2023 American Control Conference (ACC)
San Diego, CA, USA. May 31 - June 2, 2023

A Tutorial on Derivative-Free Policy Learning Methods for Interpretable Controller Representations

Joel A. Paulson, Farshud Sorourifar, and Ali Mesbah

***Abstract*— This paper provides a tutorial overview of recent advances in learning control policy representations for complex systems. We focus on control policies that are determined by solving an optimization problem that depends on the current state and some adjustable parameters. We refer to such policies as interpretable in the sense that each of the individual components can be directly understood by practitioners once the parameters are set, i.e., the objective function encodes the desired goal and the constraint functions enforce the rules of the system. We discuss how various commonly used control policies can be viewed in this manner such as the linear quadratic regulator, (nonlinear) model predictive control, and approximate dynamic programming. Traditionally, the parameters that appear in these control policies have been tuned by hand, expert knowledge, or simple trial-and-error experimentation, which can be time consuming and lead to suboptimal results in practice. To this end, we describe how the Bayesian optimization framework, which is a class of efficient derivative-free optimization methods for noisy functions, can be used to efficiently automate this process. In addition to reviewing relevant literature and demonstrating the effectiveness of these new methods on an illustrative example problem, we also offer perspectives on future research in this area.**

I. Introduction

Reinforcement learning (RL) is a subfield of machine learning that focuses on how to use past system measurements to improve the future control of a dynamic system [1]–[3]. RL can be viewed as a collection of (often approximate) solution approaches to stochastic optimal control problems that represent optimal state-to-action policies in uncertain control systems. Even though RL has been around for several decades, it recently gained massive publicity in 2016 when it was used to beat the best Go player in the world, which had been previously thought impossible due to the enormous state/action space ($> 10^{170}$ states) [4]. Although nothing short of amazing, such game applications have a few useful characteristics that do not hold in most real-world engineering problems: (i) all system states can be measured; (ii) all measurements are perfect (noise-free); and (iii) huge amounts of data can be collected across many different conditions. This dramatic progress in RL begs the question: can we exploit the same methods to solve challenging next-generation control tasks such as self-driving vehicles, agile robotic systems, and smart manufacturing? For RL to expand into such areas, the methods must work safe and reliably, especially when the three previous assumptions are not satisfied, since the failure of such systems can have severe economic and social consequences (including loss of human life).

Model-free RL methods are usually divided into one of two major categories: approximate dynamic programming (ADP) and policy search [2]. ADP methods directly approximate the optimality conditions for the stochastic optimal control problem using some type of value function representation. Even though they have the potential to converge to the true optimal solution in the limit of enough data, ADP methods often end up providing an implicit representation of the policy that can be difficult to interpret (and may perform poorly for small to medium amounts of data) [5]. Policy search methods, on the other hand, look to directly optimize parameters that appear in an explicitly parametrized control policy. Thus, in practice, we can think of policy search as a way to convert the RL problem into a derivative-free optimization (DFO) problem [6]. By selecting the right policy parametrization, we can (greatly) reduce the size of the search space, as well as generate easier to implement and more interpretable control policies. However, many possible policy parametrizations exist, and the choice of policy has an important impact on the selection of a suitable DFO method.

In this tutorial, we give an overview of recent research efforts on control policy learning that are able to address some of the previously mentioned challenges encountered by traditional RL methods when applied to real-world systems. In particular, we consider control of stochastic dynamical systems using "interpretable" feedback policies that determine the input to give to the system by solving an optimization problem in real-time. We can generally express a control policy as $u_t = \pi(x_t;\theta)$, where $\pi : \mathbb{R}^n \times \mathbb{R}^p \to \mathbb{R}^m$ is the policy function that maps a measured (or estimated) system state $x_t \in \mathbb{R}^n$ to a control input (or action) $u_t \in \mathbb{R}^m$ given some vector value $\theta \in \Theta \subseteq \mathbb{R}^p$ of policy parameters. Many different structures for π have been studied. Explicit policies refer to π that are available in an analytic form such as proportional-integral-derivative and neural network controllers. In the (shallow) neural network case, which has become increasingly popular, the policy will have the following parametrized form

$$\pi(x;\theta) = W_1\alpha(W_0x + b_0) + b_1, \tag{1}$$

where $W_0 \in \mathbb{R}^{h\times n}$ and $b_0 \in \mathbb{R}^h$ are the weight and bias parameters in the first layer given $h \in \mathbb{N}$ hidden nodes, $W_1 \in \mathbb{R}^{m\times h}$ and $b_0 \in \mathbb{R}^m$ are the weight and bias parameters in

J. A. Paulson and F. Sorourifar are with the Department of Chemical and Biomolecular Engineering, The Ohio State University, Columbus, OH 43210, USA. {paulson.82,sorourifar.1}@osu.edu

A. Mesbah is with the Department of Chemical and Biomolecular Engineering, University of California, Berkeley, CA 94720, USA. mesbah@berkeley.edu

This work was supported by the US NSF under Grant 2237616.

the output layer, and $\alpha : \mathbb{R}^h \to \mathbb{R}^h$ is a nonlinear activation function (e.g., the rectified linear unit $\alpha(y) = \max\{y, 0\}$). In this case, the learnable parameters are $\theta = \{W_0, b_1, W_1, b_1\}$. Although (1) is a universal function approximator in the sense that it is expressive enough to represent any function as $h \to \infty$ [7], it involves a large number of parameters whose values are not necessarily interpretable by humans. Not only can this increase training complexity, it also leads to less trust when attempting to implement the policy in practice since practitioners cannot be confident in its ability to extrapolate beyond the limited observed training instances.

We focus on a certain class of control policies defined implicitly in terms of an interpretable or understandable set of equations [8]. Since mathematical optimization problems require the specification of a clear goal (objective) and set of rules (constraints), they are naturally comprised of interpretable components such that we focus on well-defined structures of the following form

$$\pi(x;\theta) = \underset{u}{\operatorname{argmin}} \quad f_0(x,u;\theta), \tag{2a}$$

$$\text{subject to:} \quad f_i(x,u;\theta) \leq 0, \quad i = 1,\dots,k, \tag{2b}$$

$$g_i(x,u;\theta) = 0, \quad i = 1,\dots,r, \tag{2c}$$

where $f_0 : \mathbb{R}^n \times \mathbb{R}^m \times \mathbb{R}^p \to \mathbb{R}$ is the objective function, $f_i : \mathbb{R}^n \times \mathbb{R}^m \times \mathbb{R}^p \to \mathbb{R}$ are the inequality constraints for all $i = 1,\dots,k$, and $g_i : \mathbb{R}^n \times \mathbb{R}^m \times \mathbb{R}^p \to \mathbb{R}$ are the equality constraints for all $i = 1,\dots,r$. Specific examples of policies of the form (2) are provided in Section III. The main idea behind this structure is that (2a) can act as some type of (approximate) value function, (2b) can capture critical safety/actuator limitations, and (2c) can require the state and inputs to satisfy physical laws. Another key advantage of the structure (2) is that prior knowledge can be easily built into the policy through proper choice of the functions.

The rest of the tutorial is organized as follows. In Section II, we introduce the controller tuning (or learning) problem that is the focus of this tutorial. In Section III, we discuss some important, commonly used forms of interpretable optimization-based control policies. In Section IV, we provide a detailed overview of how Bayesian optimization, a particularly efficient DFO method, can be used to effectively tackle controller auto-tuning problems. In Section V, we apply Bayesian optimization for auto-tuning a model predictive control policy on a benchmark system of oscillating masses. Finally, Section VI concludes the paper by providing some perspectives for future work.

II. The Automated Controller Tuning Problem

A. System Dynamics

In this work, we consider (possibly) nonlinear systems with dynamics given by

$$x_{t+1} = f(x_t, u_t, w_t), \quad t = 0, 1, \dots, \tag{3}$$

where $x_t \in \mathbb{R}^n$, $u_t \in \mathbb{R}^m$, $w_t \in \mathbb{R}^q$ are the state, control input, and disturbance at time period t, respectively, and $f : \mathbb{R}^n \times \mathbb{R}^m \times \mathbb{R}^q \to \mathbb{R}^n$ is the state transition function. The initial condition x_0 and the disturbances $\{w_t\}_{t\geq 0}$ are assumed to be random variables. In the traditional stochastic optimal control literature, it is often assumed that $x_0, w_0, w_1, \dots$ are independent random variables, with the disturbance sequence being identically distributed, and the dynamics f are known. We do not make such assumptions here, as they are often unrealistic in practical applications.

B. Closed-loop Performance Metrics

By combining the dynamics (3) and the control policy (2), the state $x_0, x_1, \dots$ and input $u_0, u_1, \dots$ trajectories form a stochastic process; the only requirement we will have throughout this work is the ability to generate independent samples from this stochastic process. This means that we do not need exact knowledge of the dynamics or the distribution of the disturbances, i.e., the true system is treated as a black-box, data-generating process. To judge the performance of a given control policy (specifically, a choice of parameter $\theta \in \Theta$), we define some closed-loop performance measure over finite-time trajectories of length T. The choice of T is very application-dependent and, for continuous-time applications, should be large enough such that an average over T steps is close to the long-term average. For notational simplicity, we will denote the trajectories over this horizon as

$$X = (x_0, x_1, \dots, x_T) \in \mathbb{R}^N, \tag{4a}$$

$$U = (u_0, x_1, \dots, u_T) \in \mathbb{R}^M, \tag{4b}$$

$$W = (w_0, w_1, \dots, w_T) \in \mathbb{R}^Q, \tag{4c}$$

where $N = n(T+1)$, $M = m(T+1)$, and $Q = q(T+1)$ are the dimensions of the state, input, and disturbance trajectories, respectively. The trajectories X, U, and W are random variables whose distributions depend on the choice of parameters θ (the explicit dependence on θ is not shown for notational simplicity).

