Available online at www.sciencedirect.com

ScienceDirect

IFAC PapersOnline
CONFERENCE PAPER ARCHIVE

IFAC PapersOnLine 50-1 (2017) 11800–11807

# Constrained Bayesian Optimization with Particle Swarms for Safe Adaptive Controller Tuning

Rikky R.P.R. Duivenvoorden [*] Felix Berkenkamp [**]
Nicolas Carion [**] Andreas Krause [**] Angela P. Schoellig [*]

[*] *University of Toronto Institute for Aerospace Studies (UTIAS), Canada. (e-mail: schoellig@utias.utoronto.ca)*
[**] *Department of Computer Science, ETH Zurich, Switzerland. (e-mail: {befelix,krausea}@ethz.ch)*

**Abstract:**
Tuning controller parameters is a recurring and time-consuming problem in control. This is especially true in the field of adaptive control, where good performance is typically only achieved after significant tuning. Recently, it has been shown that constrained Bayesian optimization is a promising approach to automate the tuning process without risking system failures during the optimization process. However, this approach is computationally too expensive for tuning more than a couple of parameters. In this paper, we provide a heuristic in order to efficiently perform constrained Bayesian optimization in high-dimensional parameter spaces by using an adaptive discretization based on particle swarms. We apply the method to the tuning problem of an $\mathcal{L}_1$ adaptive controller on a quadrotor vehicle and show that we can reliably and automatically tune parameters in experiments.

*Keywords:* Adaptive Control, Constrained Bayesian Optimization, Safety, Gaussian Process, Particle Swarm Optimization, Policy Search, Reinforcement learning

## 1. INTRODUCTION

Optimizing controller parameters is a difficult and recurring problem in practice. While methods have been proposed in the field of reinforcement learning, few consider the problem of providing safety guarantees during the optimization (e.g., satisfying state and input constraints). One promising method is constrained Bayesian optimization, which enables safe tuning of controller parameters (Berkenkamp et al., 2016). However, the method relies on a discretization of the parameter space, which limits it to optimizing only few parameters in practice.

In this paper, we consider the problem of scaling up constrained Bayesian optimization to larger parameter spaces. Specifically, we use an adaptive discretization based on particle swarms as a heuristic to speed up the process of determining the next parameters to evaluate. The resulting method retains the safety guarantees for evaluated parameters, but at a reduced computational cost. We show that good results can be a achieved for the example of tuning an $\mathcal{L}_1$ adaptive controller, a task that is known to be challenging in practice (Hovakimyan and Cao, 2010).

Optimizing or tuning controller parameters has been considered as part of the policy search problem in the reinforcement learning literature (Peters and Schaal, 2006). Recently, it has been shown that Bayesian optimization (Mockus, 2012) is an effective tool for this purpose. The method relies on Gaussian processes (GPs) (Rasmussen and Williams, 2006) in order to guide function evaluations to informative locations and provably converges to the global optimum (Srinivas et al., 2010). For example, Abdelrahman et al. (2016) use Bayesian optimization to optimize the power generated in photovoltaic power plants, Marco et al. (2016) tune the weight matrices of an LQR controller, and Calandra et al. (2014) optimize gaits for bipedal locomotion.

In the Bayesian optimization setting, safety has been considered in terms of safety constraints in the optimization problem that need to be satisfied for every parameter that is evaluated on the physical system. Suitable algorithms have been presented by Schreiter et al. (2015) and Sui et al. (2015), while Berkenkamp et al. (2016) applied the latter method to the parameter optimization of nonlinear control laws on quadrotor vehicles. However, these methods are relatively inefficient: the first one relies on the DIRECT algorithm (Jones et al., 1993) to determine parameters, which wastes significant time evaluating unsafe parameters, while the second one uses a fine discretization of the parameter space, which suffers from the curse of dimensionality.

An alternative optimization method is particle swarm optimization (Kiranyaz et al., 2014). The method uses particles that move through the parameter space, where the particle's dynamics depend both on the performance of other particles and its own objective function value.

[⋆] This research was supported in part by SNSF grant 200020_159557, NSERC grant RGPIN-2014-04634, and the Connaught New Researcher Award.

