularization function $\psi : \mathbb{R}^{T-T_{\text{ini}}-T_{\text{f}}+1} \times \mathbb{R}^{pT_{\text{ini}}} \to \mathbb{R}_+^r$, and a weight vector $\lambda \in \mathbb{R}_+^r$, the DeePC algorithm operates in a receding-horizon fashion by iteratively solving the optimization problem

$$\begin{aligned} \min_{u,y,g,\sigma_y} \; & \sum_{i=1}^{T_{\text{f}}} |y_i - y^{\text{ref}|i}|_Q^2 + |u_i - u^{\text{ref}|i}|_R^2 + \lambda^\intercal \psi(g, \sigma_y), \\ \text{s.t.} \; & \begin{pmatrix} U_{\text{p}} \\ Y_{\text{p}} \\ U_{\text{f}} \\ Y_{\text{f}} \end{pmatrix} g = \begin{pmatrix} u_k^{\text{ini}} \\ y_k^{\text{ini}} \\ u \\ y \end{pmatrix} + \begin{pmatrix} 0 \\ \sigma_y \\ 0 \\ 0 \end{pmatrix}, \;\; (u, y) \in \mathcal{U} \times \mathcal{Y}, \end{aligned} \tag{6}$$

where $u_k^{\text{ini}} = \text{col}(u_{k-T_{\text{ini}}}, \dots, u_{k-1}) \in \mathbb{R}^{mT_{\text{ini}}}$ and $y_k^{\text{ini}} = \text{col}(y_{k-T_{\text{ini}}}, \dots, y_{k-1}) \in \mathbb{R}^{pT_{\text{ini}}}$ are vectors containing the T_{ini} most recent input and output measurements at time k, respectively.

Let $w_k^{\text{ini}} = \text{col}(u_k^{\text{ini}}, y_k^{\text{ini}}) \in \mathbb{R}^{qT_{\text{ini}}}$ and note that (6) is a parametric convex optimization problem with parameter λ, we define its solution map in terms of λ, w_k^{ini} and w_k^{ref} as $\mathcal{S}\left(\lambda, w_k^{\text{ini}}, w^{\text{ref}}\right) := \left(\lambda, w_k^{\text{ini}}, w^{\text{ref}}\right) \mapsto (w^*, g^*, \sigma_y^*)$, where $w^* = \text{col}(u^*, y^*)$ and (w^*, g^*, σ_y^*) is the optimal solution of the optimization problem (6). Furthermore, defining the (linear) projection $\mathcal{P}(w^*, g^*, \sigma_y^*) = u_0^*$, where $u_0^* \in \mathbb{R}^m$ is the vector defined by the first m entries of w^*, we rewrite the DeePC control policy as

$$\pi^\lambda(w_k^{\text{ini}}, w^{\text{ref}}) := (\mathcal{P} \circ \mathcal{S}) \left(\lambda, w_k^{\text{ini}}, w^{\text{ref}}\right). \tag{7}$$

The original formulation of the DeePC algorithm in (Coulson et al., 2019a) employs a one-norm regularizer on the decision variable $g \in \mathbb{R}^{T-T_{\text{ini}}-T_{\text{f}}+1}$ to promote the selection of low-complexity models. Alternatively, as detailed in (Dörfler et al., 2023), one may use the Elastic-Net Regularization function which leads to consistent predictions and connects to classic Subspace Predictive Control (SPC) algorithms (Favoreel and De Moor, 1999). In this work, we combine such regularizer with a one-norm regularization to promote sparsity in the slack variable σ_y using the regularization function

$$\lambda^\intercal \psi(g, \sigma_y) = \lambda^0 \left|(I - \Pi)g\right|_2^2 + \lambda^1 \left|g\right|_1 + \lambda^2 |\sigma_y|_1, \tag{8}$$

where $\lambda = \text{col}(\lambda^0, \lambda^1, \lambda^2)$ and $\Pi = \begin{pmatrix} W_{\text{p}} \\ Y_{\text{f}} \end{pmatrix}^\dagger \begin{pmatrix} W_{\text{p}} \\ Y_{\text{f}} \end{pmatrix}$.

