DiffTune: Autotuning Through Autodifferentiation

Sheng Cheng, *Member, IEEE*, Minkyung Kim, *Graduate Student Member, IEEE*, Lin Song, Chengyu Yang, Yiquan Jin, Shenlong Wang, and Naira Hovakimyan, *Fellow, IEEE*

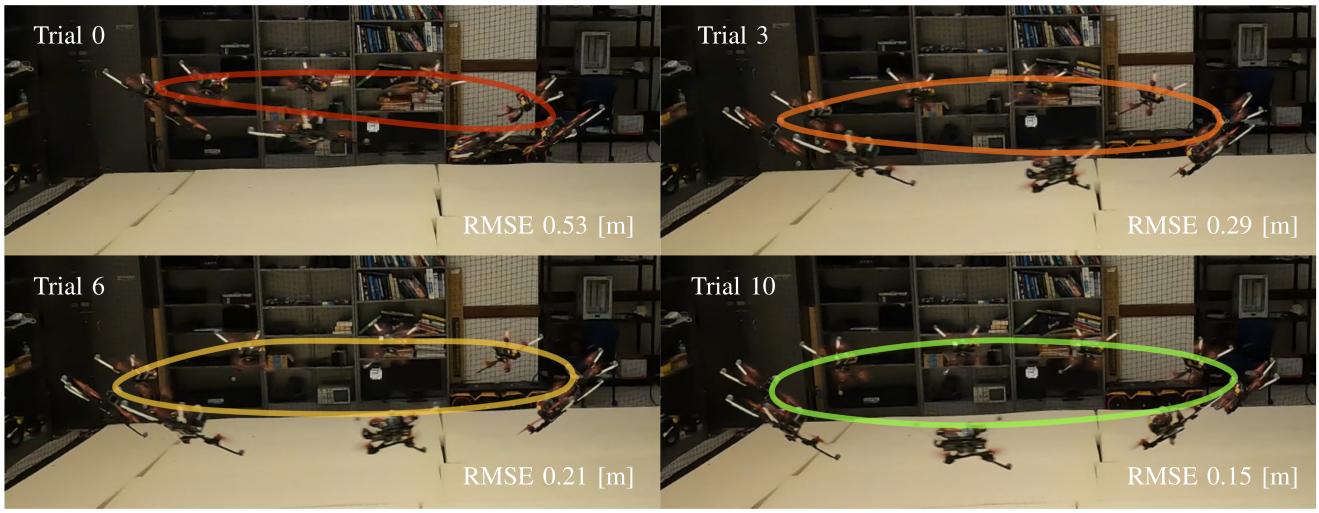


Fig. 1. Evolution of a quadrotor's tracking performance using DiffTune in ten trials [with 3.5× reduction on the root-mean-squared-error (RMSE)]. The quadrotor is commanded to track a 3 m/s circular trajectory (9 m/s^2 centripetal acceleration). The tuning is conducted over 12 parameters. Video recordings of the four trials are available in the supplementary material.

***Abstract*—The performance of robots in high-level tasks depends on the quality of their lower level controller, which requires fine-tuning. However, the intrinsically nonlinear dynamics and controllers make tuning a challenging task when it is done by hand. In this article, we present DiffTune, a novel, gradient-based automatic tuning framework. We formulate the controller tuning as a parameter optimization problem. Our method unrolls the dynamical system and controller as a computational graph and updates the controller parameters through gradient-based optimization. The gradient is obtained using sensitivity propagation, which is the only method for gradient computation when tuning for a physical system instead of its simulated counterpart. Furthermore, we use $\mathcal{L}_1$ adaptive control to compensate for the uncertainties (that unavoidably exist in a physical system) such that the gradient is not biased by the unmodeled uncertainties. We validate the DiffTune on a Dubin's car and a quadrotor in challenging simulation environments. In comparison with state-of-the-art autotuning methods, DiffTune achieves the best performance in a more efficient manner owing to its effective usage of the first-order information of the system. Experiments on tuning a nonlinear controller for quadrotor show promising results, where DiffTune achieves 3.5× tracking error reduction on an aggressive trajectory in only ten trials over a 12-D controller parameter space.**

***Index Terms*—Aerial systems: Mechanics and control, controller auto-tuning, learning and adaptive systems.**

Manuscript received 4 March 2024; revised 27 June 2024; accepted 8 July 2024. Date of publication 16 July 2024; date of current version 18 September 2024. This work was supported in part by NASA under the cooperative agreement 80NSSC20M0229, in part by National Science Foundation (NSF)-AoF Robust Intelligence under Grant 2133656, in part by NSF SLES under Grant 2331878, in part by Air Force Office of Scientific Research (AFOSR) under Grant FA9550-21-1-0411, in part by Amazon Research Award, and in part by Illinois-Insper Collaborative Research Fund. This article was recommended for publication by Associate Editor L. Righetti and Editor P. Robuffo Giordano upon evaluation of the reviewers' comments. *(Minkyung Kim and Lin Song contributed equally to this work.)*

Sheng Cheng, Minkyung Kim, Lin Song, Chengyu Yang, Yiquan Jin, and Naira Hovakimyan are with the Department of Mechanical Science and Engineering, University of Illinois Urbana-Champaign, Champaign, IL 61820 USA (e-mail: chengs@illinois.edu; mk58@illinois.edu; linsong2@illinois.edu; cy45@illinois.edu; yiquanj2@illinois.edu; nhovakim@illinois.edu).

Shenlong Wang is with the Department of Computer Science, University of Illinois Urbana-Champaign, Champaign, IL 61820 USA (e-mail: shenlong@illinois.edu).

Video: https://youtu.be/g42UxcIHUdg

Code: https://github.com/Sheng-Cheng/DiffTuneOpenSource

This article has supplementary downloadable material available at https://doi.org/10.1109/TRO.2024.3429191, provided by the authors.

Digital Object Identifier 10.1109/TRO.2024.3429191

I. Introduction

ROBOTIC systems are at the forefront of executing intricate tasks, relying on the prowess of their low-level controllers to deliver precise and agile motions. An optimal controller design starts with a meticulous analysis to ensure stability, followed by parameter tuning to achieve the intended performance on real-world robotic platforms. Traditionally, controller tuning is done either by hand using trial-and-error or proven methods for specific controllers (e.g., Ziegler–Nichols method for proportional-integral-derivative (PID) controller tuning [1]). Nevertheless, manual tuning often demands seasoned experts

and is inefficient, particularly for systems with lengthy loop times or extensive parameter space.

To improve efficiency and performance, automatic tuning (or autotuning) methods have been investigated. Such methods integrate system knowledge, expert experience, and software tools to determine the best set of controller parameters, especially for the widely used PID controllers [2], [3], [4]. Commercial autotuning products have been available since 1980s [3], [5]. A desirable autotuning scheme should have the following three qualities: i) stability of the target system; ii) compatibility with physical systems' data; and iii) efficiency for online deployment, possibly in real time. However, how to design the autotuning scheme, simultaneously having the above three qualities, for general controllers, is still a challenge.

Existing autotuning methods can be categorized into model based [6], [7] and model free [8], [9], [10], [11], [12], [13]. Both approaches iteratively select the next set of parameters for evaluation that is likely to improve the performance over the previous trials. Model-based autotuning methods leverage the knowledge of the system model to improve performance, often using the gradient of the performance criterion (e.g., tracking error) and applying gradient descent so that the performance can improve based on the local gradient information [6], [7]. Stability can be ensured by explicitly leveraging knowledge about the system dynamics. However, model-based autotuning might not work in a real environment, where the knowledge about the dynamical model might be imperfect. This issue is especially severe when controller parameters are tuned in simulation and then deployed to a physical system.

Model-free autotuning methods approximate gradient or a surrogate model to improve the performance. Representative approaches include Markov chain Monte Carlo (MCMC) [12], deep neural network (DNN) [13], and Bayesian optimization (BO) [8], [9], [10], [11], [14]. Such approaches often make no assumptions about the model and have the advantage of compatibility with physical systems' data owing to their data-driven nature. However, it is hard to establish stability guarantees with most of the data-driven methods (e.g., MCMC and DNN), where empirical methods are often applied. BO has the advantage of establishing stability/safety guarantees, but it can be inefficient when tuning in high-dimensional (e.g., $>$20) parameter spaces.

To overcome the challenges in the autotuning scheme, we present DiffTune: an autotuning method based on autodifferentiation (AD). Our method is inspired by the "end-to-end" idea from the machine learning community. Specifically, in the proposed scheme, the gradient of the loss function (evaluating the performance of the controller) with respect to the controller parameters can be directly obtained and then applied to gradient descent to improve the performance. DiffTune is generally applicable to tune all the controller parameters as long as the system dynamics and controller are differentiable (we will define "differentiable" in Section III), which is the case with most of the systems. For example, algebraically computed controllers, e.g., with the structure of gain-times-error (PID [7]), are differentiable. Moreover, following the seminal work [15] that differentiates the `argmin` operator using the Implicit Function Theorem, one can see that controllers relying on solutions of an optimization problem to generate control actions (e.g., model predictive control (MPC) [16], [17], optimal control [18], [19], safe controllers enabled by control barrier function [20], [21], [22], [23], linear-quadratic regulator (LQR) [24]) are also differentiable.

We build DiffTune by unrolling the dynamical system into a computational graph and then applying AD to compute the gradient. Since the structure of the dynamics and controller are untouched by the unrolling operation, the system is still interpretable, which is a distinctive feature compared to the NN-structured dynamics or controllers (widely applied in reinforcement learning). Furthermore, existing tools that support AD (e.g., PyTorch [25], TensorFlow [26], JAX [27], and CasADi [28]) can be conveniently applied for gradient computation.

However, when tuning physical systems, we need gradient information based on the data collected from such systems. In this scenario, the computational graph is broken because the states of the system are obtained from sensors rather than by evaluating the dynamics function. The broken graph forbids the usage of AD over the computational graph. We present an alternative way of gradient computation, called sensitivity propagation, which is based on the sensitivity equation [29] of a dynamical system. It propagates the sensitivity of the system state to the controller parameters in the forward direction in parallel to the dynamics' propagation. Finally, the gradient of the loss to controller parameters is simply a weighted sum of the sensitivities. Furthermore, uncertainties and disturbances exist in physical systems. If they are not dealt with, the resulting gradient based on the nominal dynamics (free of uncertainty and disturbance) will be biased. We propose to use $\mathcal{L}_1$ adaptive control [30] to compensate for the uncertainties and disturbances so that the physical system behaves similarly to the nominal system. The uncertainty compensation will preserve the gradient from being biased, thus resulting in more efficient tuning.

DiffTune enjoys the three earlier mentioned qualities simultaneously: *stability* is inherited from the controllers with stability guarantees by design; *compatibility with physical systems' data* is enabled by the sensitivity propagation; *efficiency* is provided since the sensitivity propagation runs forward in time and in parallel to the system's evolution. We have validated DiffTune in both simulations and experiments. DiffTune achieves smaller loss more efficiently than strong baseline autotuning methods AutoTune [12] and SafeOpt [9] (and its variant [31]) in simulations. Notably, in experiments, DiffTune achieves a 3.5$\times$ reduction in tracking error in only ten trials when tuning a nonlinear controller (12-D parameter space) of a quadrotor for tracking an aggressive trajectory, demonstrating the efficacy of the proposed approach.

Our contributions are summarized as follows: i) We propose an autotuning method for controller parameters over nonlinear dynamical systems and controllers in general forms by formulating the tuning problem as a parameter optimization problem. Only differentiability of the dynamics, controller, and loss function (for tuning) is required. ii) We treat the unrolled system as a computational graph, over which we use autodifferentiation to

compute the gradient efficiently. Specifically, we propose sensitivity propagation, which is compatible with data collected from a physical system and can be efficiently computed online. iii) We combine tuning with the $\mathcal{L}_1$ adaptive control to compensate for the model uncertainties in a physical system that can bias the computed gradient for tuning. iv) We validate the proposed approach in extensive simulations and experiments, where the compatibility with physical data, stability, and efficiency of DiffTune are demonstrated.

A previously published paper [32] has studied an extension of DiffTune for hyperparameter-free auto-tuning by optimizing the gradient update, with results validated only by simulated systems. In this article, our focus is to introduce DiffTune to solve the autotuning problem with a physical system, which has been validated through extensive experiments in Section VI. We show that simulation-based autotuning can lead to parameter overfitting to a particular simulated system, which does not apply to or even fail the physical system (details in Section VI-E). To handle the uncertainties that can bias the gradient used in the autotuning of a physical system, we use the $\mathcal{L}_1$ adaptive control for uncertainty compensation with the methodology introduced in Section IV-B and experimental results detailed in Section VI-D.

The rest of the article is organized as follows: Sections II and III review related work and background, respectively, of this article. Section IV describes our autotuning method and sensitivity propagation. We also discuss uncertainty handling when using physical systems' data. Section V shows the simulation results on a Dubins' car and on a quadrotor. Section VI demonstrates the experimental results on a quadrotor. Finally, Section VII concludes the article.

II. Related Work

Our approach closely relates to classical work on automatic parameter tuning and recent learning-based controllers. In this section, we briefly review previous work in the following directions.

Model-based autotuning leverages model knowledge to infer the parameter choice for performance improvement. In [6], an autotuning method is proposed for LQR. The gradient of a loss function with respect to the parameterized quadratic matrix coefficients is approximated using simultaneous perturbation stochastic approximation [33], which essentially computes the difference quotients at two random perturbation directions of the current parameter values. Simulations and experiments on an inverted pendulum platform demonstrate the effectiveness of this method. Kumar and Ramadge [7] applied autodifferentiation to tune a PID controller with input saturation, which is done by differentiating through the model and the feedback loop. Numerical examples are provided to show the efficacy of the proposed method on single-input-single-output systems. Romero et al. [34] proposed a probabilistic policy search method to efficiently tune a model predictive contouring control (MPCC) for quadrotor agile flight. The MPCC allows tradeoffs between progress maximization and path following in real time, albeit the dimension of parameters grows linearly to the number of gates on a race track. The search of the parameters is turned into maximizing a weighted likelihood function. The approach is validated in real-world scenarios, demonstrating superior performance compared to both manually tuned controllers and state-of-the-art autotuning baselines in aggressive quadrotor racing. In [35], actor-critic reinforcement learning (RL) is used to tune the parameters of a differentiable MPC for agile quadrotor tracking. The proposed approach enables short-term predictions and optimization of actions based on system dynamics while retaining the end-to-end training benefits and exploratory behavior of an RL agent. Zero-shot transfer to a real quadrotor is demonstrated on different high-level tasks. Giordano et al. [36], [37] proposed to use metrics on the sensitivity of a system's state to uncertain parameters to reduce the variance of the system's performance to uncertainties. The approach has been validated in Monte Carlo simulations to show the benefits of such optimization metrics.