While no convergence guarantees exist, these methods have been shown to work well in practice (Engelbrecht, 2007). One appealing property of the method is that safety constraints can be considered by adapting the dynamics of the particles (Hu and Eberhart, 2002). this approach is not practical to apply on real systems due to the many simultaneous evaluations required for each particle.

In this paper, we use particle swarms to adaptively discretize the safe parameter space in constrained Bayesian optimization. We adapt the definitions in (Sui et al., 2015) and (Berkenkamp et al., 2016) to make them suitable for particle swarms, without losing safety guarantees, and show that the adaptive discretization drastically decreases the computation time required to determine promising controller parameters. We evaluate our approach on the challenging $\mathcal{L}_1$ adaptive controller tuning problem and demonstrate the effectiveness of our approach.

## 2. PROBLEM STATEMENT

We consider the controller optimization objective from (Berkenkamp et al., 2016). We assume that a control policy is available, which is parameterized by parameters $\boldsymbol{\theta}$ within a domain $\mathcal{D}$; in our case, this is an $\mathcal{L}_1$ adaptive controller. Given a certain task that we want the controlled system to perform, the goal is to determine the optimal controller parameters that maximize a performance measure $J(\boldsymbol{\theta})$ over the task, $\max_{\boldsymbol{\theta}\in\mathcal{D}} J(\boldsymbol{\theta})$. While the performance depends on the choice of controller parameters $\boldsymbol{\theta}$, it is typically calculated along a trajectory as a function of the tracking error and the control inputs. This corresponds to the standard reinforcement learning policy search setting, where we can query a parameter $\boldsymbol{\theta}_n$ at each iteration $n$ and observe a noisy measurement of the resulting performance $J(\boldsymbol{\theta}_n)$. Since these experiments cause wear in the robot and take time, the goal is to minimize the number of iterations before the optimal parameters are found.

In addition to the performance objective, we assume that the system is safety-critical; that is, for each parameter $\boldsymbol{\theta}_n$ that we evaluate on the system, a set of $q$ safety constraints must be satisfied, $g_i(\boldsymbol{\theta}_n) \geq 0$, $i \in \mathcal{I} = \{1,\dots,q\}$. For example, these constraints may correspond to state constraints that ensure operational safety. The resulting optimization problem is

$$\max_{\boldsymbol{\theta}\in\mathcal{D}} J(\boldsymbol{\theta}) \quad \text{such that:} \;\; g_i(\boldsymbol{\theta}) \geq 0 \;\forall i \in \mathcal{I}. \tag{1}$$

Solving (1) is difficult, since the functional dependence of $J(\boldsymbol{\theta})$ and $g_i(\boldsymbol{\theta})$ on the controller parameters in is unknown *a priori*. Existing algorithms that solve (1), e.g., SafeOpt in (Sui et al., 2015), start from an initial parameter set that is known to be safe *a priori* and use a GP model of the performance and constraint functions to generalize safety to untested parameters and safely explore the parameter space. These methods usually assume that computational resources are cheap and do not scale well with an increasing number of parameters in $\boldsymbol{\theta}_n$. In this paper, we focus on heuristics to make these methods applicable in practice.

For ease of exposition, we consider a single constraint $g(\boldsymbol{\theta}) = J(\boldsymbol{\theta}) \geq 0$ on the performance objective in the following. We provide information on how to extend the framework to multiple constraints wherever needed.

## 3. PRELIMINARIES

We start by introducing background information on GPs, Bayesian optimization, and particle swarm optimization.

### 3.1 Gaussian Processes (GPs)

While the performance $J(\boldsymbol{\theta})$ in (1) can easily be evaluated for a given parameter $\boldsymbol{\theta}$ in an experiment, the functional relationship between parameters and the cost is unknown *a priori*. We use GPs as a nonparametric model to approximate this unknown function, $J(\boldsymbol{\theta})\colon \mathcal{D} \mapsto \mathbb{R}$, from an input vector $\boldsymbol{\theta} \in \mathcal{D}$ to the function value $J(\boldsymbol{\theta})$. This is accomplished by assuming that function values $J(\boldsymbol{\theta})$, associated with different values of $\boldsymbol{\theta}$, are random variables and that any finite number of these random variables have a joint Gaussian distribution (Rasmussen and Williams, 2006).