2.3. Projected Resilient Backpropagation

Consider the constrained optimization problem

$$\min_{\lambda \in \Lambda} \; \phi(\lambda),$$

where $\Lambda \subseteq \mathbb{R}^r$ and $\phi : \mathbb{R}^r \to \mathbb{R}$ is differentiable. Finding a locally optimal solution $\lambda^* \in \mathbb{R}^r$ may be challenging with a first-order iterative optimization algorithm if $|\nabla\phi(\lambda)|$ is small for all $\lambda \in \mathbb{R}^n$, regardless of the proximity to a stationary point. Resilient backpropagation (Riedmiller and Braun, 1993) is a heuristic first-order iterative optimization algorithm that mitigates this issue by optimizing over each scalar element of the decision variable $\lambda = \text{col}(\lambda^0, \dots, \lambda^{r-1})$ using only the *sign* of the gradient and an adaptive step-size. The update rule of the algorithm is defined as

$$\lambda^i_{k+1} = \lambda^i_k - \eta^i_k \text{sign}(\nabla_{\lambda^i}\phi(\lambda^i_k)), \;\; \eta^i_{k+1} = \text{Rprop}(\eta^i_k, \lambda^i_k, \lambda^i_{k+1} | \eta^{\max}, \eta^{\min}, \beta, \alpha), \tag{9}$$

where $\text{Rprop} : \mathbb{R}_+ \times \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ is the step-size generator defined as

$$\text{Rprop}(\eta^i_k, \lambda^i_k, \lambda^i_{k+1} | \eta^{\max}, \eta^{\min}, \beta, \alpha) = \begin{cases} \min\{\alpha\eta^i_k, \eta_{\max}\}, & \text{if } \nabla_{\lambda^i}\phi(\lambda^i_{k+1})\nabla_{\lambda^i}\phi(\lambda^i_k) < 0, \\ \max\{\beta\eta^i_k, \eta_{\min}\}, & \text{if } \nabla_{\lambda^i}\phi(\lambda^i_{k+1})\nabla_{\lambda^i}\phi(\lambda^i_k) > 0, \\ \eta^i_k, & \text{if } \nabla_{\lambda^i}\phi(\lambda^i_{k+1})\nabla_{\lambda^i}\phi(\lambda^i_k) = 0, \end{cases}$$

with $(\eta^{\max}, \eta^{\min}, \beta, \alpha) \in \mathbb{R}^4_+$ being the design parameters representing the minimum step-size, maximum step-size, decay factor, and growth factor, respectively.

Resilient backpropagation is particularly useful in the context of DeePC-Hunt, because the (locally) optimal DeePC regularization parameters can be orders of magnitude away from an initial guess (Dörfler et al., 2023). However, each regularization term λ^i must be non-negative. To enforce this constraint, we propose a variant of (9) defined by the update rule

$$\begin{aligned} \gamma^i_{k+1} =& \gamma^i_k - \eta^i_k \text{sign}(\nabla_{\lambda^i}\phi(\lambda^i_k)), \;\; \eta^i_{k+1} = \text{Rprop}(\eta^i_k, \gamma^i_k, \gamma^i_{k+1} | \eta^{\max}, \eta^{\min}, \beta, \alpha), \quad \forall i = 1, \cdots, r \\ \lambda_{k+1} =& P_\Lambda\left(\gamma_{k+1}\right), \end{aligned} \tag{10}$$

where P_Λ is the projection operator onto the set Λ, defined as $P_\Lambda(\lambda) = \text{argmin}_{\mu\in\Lambda} \frac{1}{2}|\lambda - \mu|^2$. The projection onto the set Λ ensures that the constraint $\lambda^k \in \Lambda$ is satisfied at every iteration. To the best of the authors' knowledge, this is the first work to incorporate resilient backpropagation with a projection scheme to enforce feasibility in the iterates of (9). Throughout this paper, we assume that $P_\Lambda(\lambda)$ is easy to compute, as is the case for a range of common sets (Parikh and Boyd, 2014), including half-spaces, hyperrectangles, simplices, and certain cones.