Model-free autotuning relies on a zeroth-order approximate gradient or surrogate performance model to decide the new candidate parameters. In [38], the authors used extremum seeking to sinusoidally perturb the PID gains and then estimate the gradient. Gradient-free methods, e.g., Metropolis–Hastings (M-H) algorithm [39], have also been used for tuning. The M-H algorithm can produce a sequence of random samples from a desired distribution that cannot be directly accessed, whereas a score function is used instead to guide the sampling. In [12], the M-H algorithm was tailored to tuning the tracking MPC controller for high-speed quadrotor racing that demands minimum-time trajectory completion. In terms of surrogate models, machine learning tools have been frequently used for their advantages in incorporating data, which, in general, make no assumptions about the systems that produce the data. In [40], an end-to-end, data-driven hyperparameter tuning was applied to an MPC using a surrogate dynamical model. Such a method jointly optimizes the hyperparameters of system identification, task specification, and control synthesis. Simulation validation is conducted in the OpenAI Gym environment.

Gaussian process (GP) is often used as a nonparametric model that approximates an unknown function from input–output pairs with probabilistic confidence measures. This property makes GP a suitable surrogate model that approximates the performance function with respect to the tuned parameters. In [8], GP is applied to approximate the unknown cost function using noisy evaluations and then induce the probability distribution of the parameters that minimize the loss. The distribution is used to determine new parameters that can maximize the relative entropy, yielding information-efficient exploration of the parameter space. The proposed method is demonstrated to tune an LQR for balancing an inverted pole using a robot arm. In [9], the authors apply BO [41] to controller tuning (SafeOpt), which uses GP to approximate the cost map over controller parameters while constructing safe sets of parameters to ensure safe exploration. Safe sets of parameters are constructed while exploring new parameters such that the next evaluation point that can improve the current performance will also be safe. Quadrotor experiments are presented where the proposed method is used to tune a PD position controller on a single axis. Follow-up experiments [42] demonstrated the proposed autotuning scheme to quadrotor

control with nonlinear tracking controllers. However, one drawback of SafeOpt is that the search for the maximizer requires discretizing the parameters space, which scales poorly to the dimension of parameters. An improved version [31] applies particle swarm heuristics to perform adaptive discretization, which drastically reduces the computation time of SafeOpt to determine the maximizer. A more recent work [43] proposes a local BO-like optimizer called Gradient Information with BO (GIBO), which uses a GP for jointly inferring the objective function and its gradients with a probabilistic posterior. The queries of GIBO are chosen to minimize the uncertainty about the gradient. However, the assumption that the objective function is a sample from a known GP prior may not fit the autotuning scenario, especially when the system dynamics and the controller hold strong nonlinearities. Bayesian optimization has also been applied to gait optimization for bipedal walking, where GP is used to approximate the cost map of parameterized gaits [10], [11].

Learning for control is a recently trending research direction that strives to combine the advantages of model-driven control and data-driven learning for the safe operation of a robotic system. Exemplary approaches include, but not limited to, the following: reinforcement learning [44], [45], whose goal is to find an optimal policy while gathering data and knowledge of the system dynamics from interactions with the system; imitation learning [46], which aims to mimic the actions taken by a superior controller while making decisions using less information of the system than the superior controller; and iterative learning control [47], [48], which constructs the present control action by exploiting every possibility to incorporate past control and system information, typically for systems working in a repetitive mode. A recent survey [49] provided a thorough review of the safety aspect of learning for control in robotics.

Autodifferentiation is a technique that evaluates the partial derivative of a function specified by a computer program [50], [51]. Utilizing the inherent nature of computer programs, regardless of their complexity, autodifferentiation capitalizes on the execution of elementary arithmetic operations (such as addition, subtraction, multiplication, division, etc.) and elementary functions (such as exp, log, sin, cos, etc.). By iteratively applying the chain rule to these operations, it automatically computes partial derivatives of any desired order with high accuracy while incurring only a marginal increase in the number of arithmetic operations compared to the original program. Autodifferentiation has a significant advantage over other differentiation methods, e.g., manual differentiation (prone to errors and time-consuming), numerical differentiation like finite difference (poor scalability to high-dimensional inputs and proneness to round-off errors), or symbolic differentiation (suffering from overly complicated symbolic representations of the derivative, known as "expression swell"). A comprehensive comparison of these differentiation methods is provided in [51]. There are two modes of AD, reverse mode and forward mode, both relying on the chain rule to propagate the derivative. The reverse-mode AD requires a forward pass of the computational graph and keeps the values of the intermediate nodes in the memory. Subsequently, a backward pass propagates the partial derivatives from the output to the input of the graph. The forward-mode AD propagates the partial derivatives while conducting the forward pass on the graph, storing both the values and partial derivatives of the intermediate nodes in the memory.

III. Problem Formulation

Consider a discrete-time dynamical system

$$\boldsymbol{x}_{k+1} = f(\boldsymbol{x}_k, \boldsymbol{u}_k) \tag{1}$$

where $\boldsymbol{x}_k \in \mathbb{R}^n$ and $\boldsymbol{u}_k \in \mathbb{R}^m$ are the state and control, respectively, and the initial state $\boldsymbol{x}_0$ is known. The control is generated by a feedback controller that tracks a desired state $\bar{\boldsymbol{x}}_k \in \mathbb{R}^n$ such that

$$\boldsymbol{u}_k = h(\boldsymbol{x}_k, \bar{\boldsymbol{x}}_k, \boldsymbol{\theta}) \tag{2}$$

where $\boldsymbol{\theta} \in \mathbb{R}^p$ denotes the parameters of the controller, e.g., $\boldsymbol{\theta} \in \mathbb{R}^2$ may represent the P- and D-gain in a PD controller. We assume that the state $\boldsymbol{x}_k$ can be measured directly or, if not, an appropriate state estimator is used. Furthermore, we assume the dynamics (1) and controller (2) are differentiable, i.e., the Jacobians $\nabla_{\boldsymbol{x}} f$, $\nabla_{\boldsymbol{u}} f$, $\nabla_{\boldsymbol{x}} h$, and $\nabla_{\boldsymbol{\theta}} h$ exist, which widely applies to general systems.

The tuning task adjusts $\boldsymbol{\theta}$ to minimize an evaluation criterion, denoted by $L(\cdot)$, which is a differentiable function of the desired states $\bar{\boldsymbol{x}}$, actual states $\boldsymbol{x}$, and control actions $\boldsymbol{u}$ over a time interval of length N. An illustrative example is the tracking error plus control-effort penalty, where $L(\boldsymbol{x}_{0:N}, \bar{\boldsymbol{x}}_{0:N}, \boldsymbol{u}_{0:N-1}; \boldsymbol{\theta}) = \sum_{k=0}^{N} \|\boldsymbol{x}_k - \bar{\boldsymbol{x}}_k\|^2 + \sum_{k=0}^{N-1} \lambda \|\boldsymbol{u}_k\|^2$ with $\lambda > 0$ being the penalty coefficient. We will use the short-hand notation $L(\boldsymbol{\theta})$ for conciseness in the rest of the article.

With the setup introduced above, controller tuning can be formulated as a parameter optimization problem as follows:

$$\begin{aligned} \underset{\boldsymbol{\theta} \in \Theta}{\text{minimize}} \quad & L(\boldsymbol{\theta}) \\ \text{subject to} \quad & \boldsymbol{x}_{k+1} = f(\boldsymbol{x}_k, \boldsymbol{u}_k) \\ & \boldsymbol{u}_k = h(\boldsymbol{x}_k, \bar{\boldsymbol{x}}_k, \boldsymbol{\theta}) \\ & k \in \{0, 1, \ldots, N-1\}. \end{aligned} \tag{P}$$

Note that problem (P) searches for controller parameter $\boldsymbol{\theta}$ to minimize the loss L subject to the system's dynamics and a chosen controller (to be tuned). Problem (P) is generally nonconvex due to the nonlinearity in dynamics f and controller h. We will introduce our method, *DiffTune*, in Section IV for autotuning, especially for tuning a controller for a physical system.

IV. Method

We use a gradient-based method to solve problem (P) due to its nonconvexity, where the system performance is gradually improved by adjusting the controller parameters using gradient descent. We unroll the dynamical system (1) and controller (2) into a computational graph. Fig. 2 illustrates the unrolled system, which stacks the iterative procedure of state update via the "dynamics" and control-action generation via the "controller." The gradient $\nabla_{\boldsymbol{\theta}} L$ is then applied to update the parameters $\boldsymbol{\theta}$. Specifically, since the parameters are usually confined to a

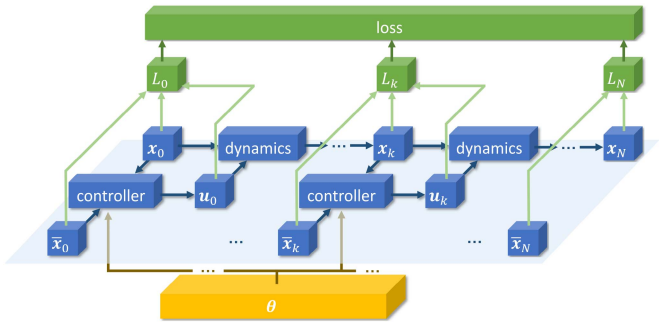


Fig. 2. Illustration of an unrolled dynamical system as a computational graph.

feasible set Θ, we use the projected gradient descent [52] to update $\boldsymbol{\theta}$:

$$\boldsymbol{\theta} \leftarrow \mathcal{P}_\Theta(\boldsymbol{\theta} - \alpha \nabla_{\boldsymbol{\theta}} L) \tag{3}$$

where $\mathcal{P}_\Theta$ is the projection operator that projects its operand into the set Θ, and α is the learning rate. The feasible set Θ is used here to ensure the stability of the system, where Θ can be determined via the Lyapunov analysis or empirically determined by engineering practice.

What remains to be done is to compute the gradient $\nabla_{\boldsymbol{\theta}} L$, for which AD can be used when the computational graph is complete (e.g., in simulations). AD can be conveniently implemented using off-the-shelf tools like PyTorch [25], TensorFlow [26], JAX [27], or CasADi [28]: one will program the computational graph using the dynamics and controller and set the parameter $\boldsymbol{\theta}$ with respect to which the loss function will be differentiated.

However, AD methods cannot incorporate data (state and control) from a physical system because AD relies on a complete computation graph, whereas the computational graph corresponding to a physical system is broken. Specifically, the dynamics (1) have to be evaluated each time to obtain a new state, which is not the case in a physical system: the states are obtained through sensor measurements or state estimation rather than evaluating the dynamics [see the comparison in Fig. 3(a) and (b)]. This explains why the computational graph is broken when considered for a physical system. Thus, AD can only be applied to autotuning in simulations, forbidding its usage with physical systems' data. We introduce sensitivity propagation next to address the compatibility with physical systems' data.

A. Sensitivity Propagation

We first break down the gradient $\nabla_{\boldsymbol{\theta}} L$ using chain rule

$$\nabla_{\boldsymbol{\theta}} L = \sum_{k=1}^{N} \frac{\partial L}{\partial \boldsymbol{x}_k} \frac{\partial \boldsymbol{x}_k}{\partial \boldsymbol{\theta}} + \sum_{k=0}^{N-1} \frac{\partial L}{\partial \boldsymbol{u}_k} \frac{\partial \boldsymbol{u}_k}{\partial \boldsymbol{\theta}}. \tag{4}$$

Since $\partial L/\partial \boldsymbol{x}_k$ and $\partial L/\partial \boldsymbol{u}_k$ can be determined once L is chosen, what remains to be done is to obtain $\partial \boldsymbol{x}_k/\partial \boldsymbol{\theta}$ and $\partial \boldsymbol{u}_k/\partial \boldsymbol{\theta}$. Given that the system states $\boldsymbol{x}_k$ are iteratively defined using the dynamics (1), we can derive an iterative formula for $\partial \boldsymbol{x}_k/\partial \boldsymbol{\theta}$ and $\partial \boldsymbol{u}_k/\partial \boldsymbol{\theta}$ by taking partial derivative with respect to $\boldsymbol{\theta}$ on

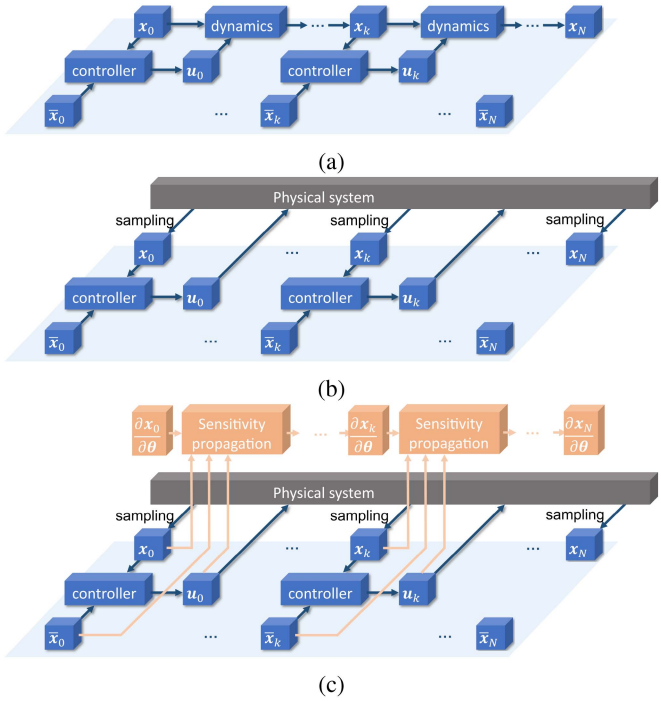


Fig. 3. Difference between a conceptual dynamical system (a) and a physical system (b). Since a physical system in (b) does not have a complete computational graph for autodifferentiation, sensitivity propagation is used for gradient computation and is illustrated in (c).

both sides of the dynamics (1) and controller (2):

$$\frac{\partial \boldsymbol{x}_{k+1}}{\partial \boldsymbol{\theta}} = (\nabla_{\boldsymbol{x}_k} f + \nabla_{\boldsymbol{u}_k} f \nabla_{\boldsymbol{x}_k} h) \frac{\partial \boldsymbol{x}_k}{\partial \boldsymbol{\theta}} + \nabla_{\boldsymbol{u}_k} f \nabla_{\boldsymbol{\theta}} h \tag{5a}$$

$$\frac{\partial \boldsymbol{u}_k}{\partial \boldsymbol{\theta}} = \nabla_{\boldsymbol{x}_k} h \frac{\partial \boldsymbol{x}_k}{\partial \boldsymbol{\theta}} + \nabla_{\boldsymbol{\theta}} h \tag{5b}$$

with $\partial \boldsymbol{x}_0/\partial \boldsymbol{\theta} = 0$. Note that (5a) is essentially the sensitivity equation of a system [29, Ch. 3.3]. We name the Jacobians $\partial \boldsymbol{x}_k/\partial \boldsymbol{\theta}$ and $\partial \boldsymbol{u}_k/\partial \boldsymbol{\theta}$ by sensitivity states.