The cost of any trajectory can then be defined by the evaluation of a function $\psi : \mathbb{R}^N \times \mathbb{R}^M \times \mathbb{R}^Q \to \mathbb{R} \cup \{+\infty\}$. Note that ψ must be specified by the user and depends on the closed-loop behavior of the process (with the control policy embedded). A control policy is then judged according to the expected value of this cost (with respect to the random initial condition and disturbances)

$$J(\theta) = \mathbf{E}\{\psi(X, U, W)\}. \tag{5}$$

Since the trajectories X, U, and W are fully parametrized by θ, the expected cost J only depends on θ. We briefly comment that ψ can take on many forms, though traditionally this function will be separable across time, with the form

$$\psi(X, U, W) = \frac{1}{T+1} \sum_{t=0}^{T} g(x_t, u_t, w_t), \tag{6}$$

where $g : \mathbb{R}^n \times \mathbb{R}^m \times \mathbb{R}^q \to \mathbb{R} \cup \{+\infty\}$ is the so-called stage cost function. This is not a requirement in this work.

C. Controller Tuning as Optimization

Given the performance indicator (5), we can compactly represent the controller tuning problem as solving the following optimization problem

$$\min_{\theta\in\Theta} J(\theta). \tag{7}$$

However, it is important to recognize that in general we cannot tackle the tuning problem (7) with traditional optimization methods (e.g., gradient descent) due to the features of J. First, the mathematical structure of J will not be known in closed-form since the optimization-based policy is defined implicitly by (2). This problem is further complicated by the fact that the dynamics f are generally unknown, which prevents the calculation of derivatives $\nabla_\theta J(\theta)$. Second, the expectation operator cannot be exactly computed in general. Instead, under the assumption that we can generate independent samples from the (joint) initial state and disturbance distribution, we can compute a Monte Carlo approximation of J as follows

$$\hat{J}(\theta) = \frac{1}{K}\sum_{i=1}^{K} \psi(X^i, U^i, W^i), \tag{8}$$

where $\{(X^i, U^i, W^i)\}_{i=1}^K$ denotes a collection of K independent trajectory samples. Third, evaluating the cost approximation (8) is expensive since it involves running K closed-loop simulations, which requires solving $K(T+1)$ optimization problems of the form (2) in order to evaluate u_t^i for all $t = 0, \dots, T$ and $i = 1, \dots, K$.

We note that the estimator $\hat{J}(\theta)$ is an unbiased approximation of $J(\theta)$, i.e., $J(\theta) = \mathbf{E}\{\hat{J}(\theta)\}$ for any $K \geq 1$ where the expectation is taken over the random samples. The quality of the approximation improves as K increases since $\mathbf{Var}\{\hat{J}(\theta)\} = \frac{1}{K}\mathbf{Var}\{\psi(X,U,W)\}$ where $\mathbf{Var}\{\cdot\}$ denotes variance, which goes to 0 as $K \to \infty$ [9].

D. Incorporation of Safety Constraints

An important consideration is how safety constraints can be incorporated into the auto-tuning problem (7). Technically, one can use infinite values for ψ to encode constraints on trajectories; however, this can create several problems in practice, including the inability to guarantee satisfaction of these constraints during the process of tuning θ. Constraint satisfaction is particularly important in safety-critical applications, wherein violating constraints can lead to dangerous and undesirable outcomes. Due to the stochastic nature of (3), we assume that safety constraints take the form of chance constraints [10]

$$\mathbf{Pr}\{\varphi(X,U,W) \geq 0\} \geq 1 - \epsilon, \tag{9}$$

where $\varphi : \mathbb{R}^N \times \mathbb{R}^M \times \mathbb{R}^Q \to \mathbb{R}$ is a function that represents if a trajectory is safe (≥ 0) or unsafe (< 0) and $\epsilon \in [0,1]$ is the allowed probability of violation. By defining $c(\theta) = \mathbf{Pr}\{\varphi(X,U,W) \geq 0\} - 1 + \epsilon$, we can compactly write the safe controller auto-tuning problem as

$$\min_{\theta\in\Theta} \{J(\theta) \;\text{ subject to }\; c(\theta) \geq 0\}, \tag{10}$$

which is an extension of (7) to incorporate safety constraints in addition to performance. The safety function c suffers from the exact same challenges as the performance function J, but it can similarly be approximated with Monte Carlo sampling

$$\hat{c}(\theta) = \frac{1}{K}\sum_{i=1}^{K} \mathbf{1}_{[0,\infty)}(\varphi(X^i, U^i, W^i)) - 1 + \epsilon, \tag{11}$$

where $\mathbf{1}_A(x)$ denotes the indicator function on set A (equal to 1 when $x \in A$ and 0 otherwise). As such, we can estimate the desired performance and safety metrics from the same set of K closed-loop simulations. In this tutorial, we mainly focus on methods that can solve the safe controller tuning problem (10).

III. Examples of Tunable Optimization-based Control Policies

We briefly describe specific forms of interpretable control policies (2) in this section. This list is not comprehensive – it is merely meant to make the abstract form of the optimization problem (2) more concrete.

A. Linear Quadratic Regulator

It is well-known that, when the random variables $x_0, w_0, w_1, \dots$ are independent, no explicit constraints are considered, and the overall cost is separable in time (6), then the optimal policy for $T \to \infty$ that minimizes J over all possible state feedback policies has the following form [11]

$$\pi(x) = \operatorname*{argmin}_u \; \mathbf{E}\{g(x,u,w) + V(f(x,u,w))\}, \tag{12}$$

where $V : \mathbb{R}^n \to \mathbb{R}$ is the optimal cost-to-go value function, which can be derived from dynamic programming (DP). Notice how no parameter θ appears in this expression since we have assumed access to the exact value function V. Unfortunately, the construction of the optimal value function is very difficult in general and can only be derived for a limited number of special cases. The most well-known case is often referred to as the linear quadratic regulator (LQR), which further assumes linear dynamics and quadratic stage cost

$$f(x,u,w) = Ax + Bu + w, \tag{13a}$$
$$g(x,u,w) = x^\top Q x + u^\top R u, \tag{13b}$$

where $A \in \mathbb{R}^{n\times n}$, $B \in \mathbb{R}^{n\times m}$, $Q \in \mathbb{S}_+^n$ (set of $n \times n$ symmetric positive semidefinite matrices), $\mathbb{S}_{++}^n$ (set of $n \times n$ symmetric positive definite matrices), and $w \in \mathcal{N}(0, \Sigma_w)$. The value function in this case can be analytically computed to be $V(x) = x^\top P x$ for a specific $P \in \mathbb{S}_+^n$ that solves the discrete-time algebraic Riccati equation [12].

To compute P, however, one requires exact knowledge of the system matrices A and B. Assuming we do not know these values, we can express the LQR policy in the form of (2) by selecting the following objective parametrization

$$f_0(x,u;\theta) = \begin{bmatrix} x \\ u \end{bmatrix}^\top \begin{bmatrix} Q + \theta_x^\top \theta_x & \theta_x^\top \theta_u \\ \theta_u^\top \theta_x & R + \theta_u^\top \theta_u \end{bmatrix} \begin{bmatrix} x \\ u \end{bmatrix}, \tag{14}$$

where $\theta = (\theta_x, \theta_u)$ represent the tunable parameters with $\theta_x \in \mathbb{R}^{n\times n}$ and $\theta_u \in \mathbb{R}^{n\times m}$. One can show that (2) with an objective (14) is equivalent to the LQR solution whenever $\theta_u = BP^{1/2}$ and $\theta_x = AP^{1/2}$.

B. Approximate Dynamic Programming

Approximate DP (ADP) policies take the form [13]

$$\pi(x) = \underset{u}{\operatorname{argmin}} \ \mathbf{E}\{g(x,u,w) + \hat{V}(f(x,u,w))\}, \quad (15)$$

where $\hat{V}$ is an approximation of the true value function. Policies of the form (15) are sometimes referred to as control Lyapunov functions (CLFs) [14]. By developing a suitable parametrization of $\hat{V}$ and the dynamics f, this policy can be written in the form of (2). Interested readers are referred to, e.g., [8], [15] for examples. It is also worth noting that hard input constraints $u \in \mathcal{U}$ (almost always present in practice) can be easily incorporated into (15), and it will remain a special case of (2) through proper selection of f_i. Furthermore, although these constraints may increase the online cost of evaluating $\pi(x;\theta)$, it does not change the dimensionality of (10).