3. DeePC-Hunt

The regularization parameter λ has a significant impact on the closed-loop performance of DeePC (Coulson et al., 2019a). DeePC-Hunt tunes λ to optimize the closed-loop performance of a DeePC policy implemented on a simplified or inaccurate surrogate model, leveraging the intuition that optimal regularization parameters for "similar" systems are typically "close." This approach enables offline hyperparameter tuning with a minimal sim-to-real gap: by optimizing λ with respect to the closed-loop performance on the surrogate model, we obtain a regularizer λ that leads to acceptable performance on the true system. We consider a surrogate model described by the equations

$$\tilde{x}_{k+1} = \tilde{f}\left(\tilde{x}_k, \tilde{u}_k\right), \;\; \tilde{y}_k = \tilde{h}\left(\tilde{x}_k, \tilde{u}_k\right), \tag{11}$$

where $\tilde{u} \in \mathbb{R}^m$, $\tilde{x} \in \mathbb{R}^n$ and $\tilde{y} \in \mathbb{R}^p$, respectively. The functions $\tilde{f}$ and $\tilde{h}$ are assumed to approximate the input-output behavior of system (5) with respect to a given metric (e.g., $\mathcal{L}_2$ gain). The approximate

model may be derived from the physics of the system (5) or through the use of model reduction techniques (see, e.g., (Antoulas, 2005) for an overview of available techniques).

Given the approximate model (11), the vector w_1^{ini} consisting of the most recent T_{ini} measurements at timestep 1, the reference trajectory w^{ref}, and the DeePC policy π^λ (Section 2.2), we define the approximate closed-loop cost, $\tilde{C}^{\pi^\lambda} : \mathbb{R}^{qT_{\text{ini}}} \times \mathbb{R}^{qT_{\text{f}}} \to \mathbb{R}_+$, over the simulation length N as

$$\tilde{C}^{\pi^\lambda}\left(w_1^{\text{ini}}, w^{\text{ref}}\right) = \sum_{i=1}^{N} |\tilde{y}_i - y^{\text{ref}|i}|_Q^2 + |u_i^* - u^{\text{ref}|i}|_R^2, \tag{12}$$

where u_i^* is the input produced by $\pi^\lambda\left(w_i^{\text{ini}}, w^{\text{ref}}\right)$ in (7) and $\tilde{y}_i$ is the output of the surrogate model given by (11) when u_i^* is applied. At each timestep i, w_{i+1}^{ini} is given by inserting u_i^* and $\tilde{y}_i$ into w_i^{ini}, and removing the measurements furthest in the past corresponding to time $i - T_{\text{ini}}$. The initial state $\tilde{x}_0$ required by the surrogate model (11) is determined from w_1^{ini} using a state estimation method (e.g., a Kalman Filter). We define $\zeta(\cdot)$ as the selection operator which implements the Kalman Filter, i.e., $\zeta(w_i^{\text{ini}}) = \tilde{x}_{i-1}$. Moreover, we define the operators $\Omega_u(\cdot)$ and $\Omega_y(\cdot)$ such that for a given w_i^{ini}, we have $\Omega_u(w_i^{\text{ini}}) = (u_{i-T_{\text{ini}}+2}^{\text{ini}}, \ldots, u_i^{\text{ini}})$ and $\Omega_y(w_i^{\text{ini}}) = (y_{i-T_{\text{ini}}+2}^{\text{ini}}, \ldots, y_i^{\text{ini}})$.

The aim of DeePC-Hunt is to find a λ that is effective for all initial conditions. Therefore, the expected closed-loop cost over a probability distribution of initial conditions w_1^{ini} is optimized [2]. For simplicity, the w_1^{ini} distribution is taken to be the uniform distribution over the columns of W_{p}, denoted $\mathcal{D}(W_{\text{p}})$. Thus, the expected closed-loop cost is $\mathbb{E}_{w_1^{\text{ini}} \sim \mathcal{D}(W_{\text{p}})}\left[\tilde{C}^{\pi^\lambda}\left(w_1^{\text{ini}}, w^{\text{ref}}\right)\right]$ and the constrained, non-convex DeePC-Hunt optimization problem is defined by (13) the bilevel optimization problem