The sensitivity propagation (5a) works by propagating the sensitivity state $\partial \boldsymbol{x}_k/\partial \boldsymbol{\theta}$ forward in time. In fact, (5a) is a time-varying linear system with the sensitivity state $\partial \boldsymbol{x}_k/\partial \boldsymbol{\theta}$. The system matrix $\nabla_{\boldsymbol{x}_k} f + \nabla_{\boldsymbol{u}_k} f \nabla_{\boldsymbol{x}_k} h$ and the excitation $\nabla_{\boldsymbol{u}_k} f \nabla_{\boldsymbol{\theta}} h$ are computed each time with the data sampled from the physical system. Specifically, the coefficients $\nabla_{\boldsymbol{x}_k} f$, $\nabla_{\boldsymbol{u}_k} f$, $\nabla_{\boldsymbol{x}_k} h$, and $\nabla_{\boldsymbol{\theta}} h$, whose formula are known since f and h are known, are evaluated at sampled state $\boldsymbol{x}_k$ and control $\boldsymbol{u}_k$. Once $\{\partial \boldsymbol{x}_k/\partial \boldsymbol{\theta}\}_{k=0:N}$ and $\{\partial \boldsymbol{u}_k/\partial \boldsymbol{\theta}\}_{k=0:N-1}$ are all computed, $\nabla_{\boldsymbol{\theta}} L$ can be computed as the weighted sum of the sensitivity states, where the weights $\{\partial L/\partial \boldsymbol{x}_k\}_{k=0:N}$ and $\{\partial L/\partial \boldsymbol{u}_k\}_{k=0:N-1}$ (whose formula are also known) are evaluated at the sampled data. An illustration of how sensitivity propagation works is shown in Fig. 3(c). Furthermore, sensitivity propagation permits online tuning. Since the formulas of $\nabla_{\boldsymbol{x}_k} f$, $\nabla_{\boldsymbol{u}_k} f$, $\nabla_{\boldsymbol{x}_k} h$, and $\nabla_{\boldsymbol{\theta}} h$ can be derived offline, the sensitivity propagation can update $\partial \boldsymbol{x}_{k+1}/\partial \boldsymbol{\theta}$ online whenever the system data $\boldsymbol{x}_k$ and $\boldsymbol{u}_k$ are sampled. Owing to the forward-in-time nature of sensitivity propagation, the horizon N can be adjusted online by need, which further contributes to the flexibility of

gradient computation to a varying horizon using sensitivity propagation. We summarize the DiffTune algorithm with sensitivity propagation in Algorithm 1.

Remark 1: The sensitivity propagation and forward-mode AD share the same formula as in (4) and (5). The difference lies in what type of data is applied. Two types of data are considered: The first type is from simulation, where $\boldsymbol{x}_k$ is obtained by the computation $\boldsymbol{x}_k = f(\boldsymbol{x}_{k-1}, \boldsymbol{u}_{k-1})$; the second type is sampled from a physical system, where $\boldsymbol{x}_k$ is obtained by either sensor measurements or state estimation, which cannot be represented as the evaluation a mathematical expression. In principle, forward-mode AD can only work with data of the first type, which limits its application to autotuning in simulations only. Sensitivity propagation, however, can work with *both* types of data. The most significant usage is associated with the second-type data, which provides a straightforward method of tuning a physical system. When applied to the first type of data, the sensitivity propagation is equivalent to the forward-mode AD.

Remark 2: One may notice that the sensitivity state $\partial \boldsymbol{x}_0/\partial \boldsymbol{\theta}$ is set to a zero matrix. This initialization relates to how the sensitivity state is interpreted. Suppose we have sampled the sequence of state $\{\boldsymbol{x}_k\}_{k=0:N}$ and control $\{\boldsymbol{u}_k\}_{k=0:N-1}$ subject to a certain parameter $\boldsymbol{\theta}$. Consider a small perturbation $\boldsymbol{\epsilon} \in \mathbb{R}^p$ to $\boldsymbol{\theta}$. The sensitivity states allow inferring about the state and control sequence subject to the parameter being $\boldsymbol{\theta} + \boldsymbol{\epsilon}$ using first-order approximation (without implementing the controller with the parameter $\boldsymbol{\theta} + \boldsymbol{\epsilon}$ and then sampling the data). Specifically, we have

$$\boldsymbol{x}_k(\boldsymbol{\theta} + \boldsymbol{\epsilon}) \approx \boldsymbol{x}_k(\boldsymbol{\theta}) + \frac{\partial \boldsymbol{x}_k}{\partial \boldsymbol{\theta}} \boldsymbol{\epsilon} \tag{6}$$

$$\boldsymbol{u}_k(\boldsymbol{\theta} + \boldsymbol{\epsilon}) \approx \boldsymbol{u}_k(\boldsymbol{\theta}) + \frac{\partial \boldsymbol{u}_k}{\partial \boldsymbol{\theta}} \boldsymbol{\epsilon}. \tag{7}$$

The sensitivity state $\partial \boldsymbol{x}_k/\partial \boldsymbol{\theta}$ is initialized at zero such that $\boldsymbol{x}_0(\boldsymbol{\theta} + \boldsymbol{\epsilon}) = \boldsymbol{x}_0(\boldsymbol{\theta})$ to ensure the same initial state despite parameter change. Therefore, how the state $\boldsymbol{x}_k(\boldsymbol{\theta} + \boldsymbol{\epsilon})$ will change subject to the $\boldsymbol{\epsilon}$ parameter perturbation can be inferred from the sensitivity $\partial \boldsymbol{x}_k/\partial \boldsymbol{\theta}$ which simply evolves with the sensitivity equation. Furthermore, the sensitivity states allow for autotuning without hyperparameters (e.g., learning rate α), which is detailed in [32].

Remark 3: Although AD cannot be applied to the entire computational graph when using data from a physical system, it can still be applied to obtain the Jacobians $\nabla_{\boldsymbol{x}_k} f$, $\nabla_{\boldsymbol{u}_k} f$, $\nabla_{\boldsymbol{x}_k} h$, and $\nabla_{\boldsymbol{\theta}} h$. Since the iterative structure in (5) remains the same among iterations, AD packages like PyTorch [25], TensorFlow [26], JAX [27], and CasADi [28] can be applied for evaluating these Jacobians.

The unique aspect of sensitivity propagation is its compatibility with data from a physical system. Using such data for tuning is vital because the ultimate goal is to improve the performance of a physical system instead of its simulated counterpart. Despite the fidelity of the model in simulation, the physical system will have discrepancies with the model, leading to suboptimal performance if the parameters come from simulation-based tuning. This phenomenon is part of the sim-to-real gap, which leads to degraded performance on physical systems compared to their simulated counterparts. The sensitivity propagation, unlike the forward- or reverse-mode AD, can still be applied to compute the gradient while using data collected from the physical system.

Algorithm 1: DiffTune.

Input: Initial state $\tilde{\boldsymbol{x}}_0$, initial parameter $\boldsymbol{\theta}_0$, feasible set Θ, horizon N, desired state $\{\bar{\boldsymbol{x}}_k\}_{k=0:N}$, step size α, formulas of the Jacobians $\{\nabla_{\boldsymbol{x}} f, \nabla_{\boldsymbol{u}} f, \nabla_{\boldsymbol{x}} h, \nabla_{\boldsymbol{\theta}} h\}$, and termination condition $\mathcal{C}$.
Output: Tuned parameter $\boldsymbol{\theta}^*$

```
Initialize θ ← θ_0.
while C is FALSE do
    Set x_0 ← x̃_0 and ∂x_0/∂θ ← 0.
    for k ← 0 to N do
        Obtain x_k from system and compute u_k using (2).
        Update ∂x_{k+1}/∂θ and ∂u_k/∂θ using (5).
        Compute ∂L/∂x_k and ∂L/∂u_k.
        Store ∂x_{k+1}/∂θ, ∂u_k/∂θ, ∂L/∂x_k and ∂L/∂u_k
        in memory.
    end for
    Compute ∇_θ L using (4) and update θ by (3).
end while
return the tuned parameter θ* ← θ.
```

B. Autotuning With Data From Physical Systems

The core of *DiffTune* is to obtain $\nabla_{\boldsymbol{\theta}} L$ from physical systems' data and then apply projected gradient descent.

However, model uncertainties and noise have to be carefully handled when using such data. Controller design usually uses the nominal model of the system, which is uncertainty- and noise-free. However, both uncertainties and noise exist in a physical system. If not dealt with, then the uncertainties and noise will contaminate the sensitivity propagation, leading to biased sensitivities and, thus, biased gradient $\nabla_{\boldsymbol{\theta}} L$, which results in inefficient parameter update. Since noise can be efficiently addressed by filtering or state estimation, our focus will be on handling the model uncertainties.

Existing methods that can compensate for the uncertainties can be applied to mitigate this issue. For example, the $\mathcal{L}_1$ adaptive control ($\mathcal{L}_1$AC) is a robust adaptive control architecture that has the advantage of decoupling estimation from control, thereby allowing for arbitrarily fast adaptation subject only to hardware limitations [30]. It can be augmented to the controller to be tuned such that the resulting system, even though suffering from model uncertainties, behaves like a nominal system by $\mathcal{L}_1$AC's compensation for the uncertainties. To proceed with the illustration of how $\mathcal{L}_1$AC works, we use continuous-time dynamics to stay consistent with the notation in the majority of the $\mathcal{L}_1$AC references [30], [53]. Consider the nominal system dynamics:

$$\dot{\boldsymbol{x}}^\star(t) = f(\boldsymbol{x}^\star(t), t) + B_{\mathrm{m}}(\boldsymbol{x}^\star(t), t)\boldsymbol{u}(\boldsymbol{x}^\star(t)),\ \boldsymbol{x}^\star(0) = \boldsymbol{x}_0 \tag{8}$$

where we use $\boldsymbol{x}^\star$ to denote the nominal state, $B_{\mathrm{m}} \in \mathbb{R}^{n\times m}$ to denote the control input matrix and $\boldsymbol{u}$ to denote the control input to the system. For example, in the tuning setup, $\boldsymbol{u}$ is chosen as the baseline control $\boldsymbol{u}_h$ from the to-be-tuned controller h in (2).

Consider the system in the presence of uncertainties:

$$\dot{\boldsymbol{x}}(t) = f(\boldsymbol{x}(t),t) + B_{\mathrm{m}}(\boldsymbol{x}(t),t)\left(\boldsymbol{u}(\boldsymbol{x}(t)) + \boldsymbol{\sigma}_{\mathrm{m}}(\boldsymbol{x}(t),t)\right) + B_{\mathrm{um}}\boldsymbol{\sigma}_{\mathrm{um}}(\boldsymbol{x}(t),t) \tag{9}$$

where $\boldsymbol{\sigma}_{\mathrm{m}} \in \mathbb{R}^m$ and $\boldsymbol{\sigma}_{\mathrm{um}} \in \mathbb{R}^{n-m}$ denote the matched and unmatched uncertainties, respectively, and $\boldsymbol{x}(0) = \boldsymbol{x}_0$. The matrix $B_{\mathrm{um}} \in \mathbb{R}^{n\times(n-m)}$ satisfies $B_{\mathrm{m}}^\top B_{\mathrm{um}} = 0$ and $\mathrm{rank}([B_{\mathrm{m}}\ B_{\mathrm{um}}]) = n$. The uncertainty $\boldsymbol{\sigma}$, defined as $\boldsymbol{\sigma}^\top := [\boldsymbol{\sigma}_{\mathrm{m}}^\top\ \boldsymbol{\sigma}_{\mathrm{um}}^\top]$, poses challenges to the sensitivity propagation because the mapping $\boldsymbol{x}(t) \mapsto \boldsymbol{\sigma}(\boldsymbol{x}(t),t)$ may not be explicitly known, leaving $\partial\boldsymbol{\sigma}/\partial\boldsymbol{x}$ uncomputable in the sensitivity propagation. We consider the following control design $\boldsymbol{u} = \boldsymbol{u}_{\mathrm{h}} + \boldsymbol{u}_{\mathrm{ad}}$, with $\boldsymbol{u}_{\mathrm{ad}}$ being the adaptive control, which results in the following system:

$$\dot{\boldsymbol{x}}(t) = f(\boldsymbol{x}(t),t) + B_{\mathrm{um}}(\boldsymbol{x}(t),t)\boldsymbol{\sigma}_{\mathrm{um}}(\boldsymbol{x}(t),t) + B_{\mathrm{m}}(\boldsymbol{x}(t),t)(\boldsymbol{u}_h(\boldsymbol{x}(t)) + \boldsymbol{u}_{\mathrm{ad}}(t) + \boldsymbol{\sigma}_{\mathrm{m}}(\boldsymbol{x}(t),t)). \tag{10}$$

The adaptive control $\boldsymbol{u}_{\mathrm{ad}}$ aims to cancel out the matched uncertainty $\boldsymbol{\sigma}_{\mathrm{m}}$, i.e., $\|\boldsymbol{\sigma}_{\mathrm{m}} + \boldsymbol{u}_{\mathrm{ad}}\| \approx 0$ (see [30], [54], [55], [56] for details of how $\mathcal{L}_1$AC is implemented). Specifically, $\mathcal{L}_1$AC estimates the uncertainty $\boldsymbol{\sigma}$ based on a state predictor and adaptation law. The state predictor propagates the state prediction $\hat{\boldsymbol{x}}$ based on the estimated uncertainty $\hat{\boldsymbol{\sigma}}$ and control inputs $\boldsymbol{u}_h$ and $\boldsymbol{u}_{\mathrm{ad}}$, i.e.,

$$\dot{\hat{\boldsymbol{x}}}(t) = f(\boldsymbol{x}(t),t) + B_{\mathrm{um}}(\boldsymbol{x}(t),t)\hat{\boldsymbol{\sigma}}_{\mathrm{um}}(t) + A_s(\hat{\boldsymbol{x}}(t) - \boldsymbol{x}(t)) + B_{\mathrm{m}}(\boldsymbol{x}(t),t)\left(\boldsymbol{u}_h(\boldsymbol{x}(t)) + \hat{\boldsymbol{\sigma}}_{\mathrm{m}}(t) + \boldsymbol{u}_{\mathrm{ad}}(t)\right) \tag{11}$$

where $A_s \in \mathbb{R}^{n\times n}$ is a Hurwitz matrix, and $\hat{\boldsymbol{x}}(0) = \boldsymbol{x}_0$. The error $\hat{\boldsymbol{x}}(t) - \boldsymbol{x}(t)$ between the predicted and actual states are used to compute the estimated uncertainty, where we use the piecewise-constant adaptation law [30]:

$$\hat{\boldsymbol{\sigma}}(t) = \begin{bmatrix}\hat{\boldsymbol{\sigma}}_{\mathrm{m}}(t)\\ \hat{\boldsymbol{\sigma}}_{\mathrm{um}}(t)\end{bmatrix} = -[B_{\mathrm{m}}\ B_{\mathrm{um}}]^{-1}\mathrm{expm}(A_sT_s - I)^{-1} A_s\mathrm{expm}(A_sT_s)(\hat{\boldsymbol{x}}(t) - \boldsymbol{x}(t)) \tag{12}$$

with T_s being the sample time of $\mathcal{L}_1$AC, $\mathrm{expm}(\cdot)$ denoting matrix exponential, and I being the identity matrix. The uncertainty's estimation error $\|\boldsymbol{\sigma} - \hat{\boldsymbol{\sigma}}\|$ is shown to be uniformly bounded under a set of mild regularity assumptions [57], [58]. Once the estimated uncertainty $\hat{\boldsymbol{\sigma}}$ is computed, the compensation $\boldsymbol{u}_{\mathrm{ad}}$ is obtained by low-pass filtering $\hat{\boldsymbol{\sigma}}$, i.e.,

$$\boldsymbol{u}_{\mathrm{ad}}(s) = C(s)\hat{\boldsymbol{\sigma}}_{\mathrm{m}}(s) \tag{13}$$

with s being the complex variable in the frequency domain, and $C(s)$ is the transfer function of the low-pass filter (LPF). The LPF is used here because the compensation is limited by the bandwidth of the actuator, where only the low-frequency components of $\hat{\boldsymbol{\sigma}}$ can be implemented by the actuator. It can be shown that the residual $\|\boldsymbol{\sigma}_{\mathrm{m}} + \boldsymbol{u}_{\mathrm{ad}}\|$ is bounded [53], [59], and the error norm $\|\boldsymbol{x}^\star - \boldsymbol{x}\|$ between the nominal state $\boldsymbol{x}^\star$ and the closed-loop state $\boldsymbol{x}$ in (10) is uniformly bounded both in transient and steady-state [58], [59], which renders the uncertain system (10) behaving similar to the nominal system (8). Therefore, the sensitivity propagation remains unchanged while $\mathcal{L}_1$AC handles the uncertainties. We will illustrate how the $\mathcal{L}_1$AC facilitates the autotuning of a physical system in Sections V and VI.