C. Model Predictive Control

Model predictive control (MPC) has become the prime technology for achieving high-performance control of constrained multivariable systems [16], [17], which has diverse applications in chemical, energy, manufacturing, automotive, and robotic systems [18]. A standard formulation for nominal (or, certainty equivalence) MPC is given by

$$\begin{aligned}
\pi(x) = \underset{u_{0|t}}{\operatorname{argmin}} \ & \sum_{k=0}^{H-1} g(x_{k|t}, u_{k|t}, \hat{w}_{k|t}) + g_H(x_{H|t}), && \text{(16a)}\\
\text{s.t. } & x_{k+1|t} = f(x_{k|t}, u_{k|t}, \hat{w}_{k|t}), && \text{(16b)}\\
& x_{0|t} = x, && \text{(16c)}\\
& (x_{k|t}, u_{k|t}) \in \mathcal{Z}, && \text{(16d)}\\
& x_{H|t} \in \mathcal{X}_H, \quad \forall k = 0,\dots,H-1, && \text{(16e)}
\end{aligned}$$

where $H \geq 1$ is the prediction horizon; $x_{k|t}$, $u_{k|t}$, and $\hat{w}_{k|t}$ are, respectively, the predicted state, input, and disturbance values k steps ahead of current time t; $\mathcal{Z} \subset \mathbb{R}^{n+m}$ is the set of mixed state and input constraints (often decomposes to the form $\mathcal{Z} = \mathcal{X} \times \mathcal{U}$); and $g_H : \mathbb{R}^n \to \mathbb{R}$ and $\mathcal{X}_H$ are the terminal cost and constraints, respectively. We refer to this as the "nominal" formulation since $\hat{w}_{0|t}, \dots, \hat{w}_{H-1|t}$ represents a single predicted disturbance sequence. The initial state x acts as a parameter in MPC, and is a key source of "feedback" from the true system. Although the optimization problem (16) has variables $u_{0|t}, \dots, u_{H-1|t}$ and $x_{0|t}, \dots, x_{H|t}$, the argmin is taken over $u_{0|t}$ since MPC only applies the first input to the system at every time step.

When chosen as an invariant set for the predicted dynamics, $\mathcal{X}_H$ guarantees that the constrained optimization problem (16) is recursively feasible. Additionally, the terminal cost g_H can be chosen to ensure closed-loop stability whenever the dynamics and stage cost satisfy certain assumptions; see [17, Chapter 2] for a comprehensive discussion on feasibility and stability of MPC. These issues are not as important in our setting since we assume the true dynamics are unknown, so we are unable to construct these sets without gathering more data from the system.

Nearly all components of the MPC problem (16) can be treated as a tuning parameter, including

$$\{H, g, g_H, f, \hat{w}_{0|k}, \dots, \hat{w}_{H-1|k}, \mathcal{Z}, \mathcal{X}_H\}. \quad (17)$$

In practice, however, one desires to incorporate as much prior knowledge as possible to reduce the complexity of the auto-tuning problem (keep p as low as possible). It is important to note that any parameter that gets introduced in the implementation of (16) can be added to θ, including parameters appearing in the algorithm used to (approximately) solve (16) (e.g., tolerance values, scaling factors, initial guesses). Finally, even though we focused on the nominal MPC formulation, the ideas presented in the next section can be directly applied to more advanced robust and stochastic MPC formulations (see [6, Section 3] for examples and key references).

IV. Efficient Derivative-Free Optimization Methods for Auto-Tuning Applications

A. Zeroth- versus First-order Optimization Methods

Solving the controller tuning problem (7) is very hard in general due to its expensive black-box nature so that we must resort to approximate solution methods in practice. Recent work has discussed the value of gradient-based methods [8] that start with an initial guess for the parameters θ_0 that is updated, at each iteration k, by simulating the closed-loop system and computing an unbiased stochastic gradient of J, e.g., $g_k = \nabla_\theta \hat{J}(\theta_k)$, and updating the parameters via $\theta_{k+1} = \Pi_\Theta(\theta_k - \alpha_k g_k)$ where $\Pi_\Theta(\theta)$ denotes the Euclidean projection of θ onto set Θ and $\alpha_k > 0$ is a step size. The main challenge with this type of approach, however, is computing the gradient estimate g_k, which requires differentiating through the dynamics f, the cost ψ, and the policy π defined implicitly as the solution to an optimization problem (2). It is known that derivatives of (2) can be computed by differentiating through the KKT optimality conditions if the objective and constraint functions are smooth and satisfy some regularity conditions [19]; however, these requirements may not always be satisfied. An even bigger challenge is differentiating through the dynamics f, which cannot be done exactly unless we have perfect knowledge of this function. Two other challenges with first-order methods in this context are: (i) they are local in nature and so can become stuck in suboptimal local optima, and (ii) they do not easily extend to the safe (constrained) setting in (10).

Due to these major challenges, we must look to an alternative class of methods to tackle the auto-tuning problem. A natural choice is so-called derivative-free optimization (DFO) methods, which have become increasingly popular due to their general applicability; they make little-to-no assumptions on the objective or constraint functions. Even though DFO methods may not as effectively scale as the number of tunable parameters p becomes very large compared to the

first-order methods, it is important to recognize that p is expected to be small for interpretable control policies that take advantage of prior knowledge to effectively parametrize the objective and constraint functions. Therefore, we have in essence traded a more complex black-box policy, e.g., (1) (with a higher degree of representation power) for a "physics-informed" interpretable policy, which makes the auto-tuning problem more manageable in the sense that it requires optimization over a significantly lower-dimensional space. The latter set of problems are particularly amenable to DFO, which we discuss next.

B. Why Bayesian Optimization for Auto-Tuning?

There has been a significant amount of work on DFO, with a comprehensive survey of methods provided in [20], [21]. Some specific examples include random search [22], mesh adaptive direct search (MADS) [23], generalized pattern search (GPS) [24], genetic algorithms [25], particle swarm optimization (PSO) [26], and covariance matrix adaptation evolution strategy (CMA-ES) [27]. A key limitation of many DFO methods is that they require a large number of function evaluations to find the optimum – this is not a problem when the function is cheap to evaluate (i.e., can be efficiently evaluated several thousands to millions of times). Unfortunately, controller auto-tuning problems do not satisfy this requirement since estimating J (and c) is noisy and expensive, as discussed in Section II-C. This implies that extreme care should be taken when designing the next tuning parameter to test since we only have a limited number of attempts. In addition, if there are safety requirements that must be satisfied at each iteration, we must only test tuning parameter values that will not violate constraints.

Bayesian optimization (BO) is a class of machine learning-based optimization methods specifically designed for noisy and expensive functions [28]–[30]. As such, it has surpassed human and state-of-the-art algorithmic performance on a variety of real-world applications, including controller auto-tuning for a variety of different types of policies [31]–[35]. In addition, black-box constraints can be incorporated into BO to guarantee a high probability of safety during the optimization process [36]–[38]. Therefore, due to its data efficiency and ability to accommodate safety constraints, we will exclusively focus on BO for the rest of the paper. We provide an overview of (safe) BO next.

C. Overview of (Safe) Bayesian Optimization

BO effectively translates optimization tasks into learning tasks, which allows one to take advantage of the full data history when designing the next set of simulation or experimental conditions. This is the key advantage of BO compared to other competing DFO methods that use a small window of recent function observations. We will first focus on the auto-tuning problem in (7) and then show how this method extends to the safe version of the problem in (10).

A BO method consists of two components: (i) a predictive model given by a Bayesian posterior distribution over J, serving as a surrogate equipped with uncertainty estimates, and (ii) an acquisition function that depends on the posterior distribution of J, whose value at any $\theta \in \Theta$ quantifies the benefit of evaluating the objective function at this point.

1) Predictive model: The first major component of BO is the predictive model defined by a Bayesian posterior distribution over J. There are several examples of probabilistic surrogate models that satisfy this requirement, including random forests [39], Bayesian neural networks [40], and Gaussian processes (GP) [41]. GPs are the most commonly used class of models in BO due to their computationally tractability and non-parametric nature. The phrase "non-parametric" refers to the fact that they do not assume the function belongs to a finitely parametrized space such as the space of polynomials up to a certain degree – a consequence of this property is that GPs can learn any function satisfying certain continuity/smoothness properties [41].

A GP generalizes the notion of "distributions over variables" to "distributions over functions." It is fully specified by its prior mean function $\mu_0 : \Theta \to \mathbb{R}$ and covariance $k_0 : \Theta \times \Theta \to \mathbb{R}$ functions. Given a prior GP distribution for J and a set of s noisy evaluations $\mathscr{D}_s = \{(\theta_i, y_i)\}_{i=1}^s$, where $y_i = \hat{J}(\theta_i) = J(\theta_i) + \varepsilon_i$ and $\varepsilon_1, \dots, \varepsilon_s$ are independent and normally distributed with zero mean and variance σ^2, the posterior distribution for J remains a GP with the following analytic mean and covariance function expressions

$$\begin{aligned} \mu_s(\theta) &= \mu_0(\theta) + \boldsymbol{k}_0^\top(\theta) \left(\boldsymbol{K}_s + \sigma^2 I_s\right)^{-1} \tilde{\boldsymbol{y}}_s, \\ k_s(\theta, \theta') &= k_0(\theta, \theta') - \boldsymbol{k}_0^\top(\theta) \left(\boldsymbol{K}_s + \sigma^2 I_s\right)^{-1} \boldsymbol{k}_0(\theta), \end{aligned} \tag{18}$$

where $\tilde{\boldsymbol{y}}_s = (y_1 - \mu_0(\theta_1), \dots, y_s - \mu_0(\theta_s)) \in \mathbb{R}^s$, $\boldsymbol{k}_0(\theta) = (k_0(\theta, \theta_1), \dots, k_0(\theta, \theta_s)) \in \mathbb{R}^s$, $\boldsymbol{K}_s \in \mathbb{S}_{++}^s$ is the kernel matrix whose elements are given by $[\boldsymbol{K}_s]_{v,w} = k_0(x_v, x_w)$ for all $v, w \in \{1, \dots, s\}$, and I_s is the $s \times s$ identity matrix.