$$\begin{aligned} \min_{\lambda \in \Lambda} \ & \mathbb{E}_{w_1^{\text{ini}} \sim \mathcal{D}(W_{\text{p}})}\left[\tilde{C}^{\pi^\lambda}\left(w_1^{\text{ini}}, w^{\text{ref}}\right)\right], \\ \text{s.t.} \ & \tilde{x}_0 = \zeta(w_1^{\text{ini}}), \ \ \tilde{x}_{i+1} = \tilde{f}\left(\tilde{x}_i, \tilde{u}_i\right), \ \ \tilde{y}_i = \tilde{h}\left(\tilde{x}_i, \tilde{u}_i\right), \\ & \tilde{u}_i = \pi^\lambda\left(w_i^{\text{ini}}, w^{\text{ref}}\right), \ \ w_{i+1}^{\text{ini}} = \text{col}(\Omega_u(w_i^{\text{ini}}), \tilde{u}_{i+1}, \Omega_y(w_i^{\text{ini}}), \tilde{y}_{i+1}), \end{aligned} \tag{13}$$

where $\Lambda \subseteq \mathbb{R}_+^r$ is selected to enforce non-negative box constraints on each λ^i. Recalling (7), π^λ is the solution map of a disciplined parameterized program with a linear projection applied. Therefore, π^λ may be differentiated using CvxpyLayers, which leverages the technique discussed in Section 2.1. Thus, the projected resilient backpropagation method described in (10) can be used to compute a locally optimal solution λ^* of (13).

The DeePC-Hunt optimization problem (13) is an empirical risk minimization with dataset $\mathcal{D}(W_{\text{p}})$. Thus, while the expectation (13) can be computed, it may be prohibitively expensive if $\mathcal{D}(W_{\text{p}})$ has many datapoints as the exact expectation requires numerous simulations to produce a single gradient step. To compute the approximate gradient of the expectation in (13) in a computationally tractable manner, a similar approach is taken to Stochastic Gradient Descent (SGD) methods by computing an unbiased estimate of (13) in the form of Monte Carlo samples, which is common practice when performing empirical risk minimization with large datasets (Ryu and Yin, 2022). Pseudo-code for the sample-estimate/Monte Carlo DeePC-Hunt implementation for solving (13) is given in Algorithm 1, where $B \in \mathbb{N}$ is the number of Monte Carlo samples taken at each step k and $N_{\text{iter}} \in \mathbb{N}$ is the number of steps.

2. A similar procedure could be followed for w^{ref}, if the DeePC policy needs to work for a distribution of reference trajectories as well.

Algorithm 1 DeePC-HUNT

Initialize: η_0^i, λ_0^i, γ_0^i, $\forall i = 1, \dots, r$
for $k = 1 : N_{\text{iter}}$ **do**
  for $j = 1 : B$ **in parallel do**
    $w_1^{\text{ini},j} \sim \mathcal{D}(W_{\text{p}})$
    $\tilde{x}_0^j = \zeta(w_1^{\text{ini},j})$
    for $i = 0 : N-1$ **do**
      $\tilde{u}_i^j \leftarrow \pi^{\lambda^k}\left(w_i^{\text{ini},j}, w^{\text{ref}}\right)$
      $(\tilde{x}_{i+1}^j, \tilde{y}_i^j) \leftarrow (\tilde{f}(\tilde{x}_i^j, \tilde{u}_i^j), \tilde{h}(\tilde{x}_i^j, \tilde{u}_i^j))$
      $w_{i+1}^{\text{ini},j} \leftarrow \text{col}(\Omega_u(w_i^{\text{ini},j}), \tilde{u}_i^j, \Omega_y(w_i^{\text{ini},j}), \tilde{y}_i^j)$
    end for
  end for
  $J(\lambda^k) \leftarrow \frac{1}{b}\sum_{j=1}^{b} \tilde{C}^{\pi^\lambda}\left(w^{\text{ini},j}, w^{\text{ref}}\right)$
  for $i = 1 : r$ **in parallel do**
    $\gamma_{k+1}^i \leftarrow \gamma_k^i - \eta_k^i \text{sign}(\nabla_\lambda J(\lambda_k^i))$
    $\eta_{k+1}^i \leftarrow \text{Rprop}\left(\eta_k^i, \gamma_k^i, \gamma_{k+1}^i \,|\eta^{\max}, \eta^{\min}, \beta, \alpha\right)$
  end for
  $\lambda_{k+1} \leftarrow P_\Lambda(\gamma_{k+1})$
end for

4. Numerical Simulation: landing a VTVL vehicle

4.1. Rocket Lander Gym Environment

The performance of DeePC-Hunt is demonstrated via the task of safely landing a Vertical Takeoff and Vertical Landing (VTVL) rocket. The objective is to land the VTVL vehicle on a designated oceanic platform in a vertical position. The problem is introduced in (Ferrante, 2017) and implemented using OpenAI's Gym Library (Brockman et al., 2016).