Remark 4: Note that $\boldsymbol{u}_{\mathrm{ad}}$ is not applied to the sensitivity propagation (only $\boldsymbol{u}_h$ is applied) because $\boldsymbol{u}_{\mathrm{ad}}$ is used to cancel out the uncertainty $\boldsymbol{\sigma}_{\mathrm{m}}$ to preserve the validity of the nominal dynamics (8).

C. Open-Source DiffTune Toolset

Our toolset `DiffTuneOpenSource` [60] is publicly available, which facilitates users' DiffTune applications in two ways. First, it enables the automatic generation of the partial derivatives required in sensitivity propagation. In this way, a user only needs to program the dynamics and controller, eliminating the need for additional programming of the partial derivatives. Second, we provide a template that allows users to quickly set up DiffTune for custom systems and controllers. The Dubin's car and quadrotor cases used in Section V are used as examples to illustrate the usage of the template.

V. Simulation Results

In this section, we implement *DiffTune* for a Dubin's car and a quadrotor in simulations, where the controller in each case is differentiable. For all simulations, we use `ode45` to obtain the system states by integrating the continuous-time dynamics (mimicking the continuous-time process on a physical system). The states are sampled at discrete-time steps. We use the sensitivity propagation to compute $\nabla_{\boldsymbol{\theta}} L$, where the discrete-time dynamics in (1) are obtained by forward-Euler discretization.

We intend to answer the following questions through the simulation study:

1) How does DiffTune compare to other autotuning methods? Since equipment wear is not an issue for tuning in simulations, we conduct sufficiently many trials to understand the asymptotic performance of autotuning methods for comparison.
2) How can the tuned parameters generalize to other unseen trajectories during tuning?
3) How does $\mathcal{L}_1$AC help tuning when the system has uncertainties? We show our main results in the following sections while supplying the details of configurations in Appendices A and B.

A. Dubin's Car

Dynamics, controller, and loss function: Consider the following nonlinear model:

$$\dot{x}(t) = v(t)\cos(\psi(t)),\ \dot{y}(t) = v(t)\sin(\psi(t)) \tag{14a}$$

$$\dot{\psi}(t) = \omega(t),\ \dot{v}(t) = F(t)/m,\ \dot{\omega}(t) = M(t)/J \tag{14b}$$

where the state contains five scalar variables (x, y, ψ, v, ω), which stand for horizontal position, vertical position, yaw angle, linear speed in the forward direction, and angular speed.

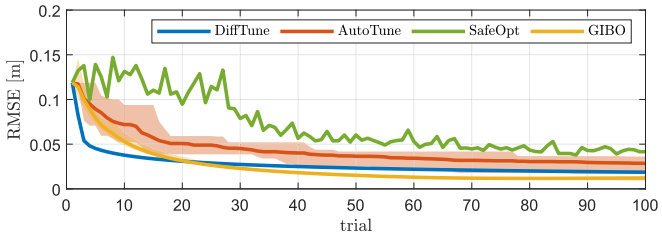


Fig. 4. Comparison of autotuning of Dubin's car using DiffTune, Auto-Tune [12], SafeOpt [9], and GIBO [43]. The shaded areas show the range of RMSEs (min to max) achieved in a total of 10 runs of algorithms that involve stochasticity (AutoTune and GIBO).

The control actions in this model include the force $F \in \mathbb{R}$ in the forward direction of the vehicle and the moment $M \in \mathbb{R}$. The vehicle's mass and moment of inertia (MoI) are known and denoted by m and J, respectively. The feedback tracking controller with tunable parameter $\boldsymbol{\theta} = (k_{\boldsymbol{p}}, k_{\boldsymbol{v}}, k_\psi, k_\omega)$ is given by

$$F(t) = m(k_{\boldsymbol{p}}\boldsymbol{e}_{\boldsymbol{p}}(t) + k_{\boldsymbol{v}}\boldsymbol{e}_{\boldsymbol{v}}(t) + \dot{\bar{\boldsymbol{v}}}(t))^\top \boldsymbol{q}(t) \tag{15a}$$

$$M(t) = J(k_\psi e_\psi(t) + k_\omega e_\omega(t) + \dot{\bar{\omega}}(t)) \tag{15b}$$

where $\bar{\cdot}$ indicates the desired value, the error terms are defined by $\boldsymbol{e}_{\boldsymbol{p}} = \bar{\boldsymbol{p}} - \boldsymbol{p}$, $\boldsymbol{e}_{\boldsymbol{v}} = \bar{\boldsymbol{v}} - \boldsymbol{v}$, $e_\psi = \bar{\psi} - \psi$, and $e_\omega = \bar{\omega} - \omega$ for $\boldsymbol{p}$ and $\boldsymbol{v}$ being the 2-D vector of position and velocity, respectively, $\boldsymbol{q} = [\cos(\psi)\ \ \sin(\psi)]^\top$ being the heading of the vehicle, $\bar{\boldsymbol{v}} = [\bar{v}\cos(\bar{\phi})\ \bar{v}\sin(\bar{\phi})]^\top$ and $\dot{\bar{\boldsymbol{v}}}$ being the desired linear velocity and acceleration, respectively. The control law (15) is a PD controller with proportional gains $(k_{\boldsymbol{p}}, k_\psi)$ and derivative gains $(k_{\boldsymbol{v}}, k_\omega)$. If $\boldsymbol{\theta} > 0$, then this controller is exponentially stable for the tracking errors $(\|\boldsymbol{e}_{\boldsymbol{p}}\|, \|\boldsymbol{e}_{\boldsymbol{v}}\|, \|e_\psi\|, \|e_\omega\|)$. We set the loss function as the RMSE of the position tracking error.

Comparison to other methods: We compare DiffTune with strong baseline autotuning methods: Auto-Tune [12], SafeOpt [9], and GIBO [43]. Note that these baseline methods are model-free probabilistic approaches that do not require knowledge of system dynamics and control. We compare the tuning performance on a circular trajectory and assign 100 trials in each method. Other details of implementation are available in Appendix A. The results are shown in Fig. 4. The final tracking errors obtained by all methods are similar, and all are below 0.05 m. DiffTune achieves the fastest error reduction in the first 20 trials due to its efficient usage of the system's first-order information that can effectively guide the parameter search. After 20 trials, GIBO achieves the minimum error, indicating that the GP model has captured the mapping from the control parameters to the objective function. The performance of AutoTune and SafeOpt is inferior to that of DiffTune and GIBO, with a slower error reduction and larger errors at the end of 100 trials. However, GIBO and AutoTune are limited to autotuning in simulations because they require randomly sampled parameters for controller implementation and rollouts for performance evaluation, and then decide how to pick the next candidate parameter. This procedure leads to huge mechanical wear and tear when applied to autotuning of a physical system. SafeOpt can produce acceptable performance by the end of the 100 trials, albeit the RMSE reduction is not smooth. The performance of SafeOpt relies on both prior knowledge (including the kernel function and its parameters and the range of feasible parameters) and the parameter space's discretization (for searching maximizers), both of which are difficult to tune (as hyperparameters in autotuning).

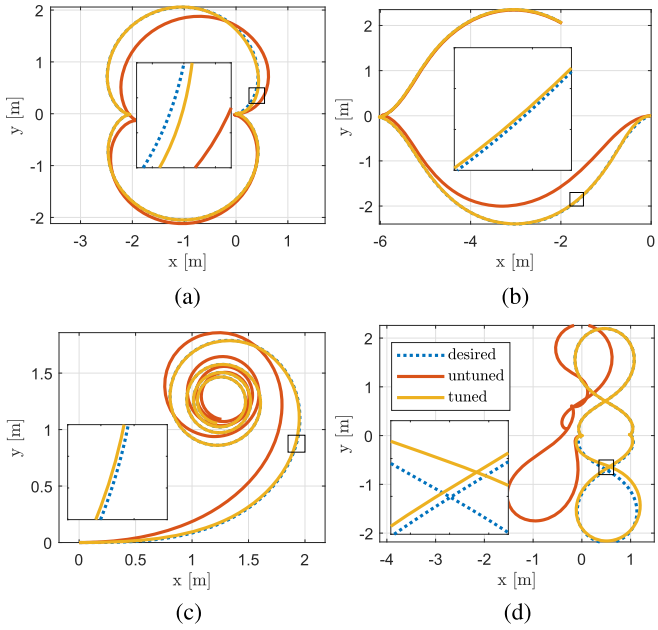


Fig. 5. Comparison of performance between tuned and untuned controller parameters on testing trajectories for generalization. (a) Peanut. (b) Lemon. (c) Spiral. (d) Twist.

Generalization: We illustrate the generalization of *DiffTune* in a batch tuning example. We select nine trajectories (shown in Fig. 13 in Appendix A) as the batch tuning set. These trajectories are generated by composing constant, sinusoidal, and cosinusoidal signals for the desired linear and angular velocities. The maximum linear speed and angular speed are set to 1 m/s and 1 rd/s, respectively, to represent trajectories in one operating region. The four control parameters are all initialized at 2. The tuning proceeds by batch gradient descent on the tuning set. The controller parameters converge to $(k_{\boldsymbol{p}}, k_{\boldsymbol{v}}, k_\psi, k_\omega) = (18.83, 6.69, 14.97, 2.66)$. We then test the tuned parameters on four testing trajectories (unseen in the tuning set) with lemon-, twist-, peanut-, and spiral-shape, as shown in Fig. 5. The tuned parameters lead to better tracking performance than the untuned ones. The loss on the testing set is compared to the untuned parameters in Table I. It can be observed that the tuned parameters generalize well and are robust to the previously unseen trajectories.

Handling uncertainties: In this simulation, we implement the $\mathcal{L}_1$AC to facilitate the compensation for the uncertainties during tuning. For the $\mathcal{L}_1$AC, we use the piecewise-constant adaptation law and a first-order low-pass filter with 20 rd/s bandwidth. In this simulation, we inject additive force $0.1a_1\sin(t)$ and moment $0.1a_2\cos(t)$ to the control channels in the dynamics (14) as

TABLE I
TESTING TRAJECTORIES FOR DUBIN'S CAR SIMULATION AND ASSOCIATED LOSSES

Trajectory	max speed		loss	
	linear	angular	w/ *DiffTune*	w/o *DiffTune*
peanut	2	1.3	**0.18**	63.02
lemon	2	1	**0.07**	55.18
spiral	1	5	**0.39**	21.11
twist	2	3.4	**2.05**	974.57

The average loss of the tuned parameters on the tuning set is 0.36. (The units associated with linear and angular maximum speeds are m/s and rad/s, respectively.)

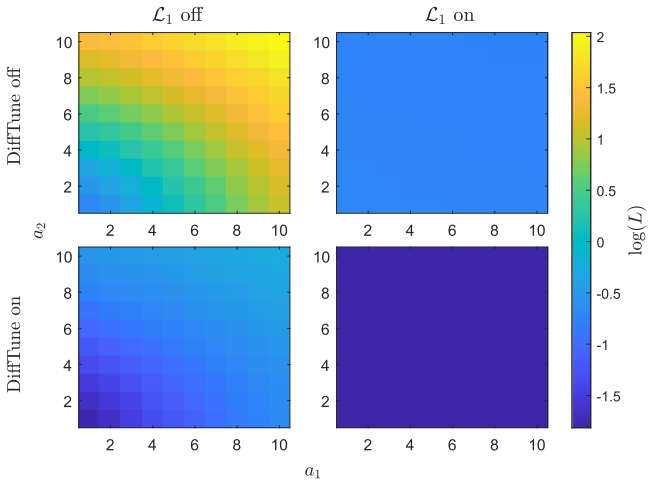


Fig. 6. Loss L subject to uncertainties (additive force $0.1a_1 \sin(t)$ and moment $0.1a_2 \cos(t)$ with a_1 and a_2 taking values from 1 to 10) in the ablation study of *DiffTune* and $\mathcal{L}_1$AC.

uncertainties from the environment. To understand how the performance is impacted by the uncertainties, we set (a_1, a_2) to a 10×10 grid such that a_1 and a_2 take integer values from 1 to 10, representing gradually intensified uncertainties. The four control parameters are all initialized at 10. We tune the controller parameters with both $\mathcal{L}_1$ ON and $\mathcal{L}_1$ OFF, where the sensitivity propagation in both cases is based on the nominal model in (14). Different from the generalization test, we only tune the parameters on one trajectory (the focus is on how to reduce the impact of the uncertainties that are not considered in the nominal dynamics). The step size and termination criterion remain the same as before. To clearly understand the individual role of *DiffTune* and the $\mathcal{L}_1$AC in tuning, we conduct an ablation study. The losses are shown in Fig. 6. It can be observed that both *DiffTune* and $\mathcal{L}_1$AC improve the performance, and a combination of both achieves the best overall performance: the $\mathcal{L}_1$AC does so by compensating for the uncertainties, whereas *DiffTune* does so by driving the parameters to achieve smaller tracking error. Although the two heatmaps with $\mathcal{L}_1$ on show indistinguishable colors within each itself, the actual loss values have minor fluctuations.