It is worth noting that the posterior mean function μ_s can be interpreted as a surrogate model for J, while the posterior covariance function k_s equips this surrogate with the ability to generate rigorous uncertainty estimates, which is a key advantage of GPs over deterministic models. These uncertainty estimates are often based on the posterior standard deviation of J, which can be calculated as $\sigma_s(\theta) = \sqrt{k_s(\theta, \theta)}$. Additionally, the prior mean and covariance function are often defined in terms of a set of hyperparameters that are not exactly known. In practice, these hyperparameters can be trained by maximizing the likelihood of the model predictions on the observation data $\mathscr{D}_s$. The detailed derivation of the likelihood function can be found in [41, Chapter 2]. The likelihood contains a model fit and regularization term such that the training process is robust to overfitting by construction.

2) Acquisition function: The second major component of BO is the choice of acquisition function $\alpha_s : \Theta \to \mathbb{R}$ (also called the expected utility), where the subscript s denotes its dependence on the posterior distribution of J shown in (18). The choice of α_s is completely open to the user and, when properly selected, the value of $\alpha_s(\theta)$ at any particular $\theta \in \Theta$ should be a good measure of the (expected) benefit in querying J at this point in the future. According to this

definition, one looks to preferentially sample at the point that produces the highest possible value of the acquisition function. We then formally define a BO method as the sequential learning process of selecting the next point $\theta_{s+1} \in \operatorname{argmax}_{\theta\in\Theta} \alpha_s(\theta)$ as the maximizer of α_s. Note that, unlike J, α_s is often available in closed-form such that it is much simpler to optimize than J itself.

Several acquisition functions have been proposed, including expected improvement (EI) [42], lower confidence bound (LCB) [43], knowledge gradient (KG) [44], Thompson sampling [45], and entropy search [46]. The most common way to derive an acquisition function is through the selection of a reward (or utility) function $r(\mathscr{D}_s)$ that depends on the collected data $\mathscr{D}_s$. The acquisition function then corresponds to the expected increase in the reward, i.e.,

$$\alpha_s(\theta) = \mathbf{E}_s \left\{ r(\mathscr{D}_s \cup \{\theta, J(\theta)\}) - r(\mathscr{D}_s) \right\}, \tag{19}$$

where $\mathbf{E}_s$ refers to the expected value under the posterior distribution at iteration s. For example, EI corresponds to a reward function defined in terms of the observed sample $r(\mathscr{D}_s) = -J_s^\star = -\min_{(\theta,y)\in\mathcal{D}_s} y = -\min_{i=1,\dots,s} y_i$; the negative sign represents the fact that lower values increase the reward in the case of a minimization problem. In this case, we can simplify (19) as follows

$$\begin{aligned} \mathrm{EI}_s(\theta) &= \mathbf{E}_s\{J_{s+1}^\star - J_s^\star\}, \\ &= \mathbf{E}_s\{\max\{0, J_s^\star - y_{s+1}\}\}, \\ &= \mathbf{E}_Z\{\max\{0, J_s^\star - \mu_s(\theta) - \sigma_s(\theta)Z\}\}, \\ &= \rho(J_s^\star - \mu_s(\theta), \sigma_s(\theta)), \end{aligned} \tag{20}$$

where

$$\rho(y,s) = \begin{cases} y\Phi(y/s) + s\phi(y/s), & s \geq 0, \\ \max\{y, 0\} & s = 0, \end{cases} \tag{21}$$

and Φ and ϕ are, respectively, the standard normal cumulative distribution function and probability density function. The third line in (20) follows from the fact that our prediction of the next sample $J(\theta_{s+1})|\mathscr{D}_s$ is normally distributed such that it can be written as $J(\theta_{s+1})|\mathscr{D}_s = \mu_s(\theta) + \sigma_s(\theta)Z$ where $Z \sim \mathcal{N}(0,1)$. The final line then follows by solving for the expectation with respect to Z using integration by parts.

A standard implementation of BO, for a chosen acquisition function, is summarized in Algorithm 1. Although it is common to use a single acquisition function at every iteration, it is straightforward to modify Step 3 of this algorithm to perform some adaptation to select from a set of candidate solutions. One approach for doing this is to randomly sample α_s from a discrete set of options and updating the future probability of selection based on its performance in that iteration [47]. An alternative approach is to solve a multi-objective optimization (MOO) problem that constructs a Pareto frontier (tradeoff curve) between multiple acquisition functions [48]. This can be interpreted as finding a point θ_{s+1} for which there is a reasonable consensus among these different reward measures. The derivation of new acquisition functions, as well as strategies for selecting among them, is still an active area of research in BO and further research is needed to understand the impact of these choices in the context of controller auto-tuning problems.

Algorithm 1 The Bayesian Optimization (BO) Framework

Require: Domain Ω, initial data $\mathscr{D}_0$, prior GP mean μ_0 and kernel k_0 function, and choice of acquisition function

1: **for** $s = 0, 1, \dots$ **do**
2: Construct GP posterior for J given $\mathscr{D}_s$ using (18).
3: Maximize acquisition $\theta_{s+1} \leftarrow \operatorname{argmax}_{\theta\in\Theta} \alpha_s(\theta)$
4: Query at θ_{s+1} and observe cost $y_{s+1} = \hat{J}(\theta_{s+1})$
5: Update data $\mathscr{D}_{s+1} \leftarrow \mathscr{D}_s \cup \{\theta_{s+1}, y_{s+1}\}$
6: **end for**

3) Safety considerations: As discussed in the previous section, we must explicitly incorporate black-box constraints, as in (10), to be able to address safety considerations when selecting our next sample θ_{s+1}. If one tries to encode constraints by assigning infinite values to the cost, then J will no longer satisfy the GP modeling assumptions and, thus, we cannot construct an accurate surrogate model using (18). Instead, we can explicitly model the safety constraint function c using a separate (independent) GP model, just as we did for the cost function J. Let $\mathscr{D}_s^c$ denote a set of s noisy constraint evaluations. Then, given a GP prior with mean μ_0^c and covariance k_0^c functions for c, we can construct a GP posterior function similarly to (18), i.e.,

$$c(\theta)|\mathscr{D}_s^c \sim \mathcal{GP}(\mu_s^c(\theta), k_s^c(\theta, \theta')). \tag{22}$$

Constraint satisfaction can be achieved with high probability as long as the GP model (22) is sufficiently well-calibrated, meaning it satisfies the following definition.

Definition 1: The GP model of the safety constraints (22) is said to be well-calibrated if the inequality

$$|c(\theta) - \mu_s^c(\theta)| \leq \sqrt{\beta_{s+1}}\sigma_s^c(\theta), \quad \forall\theta \in \Theta, \forall s \geq 0, \tag{23}$$

holds with probability at least $1-\delta$ for some $\delta \in (0,1)$.

That is, if (22) is well calibrated, then the confidence interval of the GP contains the true unknown function c with high probability for all θ and for all iterations s. As shown in [49, Theorem 2], one can select parameters $\{\beta_{s+1}\}_{s\geq 0}$ that ensure the well-calibrated model assumption is satisfied as long as c has a bounded reproducing kernel Hilbert space (RKHS) norm, which is a relatively mild assumption. Letting $\mathcal{S} = \{\theta \in \Theta : c(\theta) \geq 0\}$ denote the set of safe controller tuning parameters, we can construct a high-probability inner approximation of $\mathcal{S}$ as follows

$$\hat{\mathcal{S}}_s = \{\theta \in \Theta : \mu_s^c(\theta) - \sqrt{\beta_{s+1}}\sigma_s^c(\theta) \geq 0\}. \tag{24}$$

In other words, we can think of $\hat{\mathcal{S}}_s$ as a partially-revealed safe set that must satisfy $\mathbf{Pr}\{\hat{\mathcal{S}}_s \subseteq \mathcal{S}, \forall s \geq 0\} \geq 1-\delta$. Note that $\hat{\mathcal{S}}_s \neq \emptyset$ as long as at least one safe set of tuning parameters $\theta_0 \in \mathcal{S}$ is known, which can often be obtained from high-fidelity simulations and/or domain knowledge.