The dynamics of the VTVL vehicle are modeled by the nonlinear equations of motion

$$\begin{aligned} m\ddot{x} &= F_s \cos(\theta) - F_E \sin(\varphi + \theta) l_1, \qquad (14) \\ m\ddot{y} &= F_s \cos(\theta) - F_E \sin(\varphi + \theta) l_1 - mg, \\ m\ddot{\theta} &= F_E \sin(2\pi - \varphi) l_1 - F_s l_2, \end{aligned}$$

where $x(t) \in \mathbb{R}$ is the horizontal position of the rocket, $y(t) \in \mathbb{R}$ is the vertical position of the rocket, $\theta(t) \in \mathbb{R}$ is the vertical pitch of the rocket, $F_s(t) \in \mathbb{R}$ is the force exerted by the side engines, $F_E(t) \in \mathbb{R}_+$ is the force exerted by the main engine, $\varphi(t) \in \mathbb{R}$ is the heading angle of the main engine, $m(t) \in \mathbb{R}$ is the time-varying mass of the rocket due to loss of fuel, $l_1 \in \mathbb{R}_+$ is the length of the portion of the rocket below the center of gravity, and $l_2 \in \mathbb{R}_+$ is the length of the portion of the rocket above the center of gravity, respectively. For simplicity, we assume that

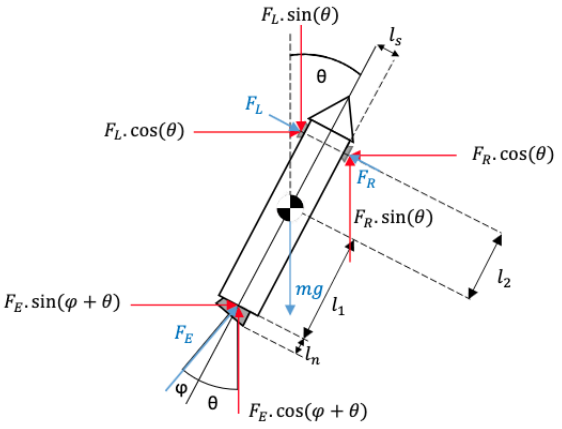

Figure 2: Free body diagram of the VTVL rocket from (Ferrante, 2017).

the landing pad is stationary, perfect state information is available, and control inputs are chosen as $u = (F_E, F_s, \varphi) \in \mathbb{R}^3$, thus facilitating full actuation. The state and input are also required to satisfy given state and input box constraints $x(t) \in X$, $y(t) \in Y$, $\theta(t) \in \Theta$, and $u(t) \in U$, respectively.

4.2. MPC Policy

With perfect knowledge of the system parameters, MPC can be used to land the VTVL vehicle on the designated landing pad (Rawlings et al., 2017). Providing an appropriate reference trajectory r, linearizing around an appropriately selected equilibrium point and discretizing the resulting system using zero-order-hold sampling gives the MPC control scheme

$$\begin{aligned} \min_{x \in \mathbb{R}^{6T_\mathrm{f}}, u \in \mathbb{R}^{3T_\mathrm{f}}} \quad & \sum_{i=1}^{T_\mathrm{f}} |x_i - r|_Q^2 + |u_i|_R^2, \\ \text{s.t.} \quad & x_0 = \hat{x}_0, \;\; x_{i+1} = Ax_i + Bu_i, \quad (x, u) \in (X \times Y \times \mathbb{R}^2 \times \Theta \times \mathbb{R} \times U)^{T_\mathrm{f}}. \end{aligned} \tag{15}$$

Numerical simulations suggest that the performance of policy (15) is highly sensitive on the quality of the state-space model (A, B). To illustrate this point, two distinct models are considered. The first model, named model A, uses the exact parameters of model (14). Model B uses inaccurately estimated parameters adjusted by increasing l_1 by 33% and by decreasing l_2 by 25%. Additionally, we set m to be time-invariant. In model A, m is set to the initial true weight $m(0)$, but is set to $\frac{1}{2}m(0)$ for model B. Fig. 3 illustrates the performance of the MPC policy. The rocket successfully lands using MPC (A), confirming the effectiveness of a well-estimated model coupled with a linearized MPC scheme (15). Performance significantly deteriorated with MPC (B), demonstrating that an inaccurate model can lead to unsatisfactory MPC performance on this system.