B. Quadrotor

Dynamics, controller, and loss function: Consider the following model on SE(3):

$$\dot{\boldsymbol{p}} = \boldsymbol{v}, \qquad \dot{\boldsymbol{v}} = g\boldsymbol{e}_3 - \frac{f}{m} R\boldsymbol{e}_3 \tag{16a}$$

$$\dot{R} = R\boldsymbol{\Omega}^{\times}, \qquad \dot{\boldsymbol{\Omega}} = J^{-1}(\boldsymbol{M} - \boldsymbol{\Omega} \times J\boldsymbol{\Omega}) \tag{16b}$$

where $\boldsymbol{p} \in \mathbb{R}^3$ and $\boldsymbol{v} \in \mathbb{R}^3$ are the position and velocity of the quadrotor, respectively, $R \in SO(3)$ is the rotation matrix describing the quadrotor's attitude, $\boldsymbol{\Omega} \in \mathbb{R}^3$ is the angular velocity, g is the gravitational acceleration, m is the vehicle mass, $J \in \mathbb{R}^{3\times 3}$ is the MoI matrix, f is the collective thrust, and $\boldsymbol{M} \in \mathbb{R}^3$ is the moment applied to the vehicle. The *wedge* operator $\cdot^{\times} : \mathbb{R}^3 \to \mathfrak{so}(3)$ denotes the mapping to the space of skew-symmetric matrices. The control actions f and $\boldsymbol{M}$ are computed using the geometric controller [61]. The geometric controller has a 12-D parameter space, which splits into four groups of parameters: $\boldsymbol{k_p}, \boldsymbol{k_v}, \boldsymbol{k}_R, \boldsymbol{k_\Omega}$ (applying to the tracking errors in position, linear velocity, attitude, and angular velocity, respectively). Each group is a 3-D vector (associated with the x-, y-, and z-component in each's corresponding tracking error). The initial parameters for tuning are set as $\boldsymbol{k_p} = 16\mathbb{I}, \boldsymbol{k_v} = 5.6\mathbb{I}$, $\boldsymbol{k}_R = 8.81\mathbb{I}$, and $\boldsymbol{k_\Omega} = 2.54\mathbb{I}$, for $\mathbb{I} = [1, 1, 1]^\top$. The feasible sets of controller parameters are set as $\boldsymbol{k_p} \in [15, 24]$, $\boldsymbol{k_v} \in [4, 16]$, $\boldsymbol{k_R} \in [8, 12]$, and $\boldsymbol{k_\Omega} \in [0.1, 3]$. We set the loss function as the squared norm of the position tracking error, summed over a horizon of 10 s. We add zero-mean Gaussian noise to the position, linear velocity, and angular velocity (with standard deviation 0.1 m, 0.1 m/s, 1e-3 rd/s, respectively).

Comparison to other methods: We compare DiffTune with strong baselines AutoTune [12], SafeOpt-PSO [31], and GIBO [43]. Note that these baseline methods are model-free probabilistic approaches that do not require knowledge of system dynamics and control. The middle is a variant of the SafeOpt [9], which applies particle swarm optimization (PSO) to enable adaptive discretization of the parameter space. The original SafeOpt is not applicable because it requires fine discretization of the parameter space to search for the maximizer, which suffers from the curse of dimensionality. Specifically, autotuning of the geometric controller requires at least 12^M discretization points if each parameter admits at least M discretization points.

The detailed settings of AutoTune and SafeOpt in this example are shown in Appendix B. We compare the three autotuning methods on three trajectories, where 100 trials are performed for each method on each trajectory. The results are shown in Fig. 7, where DiffTune achieves the minimum tracking RMSE with the best efficiency. Note that the autotuning of the quadrotor is more complicated than that of Dubin's car due to the former's higher dimensional parameter space and stronger nonlinearities in dynamics and control. In the autotuning on the 2-D/3-D circular trajectories, the RMSEs show oscillation near the end of the tuning trials, indicating the learning rate might be too large when the loss is close to a (local) minimum. In terms of the final RMSE, AutoTune, SafeOpt-PSO, and GIBO demonstrate similar performance, all inferior to that of DiffTune. AutoTune and GIBO have a smoother RMSE reduction than SafeOpt-PSO.

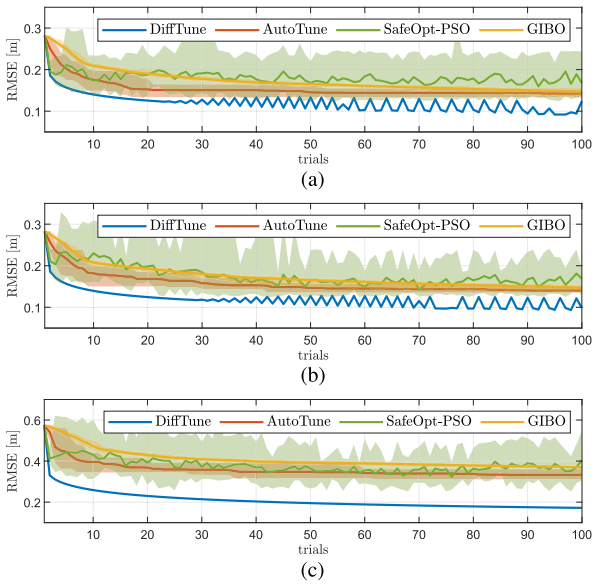


Fig. 7. Comparison of tuning the controller of a quadrotor using DiffTune (proposed), AutoTune [12], SafeOpt-PSO [31], and GIBO [43]. The shaded areas show the range of RMSEs (min to max) achieved in a total of ten runs of the algorithms that involve stochasticity (AutoTune, SafeOpt-PSO, and GIBO). (a) 2-D circle. (b) 3-D circle. (c) 3-D lemniscate.

TABLE II
TRACKING RMSE [CM] WITH MOI PERTURBATION β RANGING FROM 0.5 TO 4 IN THE ABLATION STUDY OF DIFFTUNE AND $\mathcal{L}_1$AC

β		0.5	1	1.5	2	2.5	3	3.5	4
DT	$\mathcal{L}_1$	**30.5**	**30.1**	**30.8**	**32.9**	**34.3**	**35.4**	**36.5**	**37.9**
DT	~~$\mathcal{L}_1$~~	30.8	**30.1**	31.0	33.9	37.8	39.5	40.5	41.1
~~DT~~	$\mathcal{L}_1$	57.9	57.5	58.0	59.5	62.2	66.0	70.9	76.8
~~DT~~	~~$\mathcal{L}_1$~~	57.8	57.6	58.9	62.2	67.3	74.1	82.2	90.9

DT stands for "DiffTune."

Moreover, the three baseline methods are less favorable for practical usage since they demand more hyperparameters to be tuned (e.g., the variance of the transition model for each parameter in AutoTune; kernel functions, lower/upper bound of each parameter, safety thresholds, and swarm size for SafeOpt-PSO; kernel functions, local search bounds, learning rate, and number of queries for GIBO) than the number of controller parameters for autotuning. In contrast, DiffTune only requires tuning the learning rate as the sole hyperparameter, yet still delivering the best outcome.

Handling uncertainties: In this simulation, we consider the uncertainty caused by the imprecise knowledge of the MoI J. We set the vehicle's true MoI as βJ for β from 0.5 to 4 and use J in the controller design as our best knowledge of the system. The scaled MoI can be treated as an unknown control input gain [see (16b)], leading to decreased ($\beta > 1$) or increased ($\beta < 1$) moment in reality compared to the commanded moment by the geometric controller. However, the uncertainty caused by the perturbed MoI can be well handled by $\mathcal{L}_1$AC, which is adopted in the simulation (formulation detailed in [54]). We conduct an ablation study to understand the roles of *DiffTune* and $\mathcal{L}_1$AC by comparing the root-mean-square error (RMSE) of position tracking, as shown in Table II (where the tuning is conducted over a 3-D lemniscate trajectory in 100 trials with a learning rate of $\alpha = 0.005$). It can be seen that tuning and $\mathcal{L}_1$ can individually reduce the tracking RMSE. The best performance is achieved when DiffTune and $\mathcal{L}_1$AC are applied jointly.

C. Discussion

The advantage of DiffTune is its efficient usage of the first-order information (gradient $\nabla_{\boldsymbol{\theta}} L$) of the target system. Compared with DiffTune, the baseline autotuning methods (AutoTune, SafeOpt/SafeOpt-PSO, and GIBO) have the advantage of requiring less prior information [e.g., explicit formulas for dynamics (1) and controller (2)] than DiffTune. All baseline methods use probabilistic approaches (Metropolis–Hastings algorithm or Bayesian optimization) to explore candidates of parameters and iteratively improve the performance based on observed input–output (i.e., parameters-performance) pairs. However, when the knowledge of the system is perceived through such "sampling" procedures, sufficiently many trials are needed to gain enough information and infer the optimal parameter choice, and the number of trials scales badly with the dimension of the parameter space. However, in practice, many physical systems have models obtained by physics or first principles, which can provide sufficiently useful first-order information to guide parameter searches. Such information will significantly reduce the number of trials in autotuning compared to when one uses "sampling" to obtain this information, which is clear in the comparison shown in Figs. 4 and 7. For BO-based approaches (SafeOpt/SafeOpt-PSO and GIBO), the assumption that the objective function is a sample from a known GP prior may not fit the autotuning scenario, especially when the system dynamics or controller hold strong nonlinearities. This conclusion is drawn from the observation of BO-based approaches' *inferior* performance to DiffTune for quadrotor autotuning in Fig. 7, in contrast to the *similar* performance obtained by DiffTune and BO-based approaches for the autotuning of Dubin's car (whose dynamics and controller exhibit less nonlinearities than those of the quadrotor) in Fig. 4. We will illustrate the efficiency of autotuning using the first-order information with the experimental results in Section VI next.

VI. EXPERIMENT RESULTS

We validate and evaluate DiffTune on a quadrotor in experiments, through which we would like to answer the following four questions:

1) How is the performance improvement using DiffTune with only limited tuning budgets (e.g., ten trials)?
2) How do the tuned parameters generalize to trajectories that are unseen during tuning?
3) What are the individual role of DiffTune and $\mathcal{L}_1$AC in terms of performance improvement?
4) How is the real-flight performance compared between parameters autotuned with experimental data and those autotuned with simulation data?

A. Experiment Setup

We use the same dynamics and controller as used in Section V-B. The controller's initial parameters are shown in

TABLE III
TRAJECTORIES USED IN THE GENERALIZABILITY STUDY

Trajectory	Notation	Function
Circular	$C(v)$	$x(t) = \sin(vt)$ $y(t) = -\cos(vt)$ $z(t) = -1$
3D figure 8	$DF(v)$	$x(t) = 1.5\sin(vt/1.75)$ $y(t) = \sin(2vt/1.75)$ $z(t) = -1 - 0.3\sin(0.5vt/1.75)$
Vertical figure 8	$VF(v)$	$x(t) = 1.5\sin(vt/1.25)$ $y(t) = \sin(vt/1.25)$ $z(t) = -1 - 0.3\sin(2vt/1.25)$

The trajectories are parameterized by v, and their xyz-coordinates are shown in the "Function" column (t is the time argument).

TABLE IV
TRACKING RMSE [CM] IN THE GENERALIZABILITY TEST

Traj.	Spd. [m/s]	Acc. [m/s^2]	P1 [cm]	P2 [cm]	P3 [cm]	HT [cm]
$C(1)$	1	1	**5.7**	**5.7**	5.5*	9.0
$C(2)$	2	4	27.2	**10.7**	16.6	18.0
$C(3)$	3	9	72.8	31.7	**15.4**	28.6
$DF(1)$	0.59–1.43	0.00–1.35	5.5	**4.9**	4.8*	5.3
$DF(2)$	1.17–2.86	0.00–5.41	10.6	8.8	**8.3**	9.2
$VF(1)$	0.48–1.52	0.00–1.20	12.5	10.5	9.7*	**9.6**
$VF(2)$	0.96–3.04	0.00–4.81	17.8	13.0	**9.8**	12.7

The star indicates minor oscillations on the pitch angle of the quadrotor. "HT" stands for hand-tuned.

TABLE V
TRACKING RMSE [CM] IN ABLATION STUDY WITH DIFFTUNE AND $\mathcal{L}_1$AC

Speed [m/s]		1		2		3	
$\mathcal{L}_1$AC		off	on	off	on	off	on
DiffTune	off	8.9	7.5	27.5	25.1	61.8	46.6
	on	5.7	**3.0**	10.7	**6.9**	17.7	**16.2**

TABLE VI
TRACKING RMSE (UNIT: [CM]) OF THE REAL QUADROTOR USING SIMULATION-BASED AUTOTUNING PARAMETERS (SIM_X), EXPERIMENT-BASED AUTOTUNING PARAMETERS (EXPT), AND HAND-TUNED PARAMETERS (HT)

Traj.	Sim_{10}	Sim_{30}	Sim_{50}	ExpT	HT
$C(1)$	7.3	6.6	**4.3**	5.7	9.0
$C(2)$	18.2	16.2	crash	**10.7**	18.0
$C(3)$	32.3	crash	crash	**15.4**	28.6

The subscript X in Sim_X indicates the number of trials in the simulation auto-tuning. For ExpT, parameters P1, P2, and P3 from Table VII are applied to trajectories $C(1)$, $C(2)$, and $C(3)$, respectively. The hand-tuned parameters can be found in Table VII.

Table VII. We only permit ten tuning trials as the budget to limit the time and mechanical wear of the tuning. The loss function is chosen as the sum of the translational and rotational tracking errors to penalize undesirable tracking performance in these two perspectives, i.e., $L = \|\boldsymbol{p} - \bar{\boldsymbol{p}}\|^2 + \text{tr}(I - \bar{R}^\top R)/2$. A horizon of 7 s is used to collect the data and perform sensitivity propagation. The data contain the full state and control actions sampled at 400 Hz (by the design of Ardupilot), where the state is obtained via the original EKF developed by Ardupilot, with Vicon providing only position and yaw measurements of the quadrotor. The data are logged onboard and downloaded to a laptop to compute the new controller parameter $\boldsymbol{\theta}$. We use learning rate $\alpha = 0.1$ together with gradient clipping such that the parameters in the next trial $\boldsymbol{\theta}_{j+1}$ will always fall within 10% of the current parameters $\boldsymbol{\theta}_j$, i.e., $\boldsymbol{\theta}_{j+1} \in [0.9\boldsymbol{\theta}_j, 1.1\boldsymbol{\theta}_j]$. Such a saturation scheme is used to i) prevent the parameters from turning negative when the gradient is large and ii) enforce a "trust region" around the current parameters to avoid overly large parameter changes. The laptop has an Intel i9-8950HK CPU, and the run time for sensitivity propagation to update the sensitivity states in one iteration (from k to $k+1$) is $91 \pm 13\,\mu$s (in MATLAB).