Given the partially-revealed safe set, one can easily modify Algorithm 1 to be safe by replacing the set Θ in Line 3 with $\hat{\mathcal{S}}_s$, which guarantees $\theta_{s+1} \in \hat{\mathcal{S}}_s$ for all $s \geq 0$, as

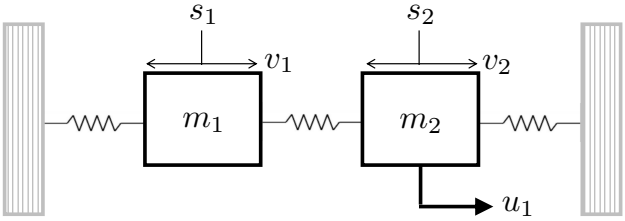


Fig. 1: Illustration of system of two oscillating masses.

shown in [38]. To reduce the complexity of the optimization procedure, one can use a log-barrier penalty function instead, which will not compromise safety guarantees. A recently identified challenge with this type of safe BO approach, however, is that it does not directly incentivize enlarging $\hat{\mathcal{S}}_s$ such that it has the potential to lead to sub-optimal performance. One way to overcome this challenge is presented in [50], which provides a practical algorithm for detecting when this behavior is likely to occur and then switching to an alternative series of relaxation and projection steps. There remain several interesting directions for future work on how to optimally tackle safe BO problems; however, the promising results obtained so far with straightforward extensions of traditional BO highlight the power of this type of framework for controller auto-tuning problems.

V. Illustrative Case Study: Learning MPC Policy Parameters with Closed-loop Data

In this section, we demonstrate some of the key advantages of the BO framework for controller auto-tuning on an illustrative example. The control policy was implemented using the do-mpc toolbox [51], and the BO implementation was done using the BOTorch package [52] with default settings. We describe the system dynamics, the (interpretable) MPC control policy, and the simulation results.

A. System Description

We consider a system of two horizontally oscillating masses interconnected via a spring. Each mass is connected via a spring to a wall, where the applied force to the second mass can be manipulated, as depicted in Fig. 1. The states of the system are the two positions, s_1 and s_2, and two velocities, v_1 and v_2, of the masses while the control input is the applied force u_1 to the second mass. The dynamics of this system are described by a model of the form (3)

$$f(x,u,w) = Ax + Bu, \tag{25}$$

where

$$A = \begin{bmatrix} 0.763 & 0.460 & 0.115 & 0.020 \\ -0.899 & 0.763 & 0.420 & 0.115 \\ 0.115 & 0.020 & 0.763 & 0.460 \\ 0.420 & 0.115 & -0.899 & 0.763 \end{bmatrix}, \; B = \begin{bmatrix} 0.014 \\ 0.063 \\ 0.221 \\ 0.367 \end{bmatrix},$$

and $x = (s_1, v_1, s_2, v_2)$ and $u = u_1$. Note that the units of s_1 and s_2 are in meters, v_1 and v_2 are in meters per second, and u_1 is Newton. The sampling time used to generate this discrete-time model is 0.5 seconds. We assume a closed-loop cost function of the form (6) with $T = 40$

$$\psi(X,U,W) = \textstyle\sum_{t=0}^{T} \|x_t\|_2^2. \tag{26}$$

We also assume that the states and control input must satisfy the following constraints

$$-4 \leq s_i \leq 4, \qquad i \in \{1,2\}, \tag{27a}$$
$$-10 \leq v_i \leq 10, \qquad i \in \{1,2\}, \tag{27b}$$
$$-0.5 \leq u_1 \leq 0.5. \tag{27c}$$

We could treat the state constraints on the position as "safety constraints," however, they are rarely active so we decided not to formally consider them here. As for the probability distribution of the initial conditions, we define the following single initial conditions (i.e., Dirac delta distribution)

$$x_0 = (0.293, 1.291, 0.617, 0.269). \tag{28}$$

B. MPC Control Policy

To control this oscillating mass system, we consider an MPC policy of the form (16). We use nearly the same stage cost function of $g(x,u,w) = \|x\|_2^2$; however, we also add a small input penalty term $10^{-4}\Delta u^2$ to the stage cost (where Δu is the difference between consecutive input values) to ensure smooth input profiles. We stress that the choice of ψ is completely open to the user and does not have to match what is used to derive the control policy. In addition, we only assume partial knowledge of the system dynamics, that is, only the matrix A is known. Therefore, we parametrize (16b) as $x_{k+1|t} = Ax_{k|t} + \theta u_{k|t}$, where $\theta \in \Theta = [0,1]^4$ are the tuning parameters.

To highlight the impact of θ on the cost, we plot the closed-loop trajectory for s_2 for the exact case $\theta = B$, a randomly chosen θ, and no control in Fig. 2. It is interesting to note that the randomly chosen incorrect θ value ends up destabilizing the system, illustrating the importance of the proper tuning of the unknown parameters.

C. Results and Performance Comparison

We start by comparing two different BO implementations with commonly used alternative DFO methods. In particular, we compare the EI and UCB acquisition functions, denoted by BO-EI and BO-UCB, respectively, to random search [22] and SnobFit [53]. Each optimizer is allowed a maximum budget of 60 closed-loop evaluations. The first 5 data points for both BO methods are generated using quasi-random sampling to ensure the hyperparameters of the GP prior are trainable. Since all methods involve some degree of random sampling, the optimization trials are repeated 50 times from different random seeds to assess their average performance. Performance of each method is measured in terms of *simple regret*, which is the difference between the best found sample up until a given iteration and the global minimum (estimated using the best found sample in all trials). The results are shown in Fig. 3. We clearly see that both BO-EI and BO-UCB consistently outperform the other DFO methods. Interestingly, the BO methods find tuning

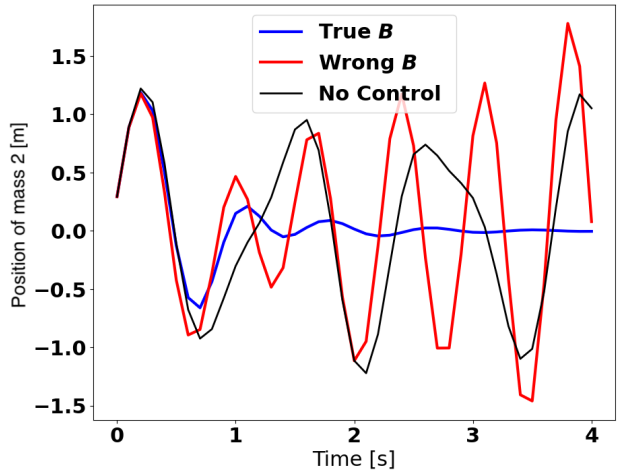


Fig. 2: Closed-loop trajectories for the most challenging state to control, s_2, using the exact model $\theta = B$ (true B), a randomly chosen tuning θ (wrong B), and no control.

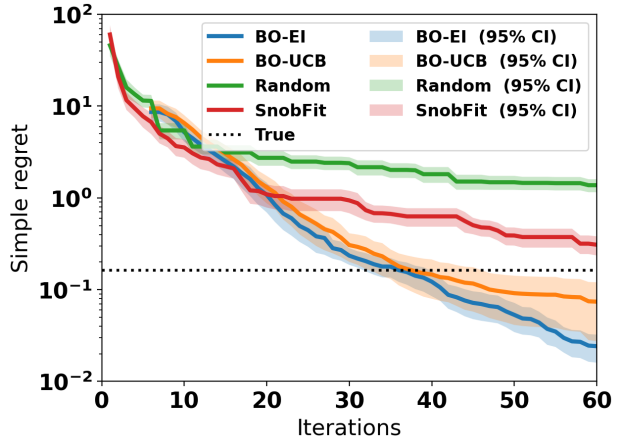


Fig. 3: Expected simple regret for the oscillating mass auto-tuning problem, with approximate 95% confidence regions, estimated from 50 independent random optimization trials. The cost obtained with the true $\theta = B$ is also shown with a dotted black line.

parameter values that lead to a lower cost than setting θ equal to the true B value. This is not as surprising when we recall the mismatch between the closed-loop cost and MPC cost functions. Since we have not penalized the smoothness of the control input profile in (26), BO has found a (somewhat non-intuitive) way to make the control actions more aggressive while simultaneously achieving good setpoint tracking performance. This highlights the importance of the choice of cost functions (and policy parametrization) when formulating the controller auto-tuning problem.

To further investigate these results, we analyze the performance of a single run of these algorithms, which lead to the following best learned tuning parameter values

$$\begin{aligned} B &= (0.014, 0.063, 0.221, 0.367), \\ \theta_{\text{BO-EI}} &= (0.000, 0.000, 0.338, 0.761), \\ \theta_{\text{random}} &= (0.105, 0.476, 0.262, 0.927), \\ \theta_{\text{snobfit}} &= (0.000, 0.094, 0.410, 0.733). \end{aligned}$$

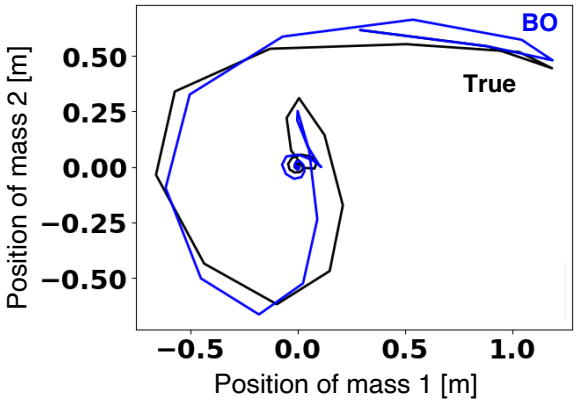
BO does not exactly learn the true B matrix, however, it clearly mimics the same trend as B wherein the first two elements are small and the fourth element is larger than the third element. This is a consequence of the interpretable policy structure, which allows useful information to be extracted from the learned θ values. In particular, here, we learned that, as expected, the control input has a small effect on the position and velocity of the first mass, but a large effect on the position and velocity of the second mass (see Fig. 1). To demonstrate how this type of knowledge can transfer to other tasks, we tested the performance of the BO-tuned controller on a new (harder) initial condition $x_0 = (2, -1, 0, -2)$. Phase plots of s_1 versus s_2 for the original (training) initial condition and the new (testing) initial condition are shown in Fig. 4. The system can be quickly steered to the origin in both cases – this type of extrapolation would be difficult to achieve in general without an interpretable control policy supplied with useful domain knowledge.