4.3. DeePC-Hunt Policy

Two DeePC-Hunt policies are evaluated, both using the same offline training data w_d to construct the Hankel matrices in (6). One policy optimizes λ using model A to define π^{λ_A}, while the other uses model B to define π^{λ_B}. Both policies employ the DeePC formulation in (6) with the regularization function in (8), setting $Q = \mathrm{diag}(100, 10, 5, 1, 3000, 30)$, $R = \mathrm{diag}(0.01, 0.01, 0.01)$, $T_\mathrm{f} = 10$, $T_\mathrm{ini} = 1$, input constraints $\mathcal{U} = U^{T_\mathrm{f}}$ and output constraints $\mathcal{Y} = (X \times Y \times \mathbb{R}^2 \times \Theta \times \mathbb{R})^{T_\mathrm{f}}$ where $X = [0, 33.33]$, $Y = [0, 26.66]$, $\Theta = [-0.61, 0.61]$ and $U = [0, 16118.5] \times [0, 322.37] \times [-0.26, 0.26]$. For the DeePC-Hunt training routine described in Algorithm 1, we initialize $\lambda_0 = (50, 50, 1000)$, set $N_\mathrm{iter} = 100$, $N = 20$, $B = 1$, $(\eta^{\max}, \eta^{\min}, \beta, \alpha) = (10^2, 10^{-3}, 1.2, 0.5)$, and $\Lambda = [10^{-5}, 10^5]^3$. Upon termination, the optimized parameters are $\lambda_A = (49.84, 8.36, 1000.05)$ for model A and $\lambda_B = (27.475, 2.128, 946.05)$ for model B.

Training data is collected by applying a persistently exciting input trajectory $u_\mathrm{d} \in \mathbb{R}^{mT}$ of length $T \in \mathbb{N}$ to the VTVL vehicle and recording the corresponding output trajectory $y_\mathrm{d} \in \mathbb{R}^{pT}$. The input trajectory is a Pseudo-Random Binary Sequence (PRBS) sequence starting from an equilibrium point $y_\mathrm{eq} \in \mathbb{R}^p$. Figure 3 illustrates that both DeePC-Hunt policies demonstrate nearly identical trajectories, even with model inaccuracies, highlighting the robustness of DeePC-Hunt.

4.4. Evaluation of Control Performance

We evaluate control performance using two metrics: (i) the realized cost, measured as the total system cost incurred over N_π time steps taken by policy π to land the VTVL vehicle, and (ii) the