B. Run DiffTune on Three Trajectories

We use a circular trajectory for tuning, where the speeds are set to [1, 2, 3] m/s for a spectrum of agility from slow to aggressive. The controller is tuned individually for these three speeds,[1] and we denote the final parameters by P1, P2, and P3 (associated with the speeds of 1, 2, and 3 m/s, respectively). The parameters P2 are obtained in seven trials because the quadrotor experiences oscillations after the seventh trial, and we decided to use the parameters tuned at the last nonoscillating trial. The reduction of the tracking RMSE is shown in Fig. 8. Comparing the tracking performance at the last trial to the initial trial, the RMSE has achieved 1.5×, 2.5×, and 3.5× reduction on the 1, 2, and 3 m/s circular trajectories, respectively. Furthermore, all the tuned parameters achieve lower tracking RMSEs than the hand-tuned parameters. While the reduction in tracking RMSE is monotone in the 2 and 3 m/s cases, for the case of 1 m/s, a minor fluctuation is superposed on the monotone reduction of the tracking RMSE. This phenomenon happens since the dominant z-axis RMSE fluctuates, which is caused by the large learning rate for z-axis tracking when the parameters are close to the (local) minimum of z-axis error (observe the z-axis RMSE reduces only from 6.8 to 5.5 cm). For the x- and y-axis tracking RMSE, we observe a monotone reduction in all three speeds. Notably, we observe the sensitivity of the angular velocity gains $\boldsymbol{k}_{\boldsymbol{\Omega}_x}$ and $\boldsymbol{k}_{\boldsymbol{\Omega}_y}$ are larger than the other parameters (by at least a magnitude). In other words, the partial derivatives $\partial L/\partial \boldsymbol{k}_{\boldsymbol{\Omega}_x}$ and $\partial L/\partial \boldsymbol{k}_{\boldsymbol{\Omega}_y}$ are large, which leads to significant changes in the gains $\boldsymbol{k}_{\boldsymbol{\Omega}_x}$ and $\boldsymbol{k}_{\boldsymbol{\Omega}_y}$. This reduction leads to a more agile response in rotational tracking on the roll and pitch commands, which efficiently improves the position tracking performance on the x- and y-axis. The trajectories on the horizontal plane through the tuning trials are shown in Fig. 9. For the 3 m/s circular tuning trajectory, we show the stacked images during the flight in Fig. 1. It is clear that the quadrotor's tracking of the circular trajectory becomes better as more trials are conducted.

[1]The video recordings of the zeroth, third, sixth, and tenth trials while tuning for the 3 m/s circular trajectory are available in the supplementary material, in which one can see the performance improvement through the trials.

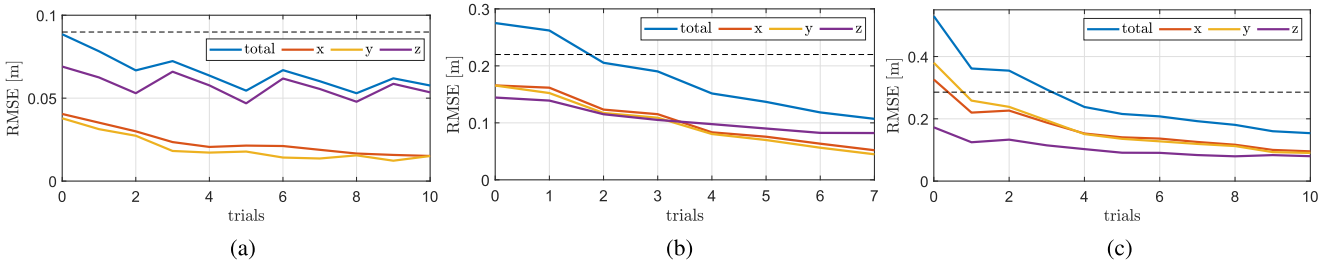


Fig. 8. Tracking error (RMSE) of the tuning on three circular trajectories. The dashed line shows the RMSE achieved by hand-tuning. (a) 1 m/s. (b) 2 m/s. (c) 3 m/s.

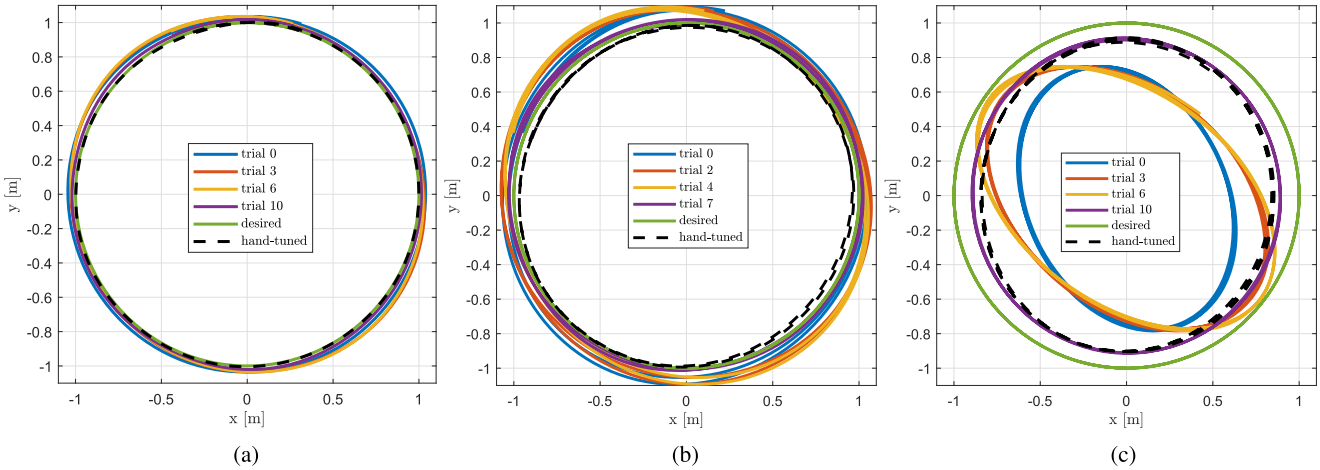


Fig. 9. Horizontal trajectories of the quadrotor during tuning on three circular trajectories with different speeds. (a) 1 m/s. (b) 2 m/s. (c) 3 m/s.

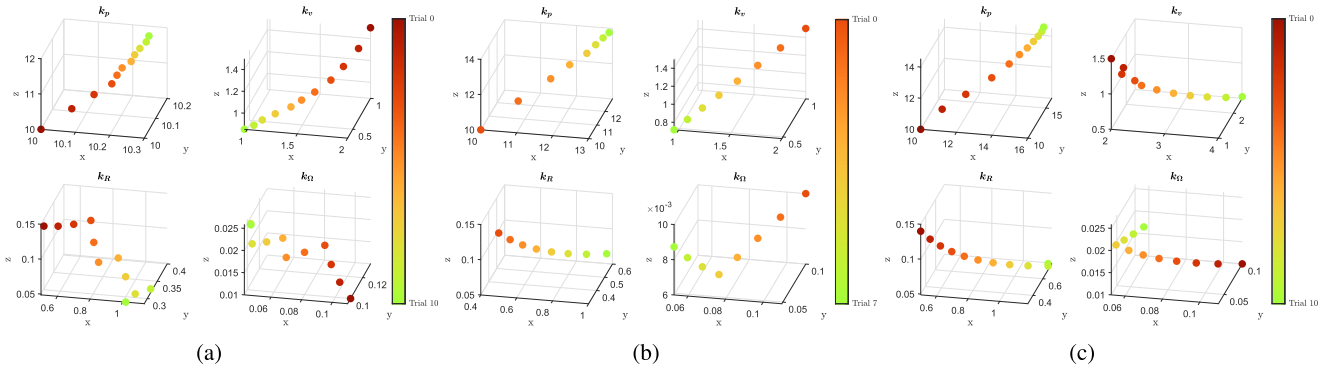


Fig. 10. History of parameters during the tuning on three circular trajectories with different speeds. (a) 1 m/s. (b) 2 m/s. (c) 3 m/s.

The evolution of the parameters in the tuning trials is shown in Fig. 10. Overall, the parameters tend to converge following one direction, except for the derivative gain $\boldsymbol{k_\Omega}$ for angular tracking. Specifically, near the end of the tuning, $\boldsymbol{k_\Omega}$ shows a change of evolution direction, which indicates that the mapping from these parameters to the loss function is likely to lie on a nonlinear manifold. Such a nonlinear manifold is difficult for a human to perceive and understand in hand-tuning unless sufficiently many, possibly pessimistically unrealistically many, trials are provided, which leads to the challenges in tuning a nonlinear controller by hand. Furthermore, another challenge of hand tuning is that one may alter one or (at most) two parameters in each trial since humans essentially perform coordinatewise finite differences via trial and error to tune the controller. These two factors combined result in inefficient tuning by hand, especially when the dimension of parameter space is high (e.g., 12 for the geometric control [61]). We display the tuned parameters in Table VII in Appendix C, along with the initial parameters and hand-tuned parameters for comparison.

C. Generalization

We conducted experiments to test the generalization capability of the tuned parameters in Section VI-B. The testing set contains circular, 3-D lemniscate, and vertical lemniscate trajectories, with their coordinates shown in Table III and shapes

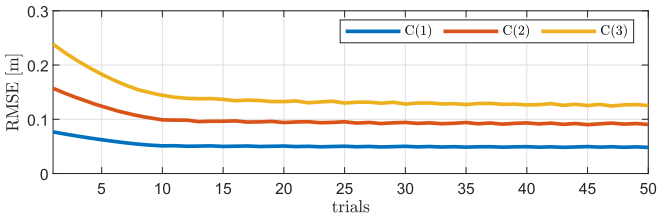


Fig. 11. Simulation-based autotuning performance for a quadrotor (with identical physical parameters to the real quadrotor in experiments) on circular trajectories with speeds ranging from 1 to 3 m/s.).

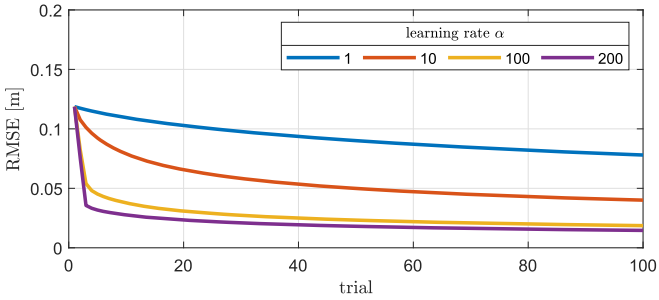


Fig. 12. Performance of DiffTune in Dubin's car example with different learning rates.

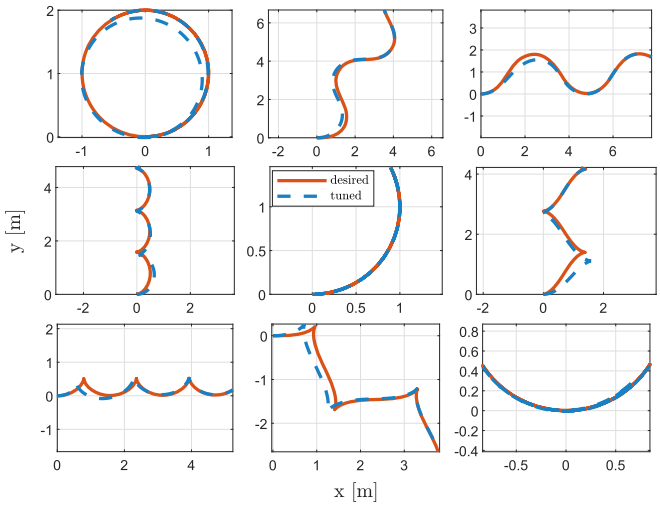


Fig. 13. Trajectories for batch tuning in Dubin's car example.

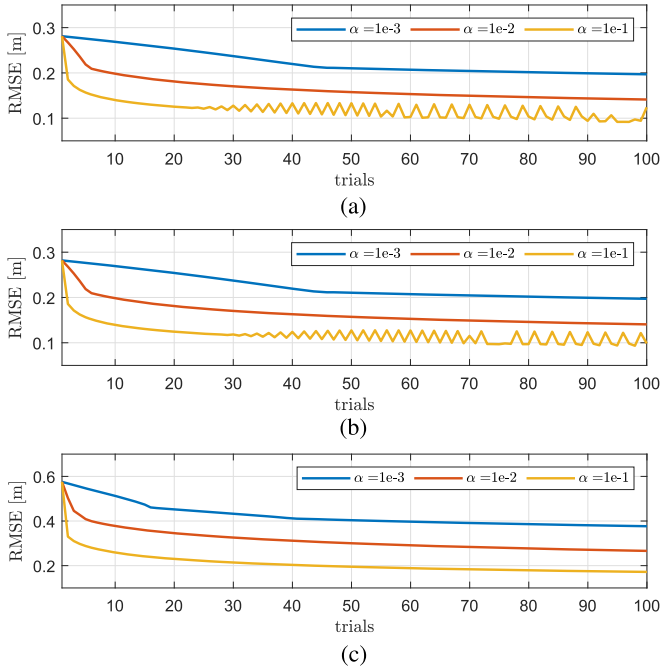


Fig. 14. Loss reduction by DiffTune under different learning rates α in the quadrotor simulation. (a) 2-D circle. (b) 3-D circle. (c) 3-D lemniscate.

shown in Fig. 15 (in the Appendix C). The latter two trajectories are considered here for their wide range of speed and acceleration (see Table IV), which is in contrast to those static values of the circular trajectories. We test the three groups of tuned parameters P1, P2, and P3 on circular trajectories with 1, 2, and 3 m/s speed, respectively. The tracking performance of these tuned parameters on the testing trajectories is shown in Table IV, in which we also include the baseline of hand-tuned parameters.

When tested on the circular trajectories, the tuned parameters perform the best on the speed that they were tuned for, i.e., Pn performs the best on trajectory $C(n)$ for $n \in \{1, 2, 3\}$. The same phenomenon has been observed in [12] (although the controller therein is different from the one used here), where parameters perform the best over the trajectories that they are autotuned on. This behavior is similar to an overfitted NN in machine learning. In our case, this type of "overfitting" is expected since the proportional-derivative structure of the geometric controller determines that there does not exist a group of parameters that work well for all conditions (e.g., the aggressive trajectory $C(3)$ demands a distinct parameter combination than for the slow trajectory $C(1)$). However, the parameters can still generalize to the 3-D lemniscate and vertical lemniscate trajectories that are not used for tuning. Specifically, P2 and P3, which are tuned for increasingly aggressive maneuvers with fast-changing directions of velocity and acceleration, generalize to the two variants of the lemniscate trajectory that demands fast-changing speed and acceleration. P3 demonstrates better agility, which shows the best performance on $DF(2)$ and $VF(2)$. However, P3 is overly agile for the speed and acceleration in $DF(1)$ and $VF(1)$, which leads to minimum RMSE compared to P1 and P2, albeit with minor oscillations on the pitch angle.