VI. Conclusions and Perspectives for Future Work

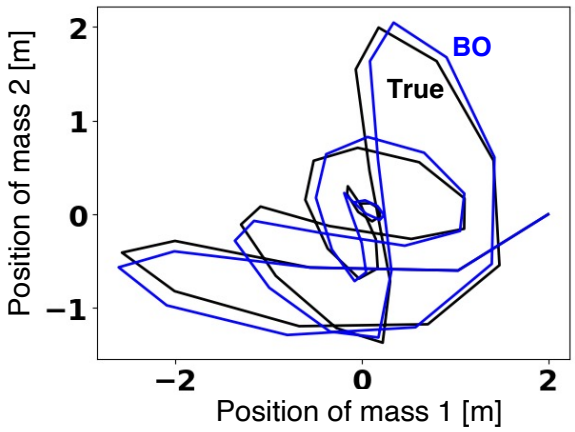
This tutorial provided an overview of recent developments in the area of control policy learning, with a particular focus on learning optimization-based control policies that are implicitly defined in terms interpretable equations, namely control objectives and system dynamics and constraints. In contrast to explicit control policies such as neural networks, these interpretable equations in fact enable us to encode prior system knowledge into a control policy with learnable parameters. We first demonstrated how some of the widely-used control strategies such as linear-quadratic regulator and model predictive control can be viewed as tunable optimization-based control policies, wherein automated and systematic policy tuning plays an important role to ensure high-performance and safe closed-loop behavior. Then, we discussed Bayesian optimization (BO) as an especially promising derivative-free optimization method for efficient solution of safe controller auto-tuning problems that rely on (often expensive) repeated closed-loop simulations of an optimization-based control policy.

Recent years have witnessed a growing interest in BO applications for controller auto-tuning. Yet, this fast evolving area presents significant opportunities for future research. Here, we discuss several extensions of traditional BO (Algorithm 1). We note that this list is by no means exhaustive, rather it is intended to provide a brief overview of several open challenges that have received less attention in the context of (safe) controller auto-tuning problems.

Noise handling. An important advantage of BO is its ability to handle noisy observations, which is intrinsic to



(a) training initial condition



(b) testing initial condition

Fig. 4: Comparison of closed-loop phase plots of s_1 versus s_2 for the original (training) initial condition in (a) and the new (testing) initial condition in (b) for the BO-based auto-tuned (blue) and true control (black) policies.

controller auto-tuning problems discussed in Section II. The GP presented in (18) is based on the simplest independent, fixed variance σ^2 Gaussian noise model. Although σ^2 is often treated as a trainable hyperparameter in the GP, which provides some degree of modeling flexibility, this type of noise model is only theoretically valid in the limit of large K in (8) wherein we can invoke the central limit theorem. Since the evaluation of cost J is proportional to K (see (8)), we may not be able to select a large enough value for K so that this assumption is accurate. There are two major directions that can be pursued to address this challenge. First, one can apply more sophisticated Monte Carlo sampling techniques (e.g., importance sampling [54]) to reduce the variance in the sample-based estimator such that it better fits the standard noise model. Second, one can incorporate more advanced noise models to better capture the observed variability. For example, heteroscedastic GP models can capture input-dependent variability in the noise variance [55]–[57]. There has been very limited work on these types of noise models in the context of BO-based controller auto-tuning [58].

Parallel evaluation. In many applications, one may have access to multiple copies of the system of interest. For high-fidelity simulators, these copies refer to multiple computers or cores that enable one to evaluate J simultaneously for different values of θ. For experimental systems, these copies would be physical "backup" systems available for testing and validation purposes. In either case, we are no longer limited to evaluating a single tuning parameter at every iteration. Assuming we can run B trials at every iteration, Line 3 of Algorithm 1 can be updated to simultaneously design B different sets of tuning parameters that can all be run in parallel in Line 4. In fact, BO can straightforwardly extend to this parallel evaluation case in both the synchronous [59]–[61] (evaluations take the same amount of time) and asynchronous [62]–[64] (evaluations take a variable amount of time) settings. Parallel (also known as batch) BO can lead to significant speedups in controller auto-tuning problems in terms of total optimization time.

Multi-fidelity representations. Strategies for improving upon the budget allocation in BO are highly desirable, especially when observations are expensive to acquire. Recent work has shown that, instead of thinking about J as a function of tuning parameters only, we can more generally represent it as a family of information sources $\mathcal{J}(\theta, \gamma)$ such that $J(\theta) = \mathcal{J}(\theta, \gamma = \mathbf{1})$ with $\gamma \in [0, 1]^{n_\gamma}$ denoting a collection of "fidelity" parameters that control the accuracy of the cost function. It is assumed the fidelity parameters have been scaled such that $\gamma = \mathbf{0}$ and $\gamma = \mathbf{1}$ correspond to the lowest and highest fidelity model, respectively. As discussed in detail in [65, Section 5], many quantities can be treated as fidelity parameters γ in controller auto-tuning applications. As such, increasing individual elements of γ is expected to increase the accuracy of the simulation, however, this increase in accuracy comes at the price of additional cost toward our budget. Therefore, given some fixed overall budget, we could attempt to simultaneously design $(\theta_{s+1}, \gamma_{s+1})$ to move us toward our goal of solving the (safe) auto-tuning problem without over-spending our budget. Multi-fidelity BO has been considered in several works [66]–[68] and applied to MPC auto-tuning in [65]. Note that the performance of these methods heavily depends on the chosen fidelity space. Additional theoretical and practical work is needed to better understand the impact of this choice on multi-fidelity BO performance, especially in auto-tuning applications. Moreover, there has been no work on extending multi-fidelity BO to handle constraints.

Composite functions. Traditional BO is fully black-box in nature, which implicitly neglects certain structural information that is often available. For example, the overall closed-loop cost function is often the sum of repeated individual terms defined in terms of a known stage cost function, as shown in (6). When we directly model J as a GP, this structure is completely neglected. This does provide some benefits in terms of reduced computational cost in terms of training and prediction; however, these benefits often come at the cost of accuracy, which can be significant in some

cases. A more general way to approach the controller auto-tuning problem is to treat the cost and constraint functions as composites of known and unknown functions, as shown in [69]. Whenever the known functions are linear, the added cost is fairly minor since the overall cost and constraint functions are still GPs (just with lower variance as proven in [70]). However, whenever these functions are nonlinear, we must resort to more advanced strategies to properly exploit this structure. A general algorithm for constrained BO of composite functions is presented in [71], which has shown significant performance improvements over black-box BO on a variety of problems. Applying and further specializing these types of composite BO methods to auto-tuning applications is a worthwhile direction for future research.

Gradient information. There has been a significant amount of recent work on BO strategies for exploiting derivative (gradient) information in the search process [72]–[74]. These methods can be interpreted as a hybrid between zeroth- and first-order methods, meaning they have the potential to achieve the fast local convergence properties of first-order methods, like (stochastic) gradient descent, along with the powerful global search properties of traditional BO. Even when the system dynamics are unknown, we can compute estimates of the gradient of the cost function from closed-loop trajectories using the policy gradient theorem [75]. As such, one can obtain significantly improved performance over standard reinforcement learning and BO auto-tuning methods with these hybrid methods, as demonstrated in [76]. Further work is needed to better understand the scalability of these methods and to extend them to handle safety constraints.

Multi-objective problems. The previous discussions focused on a single cost function, however, many auto-tuning applications involve multiple cost indicators such as those related to economics, setpoint tracking performance, and actuator use. In such cases, it is possible to formulate the auto-tuning problem as a multi-objective optimization (MOO) problem, which looks to systematically study the tradeoff between different cost functions through the construction of an optimal Pareto front. BO has been extended to MOO problems, e.g., in [77]–[79] and applied to an MPC auto-tuning problem in [80]. Additional work is needed to understand how best to incorporate safety considerations into the MOO framework, as well as how to best handle more than two cost functions.