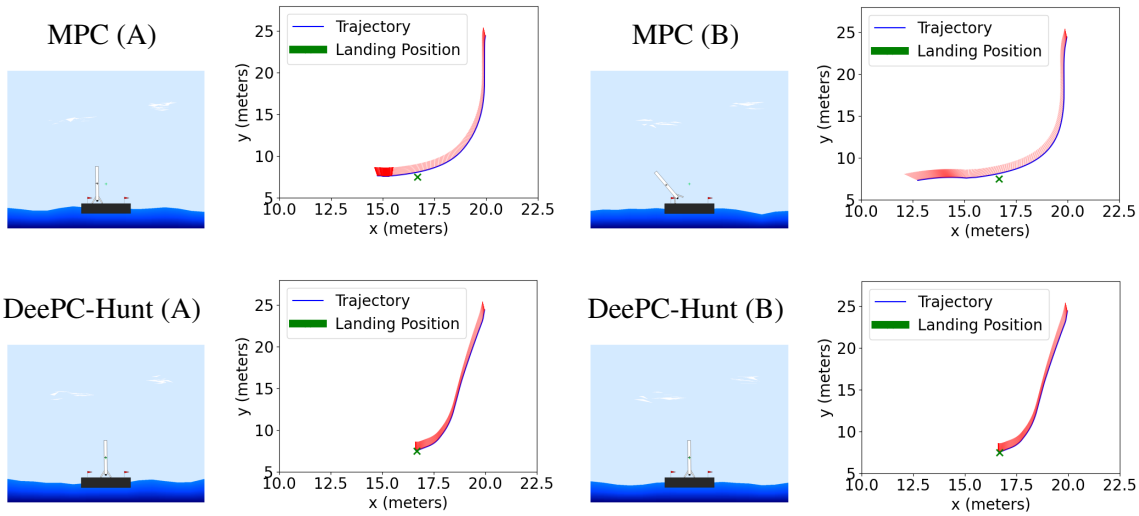

Figure 3: Performance of the MPC (top) and DeePC-Hunt (bottom) policies using models A (left) and B (right), respectively.

empirical success rate across $k = 50$ initial states uniformly sampled from a predefined range $(x, y) \in X_s \times Y_s$. The success rate represents the percentage of successful landings from these initial conditions. Results for MPC and DeePC-Hunt policies are summarized in Table 1.

	Cost ($\times 10^3$)	**% Success**
MPC (A)	11.228	46%
π^{λ_A}	16.678	56%
MPC (B)	8.213	22%
π^{λ_B}	12.165	66%

Table 1: Results for DeePC-Hunt and MPC policies using models A and B, with $X_s = [6.67, 26.66]$ and $Y_s = [18.66, 24]$.

For the sake of a fair comparison, we note that both DeePC-Hunt and MPC leverage approximate linear models, but their training pipelines differ: DeePC-Hunt relies on the system dynamics (11) for optimizing λ, while MPC uses a fixed linearization of (11). Despite this asymmetry, DeePC-Hunt yields robust closed-loop performance and significantly outperforms MPC in success rate in the presence of model inaccuracies. However, a trade-off emerges as MPC exhibits lower closed-loop costs upon successful landings. This discrepancy primarily arises from the impact of regularization on the cost function in (6), prioritizing regularization (g, σ_y) over progress towards the setpoint w^{ref}. Employing model B with DeePC-Hunt yields significantly improved performance over direct MPC.

5. Conclusion

This paper introduced DeePC-Hunt, a backpropagation-based method for automatic hyperparameter tuning of the DeePC algorithm. We demonstrated its effectiveness, showing that DeePC-Hunt outperforms model-mismatched MPC on a challenging nonlinear control task without manual tuning. Our results highlight that even with a poor approximation of the true dynamics, effective regularization parameters can be identified using backpropagation to ensure strong closed-loop performance on the true system. Given its simplicity and ease of implementation, DeePC-Hunt offers a valuable solution for automated parameter tuning, especially when exploring multiple parameters is expensive, but obtaining an approximate model is comparatively simpler.

Acknowledgments

The authors would like to thank Dylan Vogel and Gerasimos Maltezos for contributing to the development of the Gym environment, as well as Joshua Näf for writing the code to generate the trajectory plots.

References

Akshay Agrawal, Robin Verschueren, Steven Diamond, and Stephen Boyd. A rewriting system for convex optimization problems. *Journal of Control and Decision*, 5(1):42–60, 2018.

Akshay Agrawal, Brandon Amos, Shane Barratt, Stephen Boyd, Steven Diamond, and J Zico Kolter. Differentiable convex optimization layers. *Advances in neural information processing systems*, 32, 2019a.

Akshay Agrawal, Shane Barratt, Stephen Boyd, Enzo Busseti, and Walaa M Moursi. Differentiating through a cone program. *arXiv preprint arXiv:1904.09043*, 2019b.

Akshay Agrawal, Shane Barratt, Stephen Boyd, and Bartolomeo Stellato. Learning convex optimization control policies. In *Learning for Dynamics and Control*, pages 361–373. PMLR, 2020.

Brandon Amos and J Zico Kolter. Optnet: Differentiable optimization as a layer in neural networks. In *International conference on machine learning*, pages 136–145. PMLR, 2017.