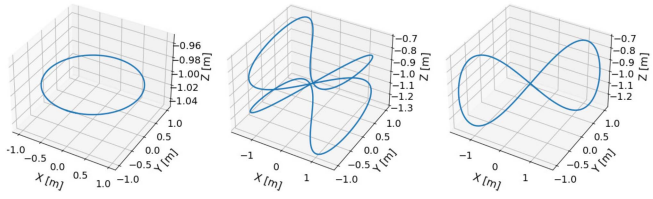
D. Ablation Study

We conduct an ablation study of how much contribution DiffTune and $\mathcal{L}_1$AC provide to performance improvement. We repeat the tuning in Section VI-B but with $\mathcal{L}_1$AC in the loop, which results in different sets of parameters for the three trajectories denoted by Pn-$\mathcal{L}_1$ for $n \in \{1, 2, 3\}$ and shown in Table VII in Appendix C. Our implementation follows [58]. Unlike the simulations in Section V-B where we deliberately introduce uncertainties in MoI, in the experiment, we do not introduce uncertainties. The quadrotor naturally has uncertainties existing in the system (e.g., varying battery voltage in flight and mismatch between the actual MoI and estimated MoI through CAD computation), in which case $\mathcal{L}_1$AC can help compensate for these uncertainties and thus improve the tracking performance. The results are shown in Table V. Here, "DiffTune off" shows

the performance of the initial parameters (no tuning occurs); "DiffTune on" shows the tracking performance in the final tuning trial. "$\mathcal{L}_1$ on/off" indicate whether $\mathcal{L}_1$AC is used or not during the tuning trials. We conclude that 1. DiffTune alone can improve the tracking performance, albeit the system has uncertainties and measurement noise. 2. When $\mathcal{L}_1$AC is combined with DiffTune, $\mathcal{L}_1$AC helps improve the performance in two ways: i) compensating for the uncertainties so that the uncertainties' degradation to system performance is mitigated; and ii) DiffTune can proceed with less biased gradient thanks to the uncertainties being "canceled out" by $\mathcal{L}_1$AC, which leads to more efficient tuning. Intuitively, DiffTune raises the performance's upper limit (in an ideal case subject to no uncertainty), whereas $\mathcal{L}_1$AC keeps the actual performance (in a realistic case subject to uncertainties) close to the upper limit. When the two are used together, the best performance is achieved.

E. Comparison of Parameters Autotuned in Experiments With Those Obtained in Simulations

In this section, we compare the parameters autotuned in experiments (detailed in Section VI-B) with those obtained in simulations. For the latter, we apply Algorithm 1 to a quadrotor system in simulation, where the vehicle's physical parameters (mass and inertia) are identical to the quadrotor used in the experiments. Furthermore, to stay consistent with the setup in experiments, we use the same initial controller parameters, loss function, horizon of evaluation, gradient clipping, and sampling time as in Section VI-B. Since equipment wear and tear is not an issue in simulation, we raise the autotuning budget to 50 trials and reduce the learning rate α to 0.01. We autotune parameters for the circular trajectories at speeds of 1, 2, and 3 m/s. In Fig. 11, we show the tracking errors through the trials. The tracking error shows a monotone reduction in the beginning and ends with small oscillations when it terminates after 50 trials, which indicates that the best performance is achieved with the selected learning rate. We deploy the parameters autotuned in simulations on the real quadrotor used in the previous experiments in Section VI-B. Furthermore, we test the parameters obtained in the 10th, 30th, and 50th trials in the simulation to examine the performance at different stages of the autotuning. The results are shown in Table VI. In general, one can observe the reduced tracking error from the parameters that have been obtained through more trials in simulation, for example, with trajectories $C(1)$ and $C(2)$. However, the crashes seen on trajectories $C(2)$ and $C(3)$ with relatively more simulation trials indicate the common issue of sim-to-real gap: as the parameters evolve on simulation data, they inevitably (over)fit the simulation rather than the real system. The new results indicate the benefit of experiment-based autotuning, which is conditioned on the model-based gradient on data from a real system, and thus provides the best knowledge of parameter change for performance improvement. Nevertheless, the parameters extracted from the tenth trial in simulations result in tracking errors close to the errors with the hand-tuned parameters, which may be used as initial parameters or warm start for further performance improvement in experiment-based autotuning.



(a) (b) (c)

Fig. 15. Trajectories used in the generalization test. (a) Circle. (b) 3-D lemniscate. (c) Vertical lemniscate.

VII. Conclusion

In this article, we propose DiffTune: an autotuning method using autodifferentiation, with the advantage of stability, compatibility with data from physical systems, and efficiency. Given a performance metric, DiffTune gradually improves the performance using gradient descent, where the gradient is computed using sensitivity propagation that is compatible with physical systems' data. We also show how to use $\mathcal{L}_1$AC to mitigate the discrepancy between the nominal model and the associated physical system when the latter suffers from uncertainties. Simulation results (on a Dubin's car and a quadrotor) and experimental results (on a quadrotor) both show that DiffTune can efficiently improve the system's performance. When uncertainties are present in a system, $\mathcal{L}_1$ adaptive control facilitates tuning by compensating for the uncertainties. Generalization of the tuned parameters to unseen trajectories during tuning is also illustrated in both simulation and experiments.

One limitation of the proposed approach is that it only applies to systems with differentiable dynamics and controllers. The requirement on differentiability is not met in contact-rich applications [62] (e.g., legged robots [63] and dexterous manipulation [64]) and systems with actuation limits [7] (e.g., saturations in magnitude or changing rate). Although subgradients [65] generally exist at the points of discontinuity, the impact of surrogate gradient on the tuning efficiency is unknown, which will be investigated in the future. Another limitation of this work is the convergence to a local minimum due to the usage of gradient descent. Future work will investigate conditions for convergence to a global minimum or methods that can help escape from local minimums.

Appendix

A. Details of the Dubin's Car Simulation

Autotuning setup: For Autotune [12], we use a Gaussian distribution with variances set to 1.0 to sample the four parameters iteratively. The scoring function is the exponential of the tracking RMSE. Since sampling is used in AutoTune, we conduct ten runs and show the mean, max, and min RMSEs in Fig. 4. For SafeOpt [9], we choose the radial basis function kernel for GP. Both the length scales and the kernel variance are set to 1.0. The noise variance for the model is set to 0.01. The objective function to maximize is negative tracking RMSE, and the safety threshold is set to -1 m. For GIBO [43], stochastic gradient descent is used as an optimizer, and the learning rate is set to 1. Like SafeOpt, the

TABLE VII
COMPARISON OF THE INITIAL PARAMETERS, DIFFTUNE FINAL PARAMETERS (P1, P2, AND P3), DIFFTUNE-$\mathcal{L}_1$AC FINAL PARAMETERS (P1-$\mathcal{L}_1$, P2-$\mathcal{L}_1$, AND P3-$\mathcal{L}_1$), AND HAND-TUNED PARAMETERS

Param.	$\boldsymbol{k_p}$			$\boldsymbol{k_v}$			$\boldsymbol{k}_R$			$\boldsymbol{k_\Omega}$		
axis	x	y	z	x	y	z	x	y	z	x	y	z
initial	10	10	10	2	1	1.5	0.5	0.3	0.14	0. 11	0.1	0.01
P1	10.25	10.20	11.72	0.97	0.35	0.85	1.02	0.29	0.05	0.05	0.10	0.03
P2	12.91	12.92	13.75	0.96	0.48	0.72	0.97	0.58	0.07	0.05	0.05	0.01
P3	15.56	16.62	14.54	4.01	2.59	0.52	1.16	0.78	0.05	0.07	0.03	0.03
P1-$\mathcal{L}_1$	10.52	10.44	10.04	0.71	0.34	1.40	1.15	0.67	0.13	0.06	0.04	0.03
P2-$\mathcal{L}_1$	11.79	11.77	12.14	0.96	0.48	0.72	0.97	0.58	0.22	0.05	0.05	0.01
P3-$\mathcal{L}_1$	16.29	16.11	16.26	1.27	0.95	0.52	1.29	0.77	0.05	0.07	0.04	0.03
hand-tuned	14	15	15	1.50	0.90	1.10	0.55	0.35	0.15	0.04	0.03	0.01

objective function to maximize is the negative tracking RMSE. The GP model uses a squared exponential kernel, and we assume the initial loss as a prior mean of the model. At each trial, four samples are used for a gradient estimate, and the last 20 sampled points are used for the approximation of the posterior. The normalized gradient is used for the parameter update. For DiffTune, we use a learning rate $\alpha = 100$. (The tracking RMSE's reduction with different learning rates in Dubin's car example is shown in Fig. 12, which is consistent with how learning rate influences the loss reduction in general in a gradient-descent algorithm.) The feasible set of each parameter is set to [1, 20] and applies to all three methods compared.

For the batch autotuning for generalization, we see the loss function as the squared norm of the position tracking error summed over a horizon of 10 s and set the learning rate α to 0.1. The termination condition is the relative reduction in the total loss between two consecutive steps being smaller than 1e$-$4 of the current loss value.

The trajectories used for batch tuning for Dubin's car are shown in Fig. 13, where the tuned parameters can achieve acceptable tracking on these trajectories.

B. Details of the Quadrotor Simulation

Autotuning setup: For all methods, we use the same feasible set of parameters and the same horizon for performance evaluation. For AutoTune [12], the variances of Gaussian distribution in the transition model are set to 2 for $\boldsymbol{k_p}$, $\boldsymbol{k_v}$, and $\boldsymbol{k}_R$ and 1 for $\boldsymbol{k_\Omega}$. The scoring function is the exponential of the RMSE tracking error. For the SafeOpt-PSO [31], the swarm size is set to 400 and we choose Matérn kernel with parameter $\nu = 5/2$. The length scales are set to 5 for $\boldsymbol{k_p}$, $\boldsymbol{k_v}$, and $\boldsymbol{k}_R$ and 0.1 for $\boldsymbol{k_\Omega}$. The kernel variance is set to 0.01, and the noise variance for the model is set to 0.05 for all trajectories. The objective function to maximize is negative tracking RMSE, and we set the safety threshold to -1 m. For GIBO [43], we have used the same setup for all trajectories. Stochastic gradient descent is used as an optimizer, and the learning rate is scheduled to 0.5 for the first 40 trials and then reduced to 0.2 until the maximum number of trials is reached. The objective function is set to the negative tracking RMSE (for maximization). The GP model uses a squared exponential kernel, and we assume the initial loss of the 3-D lemniscate trajectory as a prior mean of the model. We use a uniform prior between 0.1 and 5 as a hyperprior of the length scales. At each trial, 12 samples are used for a gradient estimate, and the last 12 sampled points are used to approximate the posterior. We set $\delta = 0.5$ to limit local search bounds. The normalized gradient is used for the parameter update.

For DiffTune, the learning rate is set to 0.1. In addition, we tested three learning rates of $\alpha \in \{0.1,\ 0.01,\ 0.001\}$ for DiffTune. The results are shown in Fig. 14, where the rate of loss reduction is positively correlated with the magnitude of the learning rate. Furthermore, oscillation in the loss value is observed when α is relatively large (0.1), indicating the learning rate is too large when the parameters are close to the (local) minimum. These observations are consistent with how the learning rate influences the loss reduction in gradient descent.

In the ablation study of $\mathcal{L}_1$AC and DiffTune, we relax the upper bound on the feasible parameters so that the parameters can grow to higher values to handle the uncertainties.

C. Details of the Quadrotor Experiment

We use a custom-built quadrotor to conduct the experiments. The quadrotor weighs 0.63 kg with a 0.22 m diagonal motor-to-motor distance. The quadrotor is controlled by a Pixhawk 4 mini flight controller running the ArduPilot firmware. We modify the firmware to enable the geometric controller and the $\mathcal{L}_1$ adaptive control, which both run at 400 Hz on the Pixhawk. Position feedback is provided by 9 Vicon V16 cameras. We use ArduPilot's EKF to fuse the Vicon measurements with IMU readings onboard.

The tuned parameters, without and with $\mathcal{L}_1$AC in the loop, are shown in Table VII. The trajectories used for testing generalization in Table III are shown in Fig. 15.

ACKNOWLEDGMENT

The authors would like to thank Pan Zhao, Aditya Gahlawat, and Zhuohuan Wu for their valuable feedback during insightful discussions and Junjie Gao for his help with demo filming.

REFERENCES

[1] A. O'dwyer, *Handbook of PI and PID Controller Tuning Rules*. Singapore: World Scientific, 2009.

[2] C.-C. Yu, *Autotuning of PID Controllers: A. Relay Feedback Approach*. Berlin, Germany: Springer, 2006.

[3] K. J. Åström, T. Hägglund, C. C. Hang, and W. K. Ho, "Automatic tuning and adaptation for PID controllers—A survey," *Control Eng. Pract.*, vol. 1, no. 4, pp. 699–714, 1993.

[4] Y. Li, K. H. Ang, and G. C. Chong, "Patents, software, and hardware for PID control: An overview and analysis of the current art," *IEEE Control Syst. Mag.*, vol. 26, no. 1, pp. 42–54, Feb. 2006.

[5] M. Zhuang and D. Atherton, "Automatic tuning of optimum PID controllers," in *Proc. IEE Proc. D. (Control Theory Appl.)*, IET Digital Library, 1993, vol. 140, no. 3, pp. 216–224.

[6] S. Trimpe, A. Millane, S. Doessegger, and R. D'Andrea, "A self-tuning LQR approach demonstrated on an inverted pendulum," *IFAC Proc. Vol.*, vol. 47, no. 3, pp. 11281–11287, 2014.

[7] A. R. Kumar and P. J. Ramadge, "DiffLoop: Tuning PID controllers by differentiating through the feedback loop," in *Proc. IEEE 55th Annu. Conf. Inf. Sci. Syst.*, Baltimore, MD, USA, 2021, pp. 1–6.

[8] A. Marco, P. Hennig, J. Bohg, S. Schaal, and S. Trimpe, "Automatic LQR tuning based on gaussian process global optimization," in *Proc. IEEE Int. Conf. Robot. Autom.*, Stockholm, Sweden, 2016, pp. 270–277.

[9] F. Berkenkamp, A. P. Schoellig, and A. Krause, "Safe controller optimization for quadrotors with Gaussian processes," in *Proc. IEEE Int. Conf. Robot. Autom.*, Stockholm, Sweden, 2016, pp. 491–496.

[10] R. Calandra, N. Gopalan, A. Seyfarth, J. Peters, and M. P. Deisenroth, "Bayesian gait optimization for bipedal locomotion," in *Proc. Int. Conf. Learn. Intell. Optim.*, Gainesville, FL, USA, 2014, pp. 274–290.

[11] D. J. Lizotte, T. Wang, M. H. Bowling, and D. Schuurmans, "Automatic gait optimization with Gaussian process regression," in *Proc. Int. Joint Conf. Artif. Intell. Org.*, vol. 7, Hyderabad, India, 2007, pp. 944–949.

[12] A. Loquercio, A. Saviolo, and D. Scaramuzza, "AutoTune: Controller tuning for high-speed flight," *IEEE Robot. Autom. Lett.*, vol. 7, no. 2, pp. 4432–4439, Apr. 2022.

[13] M. Mehndiratta, E. Camci, and E. Kayacan, "Can deep models help a robot to tune its controller? A step closer to self-tuning model predictive controllers," *Electronics*, vol. 10, no. 18, 2021, Art. no. 2187.

[14] R. Moriconi, M. P. Deisenroth, and K. Sesh Kumar, "High-dimensional Bayesian optimization using low-dimensional feature spaces," *Mach. Learn.*, vol. 109, pp. 1925–1943, 2020.