Preference learning. An important assumption made throughout this paper is that we have access to a cost J that numerically quantifies the "best" controller tuning parameters. However, there are cases when such a cost function is not easily quantifiable in this form, either because it is qualitative in nature, it involves several goals, or requires a human decision maker to be assessed. An interesting direction that has recently emerged to address this limitation is so-called preference-based Bayesian optimization (PBO) [81]–[83], wherein one uses expressed preferences (pairwise comparisons between a set of evaluations) to learn an underlying GP surrogate preference function that can be used for optimization tasks. An efficient preference-based learning optimization approach was recently developed and applied to controller auto-tuning problems in [84]. Additional studies of the performance of such preference-based optimization methods, especially as the number of tuning parameters increases, would be valuable. Furthermore, combining preference learning with the previously-mentioned BO extensions (such as noise and parallel and multi-fidelity evaluations) are very interesting directions for future work.

REFERENCES

[1] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement learning: A survey," *Journal of Artificial Intelligence Research*, vol. 4, pp. 237–285, 1996.

[2] B. Recht, "A tour of reinforcement learning: The view from continuous control," *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 2, pp. 253–279, 2019.

[3] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. MIT Press, 2018.

[4] D. Bertsekas, "Newtons method for reinforcement learning and model predictive control," *Results in Control and Optimization*, vol. 7, p. 100121, 2022.

[5] D. Hein, S. Udluft, and T. A. Runkler, "Interpretable policies for reinforcement learning by genetic programming," *Engineering Applications of Artificial Intelligence*, vol. 76, pp. 158–169, 2018.

[6] A. Mesbah, K. P. Wabersich, A. P. Schoellig, M. N. Zeilinger, S. Lucia, T. A. Badgwell, and J. A. Paulson, "Fusion of machine learning and mpc under uncertainty: What advances are on the horizon?," in *Proceedings of the American Control Conference*, pp. 342–357, 2022.

[7] Y. Lu and J. Lu, "A universal approximation theorem of deep neural networks for expressing probability distributions," *Advances in Neural Information Processing Systems*, vol. 33, pp. 3094–3105, 2020.

[8] A. Agrawal, S. Barratt, S. Boyd, and B. Stellato, "Learning convex optimization control policies," in *Learning for Dynamics and Control*, pp. 361–373, 2020.

[9] W.-K. Mak, D. P. Morton, and R. K. Wood, "Monte Carlo bounding techniques for determining solution quality in stochastic programs," *Operations Research Letters*, vol. 24, no. 1-2, pp. 47–56, 1999.

[10] J. A. Paulson, E. A. Buehler, R. D. Braatz, and A. Mesbah, "Stochastic model predictive control with joint chance constraints," *International Journal of Control*, vol. 93, no. 1, pp. 126–139, 2020.

[11] D. Bertsekas, *Dynamic programming and optimal control: Volume I*, vol. 1. Athena Scientific, 2012.

[12] T. Pappas, A. Laub, and N. Sandell, "On the numerical solution of the discrete-time algebraic Riccati equation," *IEEE Transactions on Automatic Control*, vol. 25, no. 4, pp. 631–641, 1980.

[13] W. B. Powell, "What you should know about approximate dynamic programming," *Naval Research Logistics*, vol. 56, no. 3, pp. 239–249, 2009.

[14] Y. Wang and S. Boyd, "Fast evaluation of quadratic control-Lyapunov policy," *IEEE Transactions on Control Systems Technology*, vol. 19, no. 4, pp. 939–946, 2010.

[15] A. Keshavarz and S. Boyd, "Quadratic approximate dynamic programming for input-affine systems," *International Journal of Robust and Nonlinear Control*, vol. 24, no. 3, pp. 432–449, 2014.

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

[17] J. B. Rawlings, D. Q. Mayne, and M. Diehl, *Model predictive control: Theory, computation, and design*, vol. 2. Nob Hill Publishing Madison, WI, 2017.

[18] J. H. Lee, "Model predictive control: Review of the three decades of development," *International Journal of Control, Automation and Systems*, vol. 9, pp. 415–424, 2011.

[19] S. Barratt, "On the differentiability of the solution to convex optimization problems," *arXiv preprint arXiv:1804.05098*, 2018.

[20] A. R. Conn, K. Scheinberg, and L. N. Vicente, *Introduction to derivative-free optimization*. SIAM, 2009.

[21] L. M. Rios and N. V. Sahinidis, "Derivative-free optimization: a review of algorithms and comparison of software implementations," *Journal of Global Optimization*, vol. 56, pp. 1247–1293, 2013.

[22] F. J. Solis and R. J.-B. Wets, "Minimization by random search techniques," *Mathematics of Operations Research*, vol. 6, no. 1, pp. 19–30, 1981.

[23] C. Audet and J. E. Dennis Jr, "Mesh adaptive direct search algorithms for constrained optimization," *SIAM Journal on Optimization*, vol. 17, no. 1, pp. 188–217, 2006.

[24] T. G. Kolda, R. M. Lewis, and V. Torczon, "Optimization by direct search: New perspectives on some classical and modern methods," *SIAM Review*, vol. 45, no. 3, pp. 385–482, 2003.

[25] D. M. Mukhopadhyay, M. O. Balitanas, A. Farkhod, S.-H. Jeon, and D. Bhattacharyya, "Genetic algorithm: A tutorial review," *International Journal of Grid and Distributed Computing*, vol. 2, no. 3, pp. 25–32, 2009.

[26] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Proceedings of International Conference on Neural Networks*, vol. 4, pp. 1942–1948, IEEE, 1995.

[27] N. Hansen, S. D. Müller, and P. Koumoutsakos, "Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (CMA-ES)," *Evolutionary Computation*, vol. 11, no. 1, pp. 1–18, 2003.

[28] B. Shahriari, K. Swersky, Z. Wang, R. P. Adams, and N. De Freitas, "Taking the human out of the loop: A review of Bayesian optimization," *Proceedings of the IEEE*, vol. 104, no. 1, pp. 148–175, 2015.

[29] P. I. Frazier, "A tutorial on Bayesian optimization," *arXiv preprint arXiv:1807.02811*, 2018.

[30] S. Greenhill, S. Rana, S. Gupta, P. Vellanki, and S. Venkatesh, "Bayesian optimization for adaptive experimental design: A review," *IEEE Access*, vol. 8, pp. 13937–13948, 2020.

[31] M. Neumann-Brosig, A. Marco, D. Schwarzmann, and S. Trimpe, "Data-efficient autotuning with Bayesian optimization: An industrial control study," *IEEE Transactions on Control Systems Technology*, vol. 28, no. 3, pp. 730–740, 2019.

[32] D. Piga, M. Forgione, S. Formentin, and A. Bemporad, "Performance-oriented model learning for data-driven mpc design," *IEEE Control Systems Letters*, vol. 3, no. 3, pp. 577–582, 2019.

[33] J. A. Paulson and A. Mesbah, "Data-driven scenario optimization for automated controller tuning with probabilistic performance guarantees," *IEEE Control Systems Letters*, vol. 5, no. 4, pp. 1477–1482, 2020.

[34] Q. Lu, R. Kumar, and V. M. Zavala, "MPC controller tuning using Bayesian optimization techniques," *arXiv preprint arXiv:2009.14175*, 2020.

[35] M. Khosravi, V. N. Behrunani, P. Myszkorowski, R. S. Smith, A. Rupenyan, and J. Lygeros, "Performance-driven cascade controller tuning with Bayesian optimization," *IEEE Transactions on Industrial Electronics*, vol. 69, no. 1, pp. 1032–1042, 2021.

[36] F. Berkenkamp, A. Krause, and A. P. Schoellig, "Bayesian optimization with safety constraints: Safe and automatic parameter tuning in robotics," *Machine Learning*, pp. 1–35, 2021.

[37] M. Khosravi, C. König, M. Maier, R. S. Smith, J. Lygeros, and A. Rupenyan, "Safety-aware cascade controller tuning using constrained Bayesian optimization," *IEEE Transactions on Industrial Electronics*, vol. 70, no. 2, pp. 2128–2138, 2022.

[38] D. Krishnamoorthy and F. J. Doyle, "Safe bayesian optimization using interior-point method - applied to personalized insulin dose guidance," *IEEE Control Systems Letters*, vol. 6, pp. 2834–2839, 2022.

[39] F. Hutter, H. H. Hoos, and K. Leyton-Brown, "Sequential model-based optimization for general algorithm configuration," in *Learning and Intelligent Optimization*, pp. 507–523, Springer, 2011.

[40] J. Snoek, O. Rippel, K. Swersky, R. Kiros, N. Satish, N. Sundaram, M. Patwary, M. Prabhat, and R. Adams, "Scalable Bayesian optimization using deep neural networks," in *International Conference on Machine Learning*, pp. 2171–2180, PMLR, 2015.

[41] C. K. Williams and C. E. Rasmussen, *Gaussian Processes for Machine Learning*, vol. 2. MIT Press, 2006.

[42] D. R. Jones, M. Schonlau, and W. J. Welch, "Efficient global optimization of expensive black-box functions," *Journal of Global optimization*, vol. 13, no. 4, p. 455, 1998.

[43] N. Srinivas, A. Krause, S. M. Kakade, and M. W. Seeger, "Information-theoretic regret bounds for Gaussian process optimization in the bandit setting," *IEEE Transactions on Information Theory*, vol. 58, no. 5, pp. 3250–3265, 2012.

[44] P. I. Frazier, W. B. Powell, and S. Dayanik, "A knowledge-gradient policy for sequential information collection," *SIAM Journal on Control and Optimization*, vol. 47, no. 5, pp. 2410–2439, 2008.