Brandon Amos, Ivan Jimenez, Jacob Sacks, Byron Boots, and J Zico Kolter. Differentiable MPC for end-to-end planning and control. *Advances in neural information processing systems*, 31, 2018.

Athanasios C Antoulas. *Approximation of large-scale dynamical systems*. SIAM, 2005.

Francesco Borrelli, Alberto Bemporad, and Manfred Morari. *Predictive Control for Linear and Hybrid Systems*. Cambridge University Press, 2017.

G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba. OpenAI Gym. *arXiv preprint arXiv:1904.09043*, 2016.

Alessandro Chiuso, Marco Fabris, Valentina Breschi, and Simone Formentin. Harnessing uncertainty for a separation principle in direct data-driven predictive control. *arXiv preprint arXiv:2312.14788*, 2023.

Jeremy Coulson, John Lygeros, and Florian Dörfler. Data-enabled predictive control: In the shallows of the deepc. *European Control Conference (ECC)*, 2019a.

Jeremy Coulson, John Lygeros, and Florian Dörfler. Regularized and distributionally robust data-enabled predictive control. *Conference on Decision and Control (CDC)*, 2019b.

Jeremy Coulson, John Lygeros, and Florian Dörfler. Distributionally robust chance constrained data-enabled predictive control. *IEEE Transactions on Automatic Control*, 67(7):3289–3304, 2021.

Steven Diamond and Stephen Boyd. CVXPY: A Python-embedded modeling language for convex optimization. *Journal of Machine Learning Research*, 17(83):1–5, 2016.

Florian Dörfler, Jeremy Coulson, and Ivan Markovsky. Bridging direct and indirect data-driven control formulations via regularizations and relaxations. *IEEE Transactions on Automatic Control*, 68(2):883–897, 2023.

Wouter Favoreel and Bart De Moor. SPC: Subspace predictive control. *IFAC Proceedings Volumes*, 32, 1999.

Reuben Ferrante. A robust control approach for rocket landing, 2017.

Bin Hu, Kaiqing Zhang, Na Li, Mehran Mesbahi, Maryam Fazel, and Tamer Başar. Toward a theoretical foundation of policy optimization for learning control policies. *Annual Review of Control, Robotics, and Autonomous Systems*, 6(1):123–158, 2023.

Neal Parikh and Stephen Boyd. Proximal algorithms. *Foundations and trendsin Optimization*, 1(3): 127–239, 2014.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. *Advances in Neural Information Processing Systems 32*, 2019.

James Blake Rawlings, David Q Mayne, Moritz Diehl, et al. *Model predictive control: theory, computation, and design*, volume 2. Nob Hill Publishing Madison, WI, 2017.

Martin Riedmiller and Heinrich Braun. A direct adaptive method for faster backpropagation learning: the rprop algorithm. *International Conference on Neural Networks*, 1993.

David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

Ernest K. Ryu and Wotao Yin. *Large-Scale Convex Optimization: Algorithms & Analyses via Monotone Operators*. Cambridge University Press, 2022.

Yunlong Song, Sangbae Kim, and Davide Scaramuzza. Learning quadruped locomotion using differentiable simulation. *arXiv preprint arXiv:2403.14864*, 2024.

Henk J. van Waarde, Claudio De Persis, M. Kanat Camlibel, and Pietro Tesi. Willems' fundamental lemma for state-space systems and its extension to multiple datasets. *IEEE Control Systems Letters*, 4(3):602–607, 2020.

Henk Waarde, Jaap Eising, Harry Trentelman, and Kanat Camlibel. Data informativity: A new perspective on data-driven analysis and control. *IEEE Transactions on Automatic Control*, pages 1–1, 01 2020.

Nina Wiedemann, Valentin Wüest, Antonio Loquercio, Matthias Müller, Dario Floreano, and Davide Scaramuzza. Training efficient controllers via analytic policy gradient. *International Conference on Robotics and Automation (ICRA)*, 2023.

Anton Xue and Nikolai Matni. Data-driven system level synthesis. *Learning for Dynamics and Control*, 2021.

Riccardo Zuliani, Efe C Balta, and John Lygeros. Bp-mpc: Optimizing closed-loop performance of mpc using backpropagation. *arXiv preprint arXiv:2312.15521*, 2023.