[15] B. Amos and J. Z. Kolter, "OptNet: Differentiable optimization as a layer in neural networks," in *Proc. 34th Int. Conf. Mach. Learn.*, Sydney, Australia, 2017, pp. 136–145.

[16] B. Amos, I. Jimenez, J. Sacks, B. Boots, and J. Z. Kolter, "Differentiable MPC for end-to-end planning and control," in *Proc. 32nd Conf. Neural Inf. Process. Syst.*, vol. 31, Montreal, Canada, 2018.

[17] S. East, M. Gallieri, J. Masci, J. Koutnik, and M. Cannon, "Infinite-horizon differentiable model predictive control," in *Proc. Int. Conf. Learn. Representations*, 2020, pp. 1–15.

[18] W. Jin, Z. Wang, Z. Yang, and S. Mou, "Pontryagin differentiable programming: An end-to-end learning and control framework," in *Proc. 34th Conf. Neural Inf. Process. Syst.*, vol. 33, Vancouver, Canada, 2020, pp. 7979–7992.

[19] W. Jin, T. D. Murphey, D. Kulić, N. Ezer, and S. Mou, "Learning from sparse demonstrations," *IEEE Trans. Robot.*, vol. 39, no. 1, pp. 645–664, Feb. 2023.

[20] H. Ma, B. Zhang, M. Tomizuka, and K. Sreenath, "Learning differentiable safety-critical control using control barrier functions for generalization to novel environments," in *Proc. Eur. Control Conf. (ECC)*, pp. 1301–1308, 2022.

[21] W. Xiao et al., "BarrierNet: A safety-guaranteed layer for neural networks," *IEEE Trans. Robot.*, vol. 39, no. 3, pp. 2289–2307, 2023.

[22] W. Xiao, T.-H. Wang, M. Chahine, A. Amini, R. Hasani, and D. Rus, "Differentiable control barrier functions for vision-based end-to-end autonomous driving," 2022, *arXiv:2203.02401*.

[23] H. Parwana and D. Panagou, "Recursive feasibility guided optimal parameter adaptation of differential convex optimization policies for safety-critical systems," in *Proc. Int. Conf. Robot. Automat.*, 2022, pp. 6807–6813.

[24] N. A. Vien and G. Neumann, "Differentiable robust LQR layers," 2021, *arXiv:2106.05535*.

[25] A. Paszke et al., "PyTorch: An imperative style, high-performance deep learning library," in *Proc. 33rd Conf. Neural Inf. Process. Syst.*, Vancouver, Canada, 2019, pp. 8024–8035.

[26] M. Abadi et al., "TensorFlow: Large-scale machine learning on heterogeneous systems," 2016, *arXiv:1603.04467*.

[27] J. Bradbury et al., "JAX: Composable transformations of Python NumPy programs," 2018. [Online]. Available: http://github.com/google/jax

[28] J. A. E. Andersson, J. Gillis, G. Horn, J. B. Rawlings, and M. Diehl, "CasADi–A software framework for nonlinear optimization and optimal control," *Math. Program. Comput.*, vol. 11, no. 1, pp. 1–36, 2019.

[29] H. K. Khalil, *Nonlinear Control*, vol. 406. London, U.K.: Pearson, 2015.

[30] N. Hovakimyan and C. Cao, L_1 *Adaptive Control Theory: Guaranteed Robustness With Fast Adaptation*. Philadelphia, PA, USA: SIAM, 2010.

[31] R. R. Duivenvoorden, F. Berkenkamp, N. Carion, A. Krause, and A. P. Schoellig, "Constrained Bayesian optimization with particle swarms for safe adaptive controller tuning," *IFAC-PapersOnLine*, vol. 50, no. 1, pp. 11800–11807, 2017.

[32] S. Cheng, L. Song, M. Kim, S. Wang, and N. Hovakimyan, "Difftune$^+$: Hyperparameter-free auto-tuning using auto-differentiation," in *Proc. Learn. Dyn. Control Conf. PMLR*, 2023, pp. 170–183.

[33] J. C. Spall, *Introduction to Stochastic Search and Optimization: Estimation, Simulation, and Control*. Hoboken, NJ, USA: Wiley, 2005.

[34] A. Romero, S. Sun, P. Foehn, and D. Scaramuzza, "Model predictive contouring control for time-optimal quadrotor flight," *IEEE Trans. Robot.*, vol. 38, no. 6, pp. 3340–3356, Dec. 2022.

[35] A. Romero, Y. Song, and D. Scaramuzza, "Actor-critic model predictive control," 2023, *arXiv:2306.09852*.

[36] P. R. Giordano, Q. Delamare, and A. Franchi, "Trajectory generation for minimum closed-loop state sensitivity," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2018, pp. 286–293.

[37] A. Srour, A. Franchi, and P. R. Giordano, "Controller and trajectory optimization for a quadrotor UAV with parametric uncertainty," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2023, pp. 1–7.

[38] N. J. Killingsworth and M. Krstic, "PID tuning using extremum seeking: Online, model-free performance optimization," *IEEE Control Syst. Mag.*, vol. 26, no. 1, pp. 70–79, Feb. 2006.

[39] W. K. Hastings, "Monte Carlo sampling methods using Markov chains and their applications," *Biometrika*, vol. 57, no. 1, pp. 97–109, doi: 10.1093/biomet/57.1.97.

[40] W. Edwards, G. Tang, G. Mamakoukas, T. Murphey, and K. Hauser, "Automatic tuning for data-driven model predictive control," in *Proc. IEEE Int. Conf. Robot. Autom.*, Xi'an, China, 2021, pp. 7379–7385.

[41] M. A. Gelbart, J. Snoek, and R. P. Adams, "Bayesian optimization with unknown constraints," 2014, *arXiv:1403.5607*.

[42] F. Berkenkamp, A. Krause, and A. P. Schoellig, "Bayesian optimization with safety constraints: Safe and automatic parameter tuning in robotics," *Mach. Learn.*, vol. 112, no. 10, pp. 3713–3747, 2023.

[43] S. Müller, A. von Rohr, and S. Trimpe, "Local policy search with Bayesian optimization," in *Proc. Adv. Neural Inf. Process. Syst.*, 2021, pp. 20708–20720.

[44] A. S. Polydoros and L. Nalpantidis, "Survey of model-based reinforcement learning: Applications on robotics," *J. Intell. Robot. Syst.*, vol. 86, no. 2, pp. 153–173, 2017.

[45] F. Tambon et al., "How to certify machine learning based safety-critical systems? A systematic literature review," *Autom. Softw. Eng.*, vol. 29, no. 2, pp. 1–74, 2022.

[46] H. Ravichandar, A. S. Polydoros, S. Chernova, and A. Billard, "Recent advances in robot learning from demonstration," *Annu. Rev. Control, Robot., Auton. Syst.*, vol. 3, pp. 297–330, 2020.

[47] J.-X. Xu, "A survey on iterative learning control for nonlinear systems," *Int. J. Control*, vol. 84, no. 7, pp. 1275–1294, 2011.

[48] D. A. Bristow, M. Tharayil, and A. G. Alleyne, "A survey of iterative learning control," *IEEE Control Syst. Mag.*, vol. 26, no. 3, pp. 96–114, Jun. 2006.

[49] L. Brunke et al., "Safe learning in robotics: From learning-based control to safe reinforcement learning," *Annu. Rev. Control, Robot., Auton. Syst.*, vol. 5, pp. 411–444, 2022.

[50] R. D. Neidinger, "Introduction to automatic differentiation and MATLAB object-oriented programming," *SIAM Rev.*, vol. 52, no. 3, pp. 545–563, 2010.

[51] A. G. Baydin, B. A. Pearlmutter, A. A. Radul, and J. M. Siskind, "Automatic differentiation in machine learning: A survey," *J. Mach. Learn. Res.*, vol. 18, pp. 1–43, 2018.

[52] N. Parikh et al., "Proximal algorithms," *Found. Trends Optim.*, vol. 1, no. 3, pp. 127–239, 2014.

[53] X. Wang and N. Hovakimyan, "$\mathcal{L}_1$ adaptive controller for nonlinear time-varying reference systems," *Syst. Control Lett.*, vol. 61, no. 4, pp. 455–463, 2012.

[54] Z. Wu et al., "$\mathcal{L}_1$ adaptive augmentation for geometric tracking control of quadrotors," in *Proc. Int. Conf. Robot. Autom.*, Philadelphia, PA, USA, 2022, pp. 1329–1336.
[55] J. Pravitra, K. A. Ackerman, C. Cao, N. Hovakimyan, and E. A. Theodorou, "$\mathcal{L}_1$-adaptive MPPI architecture for robust and agile control of multirotors," in *Proc. IEEE/RSJ Int. Conf. Intell. Robot Syst.*, Las Vegas, NV, USA, 2020, pp. 7661–7666.
[56] D. Hanover, P. Foehn, S. Sun, E. Kaufmann, and D. Scaramuzza, "Performance, precision, and payloads: Adaptive nonlinear MPC for quadrotors," *IEEE Robot. Autom. Lett.*, vol. 7, no. 2, pp. 690–697, Apr. 2022.
[57] P. Zhao, Y. Mao, C. Tao, N. Hovakimyan, and X. Wang, "Adaptive robust quadratic programs using control Lyapunov and barrier functions," in *Proc. 59th IEEE Conf. Decis. Control*, 2020, pp. 3353–3358.
[58] Z. Wu et al., "$\mathcal{L}_1$Quad: $\mathcal{L}_1$ adaptive augmentation of geometric control for agile quadrotors with performance guarantees," 2023, *arXiv:2302.07208*.
[59] A. Gahlawat et al., "Contraction $\mathcal{L}_1$-adaptive control using Gaussian processes," in *Proc. 3rd Conf. Learn. Dyn. Control*, Jun. 2021, vol. 144, pp. 1027–1040.
[60] S. Cheng, L. Song, and M. Kim, 2023. [Online]. Available: https://github.com/Sheng-Cheng/DiffTuneOpenSource
[61] T. Lee, M. Leok, and N. H. McClamroch, "Geometric tracking control of a quadrotor UAV on SE(3)," in *Proc. IEEE 49th Conf. Decis. Control*, Atlanta, GA, USA, 2010, pp. 5420–5425.
[62] M. Parmar, M. Halm, and M. Posa, "Fundamental challenges in deep learning for stiff contact dynamics," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2021, pp. 5181–5188.
[63] P.-B. Wieber, R. Tedrake, and S. Kuindersma, "Modeling and control of legged robots," in *Springer Handbook of Robotics*. Berlin, Germany: Springer, 2016, pp. 1203–1234.
[64] W. Jin and M. Posa, "Task-driven hybrid model reduction for dexterous manipulation," 2022, *arXiv:2211.16657*.
[65] S. Boyd, L. Xiao, and A. Mutapcic, "Subgradient methods," *Lecture Notes EE392o, Stanford Univ., Autumn Quarter*, vol. 2004, pp. 2004–2005, 2003.

Sheng Cheng (Member, IEEE) received the B.Eng. degree in control science and engineering from Harbin Institute of Technology, Harbin, China, and the M.S. and Ph.D. degrees in electrical engineering from the University of Maryland, College Park, MD, USA, in 2014, 2018, and 2021, respectively.

He is a Postdoctoral Research Associate with the University of Illinois Urbana-Champaign, Champaign, IL, USA. His current research interests include aerial robotics, learning for control, adaptive control, and optimization.

Minkyung Kim (Graduate Student Member, IEEE) received the B.Eng. degree in mechanical engineering from the Korea Advanced Institute of Science and Technology, Daejeon, South Korea, in 2020. She is currently working toward the Ph.D. degree in mechanical engineering with the University of Illinois Urbana-Champaign, Champaign, IL, USA.

Her research interests include robotics and machine learning, focusing on learning for control and safe autonomous systems.

Lin Song received the B.E. degree in automation from the Harbin Institute of Technology, Harbin, China, in 2017, the M.S. degree in mechanical engineering from Carnegie Mellon University, Pittsburgh, PA, USA, in 2019, and the Ph.D. degree in mechanical engineering from the University of Illinois at Urbana-Champaign, Urbana, IL, USA, in 2024.

His research interests include optimal control, nonlinear and adaptive control, and safe data-driven control.

Chengyu Yang received the B.S. degree in ocean engineering and technology from Zhejiang University, Hangzhou, China, in 2021. He received the M.S. degree in mechanical engineering from the University of Illinois, Urbana-Champaign, Champaign, IL, USA, in 2023. He is currently working toward the Ph.D. degree in mechanical engineering with the University of Illinois, Urbana-Champaign.

His research interests include adaptive control, robust control and planning for autonomous robots.

Yiquan Jin received the B.Eng. degree in mechanical engineering from Zhejiang University, Haining, China, and the B.S. degree in mechanical engineering from the University of Illinois Urbana-Champaign, Champaign, IL, USA, in 2024. He will continue the graduate studies at University of Pennsylvania, Philadelphia, PA, USA.

His current research interests include aerial robotics, machine learning and control systems.

Shenlong Wang received the Ph.D. degree from the University of Toronto, Toronto, ON, Canada, in 2021. He is an Assistant Professor with the Department of Computer Science, University of Illinois Urbana-Champaign, Champaign, IL, USA, specializing in computer vision and robotics.

His research focuses on creating a digital replica of the world and simulating realistic new content to train and validate autonomous systems. His past work received IROS Best Application Paper Runner-Up and CVPR Best Paper Candidate. His contributions to autonomy and simulation have led to 25 filed patents.

Dr. Wang has received the NSF CAREER Award, Amazon Research Award, and various fellowships from Facebook, Adobe, and the Royal Bank of Canada. He regularly serves as an Area Chair for conferences in computer vision, robotics, and machine learning.

Naira Hovakimyan (Fellow, IEEE) received the M.S. degree in applied mathematics from Yerevan State University, Yerevan, Armenia, in 1988. She received the Ph.D. degree in physics and mathematics from the Institute of Applied Mathematics, Russian Academy of Sciences in Moscow, Moscow, Russia, in 1992.

She is currently W. Grafton and Lillian B. Wilkins Professor of mechanical science and engineering and the Director of AVIATE Center of UIUC. She has coauthored two books, 11 patents, and more than 450 refereed publications.

Dr. Hovakimyan is the 2011 recipient of AIAA Mechanics and Control of Flight Award, the 2015 recipient of SWE Achievement Award, the 2017 recipient of IEEE CSS Award for Technical Excellence in Aerospace Controls, and the 2019 recipient of AIAA Pendray Aerospace Literature Award. In 2014, she was awarded the Humboldt prize for her lifetime achievements. In 2015 and 2023, she was awarded the UIUC Engineering Council Award for Excellence in Advising, and in 2024 she was recognized as the Winner of the College Award for Excellence in Translational Research. She is a fellow of AIAA, ASME, and a senior member of NAI. She is a cofounder and the Chief Scientist of Intelinair. Her work was featured in the New York Times, on Fox TV and CNBC.