[45] K. Kandasamy, A. Krishnamurthy, J. Schneider, and B. Póczos, "Parallelised Bayesian optimisation via Thompson sampling," in *International Conference on Artificial Intelligence and Statistics*, pp. 133–142, PMLR, 2018.

[46] J. M. Hernández-Lobato, M. W. Hoffman, and Z. Ghahramani, "Predictive entropy search for efficient global optimization of black-box functions," *Advances in Neural Information Processing Systems*, vol. 27, 2014.

[47] K. Kandasamy, K. R. Vysyaraju, W. Neiswanger, B. Paria, C. R. Collins, J. Schneider, B. Poczos, and E. P. Xing, "Tuning hyperparameters without grad students: Scalable and robust Bayesian optimisation with dragonfly," *The Journal of Machine Learning Research*, vol. 21, no. 1, pp. 3098–3124, 2020.

[48] J. Chen, F. Luo, and Z. Wang, "Dynamic multi-objective ensemble of acquisition functions in batch Bayesian optimization," in *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pp. 479–482, 2022.

[49] S. R. Chowdhury and A. Gopalan, "On kernelized multi-armed bandits," in *International Conference on Machine Learning*, pp. 844–853, PMLR, 2017.

[50] K. J. Chan, J. A. Paulson, and A. Mesbah, "Safe explorative Bayesian optimization - towards personalized treatments in plasma medicine," in *Submitted to 2023 Proceedings of Conference on Decision and Control*, IEEE.

[51] S. Lucia, A. Tătulea-Codrean, C. Schoppmeyer, and S. Engell, "Rapid development of modular and sustainable nonlinear model predictive control solutions," *Control Engineering Practice*, vol. 60, pp. 51–62, 2017.

[52] M. Balandat, B. Karrer, D. Jiang, S. Daulton, B. Letham, A. G. Wilson, and E. Bakshy, "Botorch: A framework for efficient Monte-Carlo Bayesian optimization," *Advances in neural information processing systems*, vol. 33, pp. 21524–21538, 2020.

[53] W. Huyer and A. Neumaier, "SNOBFIT–stable noisy optimization by branch and fit," *ACM Transactions on Mathematical Software*, vol. 35, no. 2, pp. 1–25, 2008.

[54] S. T. Tokdar and R. E. Kass, "Importance sampling: A review," *Wiley Interdisciplinary Reviews: Computational Statistics*, vol. 2, no. 1, pp. 54–60, 2010.

[55] P. Goldberg, C. Williams, and C. Bishop, "Regression with input-dependent noise: A Gaussian process treatment," *Advances in Neural Information Processing Systems*, vol. 10, 1997.

[56] K. Kersting, C. Plagemann, P. Pfaff, and W. Burgard, "Most likely heteroscedastic Gaussian process regression," in *Proceedings International Conference on Machine Learning*, pp. 393–400, 2007.

[57] M. Lázaro-Gredilla and M. K. Titsias, "Variational heteroscedastic Gaussian process regression," in *Proceedings of the International Conference on Machine Learning*, pp. 841–848, 2011.

[58] K. Tuan Hoang, S. Boersma, A. Mesbah, and L. Imsland, "Heteroscedastic Bayesian optimisation for active power control of wind farms," *IFAC-PapersOnLine*, 2023 (accepted).

[59] J. Wu and P. Frazier, "The parallel knowledge gradient method for batch Bayesian optimization," *Advances in Neural Information Processing Systems*, vol. 29, 2016.

[60] S. Daulton, M. Balandat, and E. Bakshy, "Differentiable expected hypervolume improvement for parallel multi-objective Bayesian optimization," *Advances in Neural Information Processing Systems*, vol. 33, pp. 9851–9864, 2020.

[61] L. D. González and V. M. Zavala, "New paradigms for exploiting parallel experiments in Bayesian optimization," *Computers & Chemical Engineering*, vol. 170, p. 108110, 2023.

[62] T. S. Frisby, Z. Gong, and C. J. Langmead, "Asynchronous parallel bayesian optimization for ai-driven cloud laboratories," *Bioinformatics*, vol. 37, pp. i451–i459, 2021.

[63] M. Dorier, R. Egele, P. Balaprakash, J. Koo, S. Madireddy, S. Ramesh, A. D. Malony, and R. Ross, "HPC storage service autotuning using variational-autoencoder-guided asynchronous Bayesian optimization," in *International Conference on Cluster Computing*, pp. 381–393, IEEE, 2022.

[64] J. P. Folch, R. M. Lee, B. Shafei, D. Walz, C. Tsay, M. van der Wilk, and R. Misener, "Combining multi-fidelity modelling and asynchronous batch Bayesian optimization," *Computers & Chemical Engineering*, vol. 172, p. 108194, 2023.

[65] F. Sorourifar, N. Choksi, and J. A. Paulson, "Computationally efficient integrated design and predictive control of flexible energy systems

using multi-fidelity simulation-based Bayesian optimization," *Optimal Control Applications and Methods*, 2021.
[66] K. Kandasamy, G. Dasarathy, J. Schneider, and B. Póczos, "Multi-fidelity Bayesian optimisation with continuous approximations," in *International Conference on Machine Learning*, pp. 1799–1808, PMLR, 2017.
[67] M. Poloczek, J. Wang, and P. Frazier, "Multi-information source optimization," *Advances in Neural Information Processing Systems*, vol. 30, 2017.
[68] J. Song, Y. Chen, and Y. Yue, "A general framework for multi-fidelity Bayesian optimization with Gaussian processes," in *International Conference on Artificial Intelligence and Statistics*, pp. 3158–3167, PMLR, 2019.
[69] A. Kudva, F. Sorouifar, and J. A. Paulson, "Efficient robust global optimization for simulation-based problems using decomposed Gaussian processes: Application to MPC calibration," in *Proceedings of the American Control Conference*, pp. 2091–2097, IEEE, 2022.
[70] K. Wang, B. Wilder, S.-c. Suen, B. Dilkina, and M. Tambe, "Improving GP-UCB algorithm by harnessing decomposed feedback," in *Machine Learning and Knowledge Discovery in Databases*, pp. 555–569, Springer, 2020.
[71] J. A. Paulson and C. Lu, "Cobalt: Constrained bayesian optimization of computationally expensive grey-box models exploiting derivative information," *Computers & Chemical Engineering*, vol. 160, p. 107700, 2022.
[72] J. Wu, M. Poloczek, A. G. Wilson, and P. Frazier, "Bayesian optimization with gradients," *Advances in Neural Information Processing Systems*, vol. 30, 2017.
[73] S. Müller, A. von Rohr, and S. Trimpe, "Local policy search with Bayesian optimization," *Advances in Neural Information Processing Systems*, vol. 34, pp. 20708–20720, 2021.
[74] S. Penubothula, C. Kamanchi, and S. Bhatnagar, "Novel first order Bayesian optimization with an application to reinforcement learning," *Applied Intelligence*, vol. 51, pp. 1565–1579, 2021.
[75] R. S. Sutton, D. McAllester, S. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation," *Advances in Neural Information Processing Systems*, vol. 12, 1999.
[76] G. Makrygiorgos, J. A. Paulson, and A. Mesbah, "Gradient-enhanced Bayesian optimization via acquisition ensembles with application to reinforcement learning," *IFAC-PapersOnLine*, 2023 (accepted).
[77] N. Khan, D. E. Goldberg, and M. Pelikan, "Multi-objective Bayesian optimization algorithm," in *Proceedings of the 4th Annual Conference on Genetic and Evolutionary Computation*, pp. 684–684, 2002.
[78] D. Hernández-Lobato, J. Hernandez-Lobato, A. Shah, and R. Adams, "Predictive entropy search for multi-objective Bayesian optimization," in *International Conference on Machine Learning*, pp. 1492–1501, PMLR, 2016.
[79] S. Daulton, D. Eriksson, M. Balandat, and E. Bakshy, "Multi-objective Bayesian optimization over high-dimensional search spaces," in *Uncertainty in Artificial Intelligence*, pp. 507–517, PMLR, 2022.
[80] G. Makrygiorgos, A. D. Bonzanini, V. Miller, and A. Mesbah, "Performance-oriented model learning for control via multi-objective Bayesian optimization," *Computers & Chemical Engineering*, vol. 162, p. 107770, 2022.
[81] B. Eric, N. Freitas, and A. Ghosh, "Active preference learning with discrete choice data," *Advances in Neural Information Processing Systems*, vol. 20, 2007.
[82] J. González, Z. Dai, A. Damianou, and N. D. Lawrence, "Preferential Bayesian optimization," in *International Conference on Machine Learning*, pp. 1282–1291, PMLR, 2017.
[83] M. Abdolshah, A. Shilton, S. Rana, S. Gupta, and S. Venkatesh, "Multi-objective Bayesian optimisation with preferences over objectives," *Advances in Neural Information Processing Systems*, vol. 32, 2019.
[84] M. Zhu, D. Piga, and A. Bemporad, "C-GLISp: Preference-based global optimization under unknown constraints with applications to controller calibration," *IEEE Transactions on Control Systems Technology*, vol. 30, no. 5, pp. 2176–2187, 2021.