For the nonparametric regression, we define a prior mean function $m(\boldsymbol{\theta})$, which encodes prior knowledge about the function $J(\cdot)$, and a covariance function $k(\boldsymbol{\theta}, \boldsymbol{\theta}')$, which defines the covariance of any two function values, $J(\boldsymbol{\theta})$ and $J(\boldsymbol{\theta}')$, $\boldsymbol{\theta}, \boldsymbol{\theta}' \in \mathcal{D}$, and is used to model uncertainty about the mean estimate. The latter is also known as the kernel. The choice of kernel is problem-dependent and encodes assumptions about smoothness and rate of change of the unknown function $J(\cdot)$.

The GP framework can be used to predict the function value $J(\boldsymbol{\theta}^*)$ at an arbitrary input $\boldsymbol{\theta}^* \in \mathcal{D}$, based on a set of $n$ past observations $\mathcal{D}_n = \{\boldsymbol{\theta}_i, \hat{J}(\boldsymbol{\theta}_i)\}_{i=1}^{n}$. We assume that observations are noisy measurements of the true function, $\hat{J}(\boldsymbol{\theta}) = J(\boldsymbol{\theta}) + \omega$ with Gaussian noise $\omega \sim \mathcal{N}(0, \sigma^2)$. Conditioned on the previous observations, the mean and variance of the posterior normal distribution are

$$\mu_n(\boldsymbol{\theta}^*) = m(\boldsymbol{\theta}^*) + \mathbf{k}_n(\boldsymbol{\theta}^*)\mathbf{K}_n^{-1}\hat{\mathbf{y}}_n, \tag{2}$$

$$\sigma_n^2(\boldsymbol{\theta}^*) = k(\boldsymbol{\theta}^*, \boldsymbol{\theta}^*) - \mathbf{k}_n(\boldsymbol{\theta}^*)\mathbf{K}_n^{-1}\mathbf{k}_n^{\mathrm{T}}(\boldsymbol{\theta}^*), \tag{3}$$

where $\hat{\mathbf{y}}_n = \left[\hat{J}(\boldsymbol{\theta}_1) - m(\boldsymbol{\theta}_1), \dots, \hat{J}(\boldsymbol{\theta}_n) - m(\boldsymbol{\theta}_n)\right]^{\mathrm{T}}$ is the vector of observed, noisy deviations from the mean, the vector $\mathbf{k}_n(\mathbf{a}^*) = [k(\boldsymbol{\theta}^*, \boldsymbol{\theta}_1), \dots, k(\boldsymbol{\theta}^*, \boldsymbol{\theta}_n)]$ contains the covariances between the new input $\boldsymbol{\theta}^*$ and the observed data points in $\mathcal{D}_n$, and the symmetric matrix $\mathbf{K}_n \in \mathbb{R}^{n\times n}$ has entries $[\mathbf{K}_n]_{(i,j)} = k(\boldsymbol{\theta}_i, \boldsymbol{\theta}_j) + \delta_{ij}\sigma^2$, $i, j \in \{1, \dots, n\}$.

### 3.2 Safe Bayesian Optimization

One class of methods that use GP models of the cost function to optimize parameters data-efficiently is Bayesian optimization (Mockus, 2012). These methods aim to determine the global optimum of cost functions that are expensive to evaluate. In our case, each evaluation of the cost represents an experiment on a robot with certain controller parameters, which causes wear in the system and may take a long time to perform. Bayesian optimization uses the mean, (2), and variance, (3), information provided by the GP to determine new parameters to evaluate that are promising candidates for the global optimum.

Safe Bayesian optimization (Sui et al., 2015; Schreiter et al., 2015) is an extension of this framework, which aims to solve the constrained optimization problem in (1). The method by Sui et al. (2015) uses a GP model of the safety constraints together with a discretization of the parameter space in order to determine a set of parameters $\mathcal{S}_n$

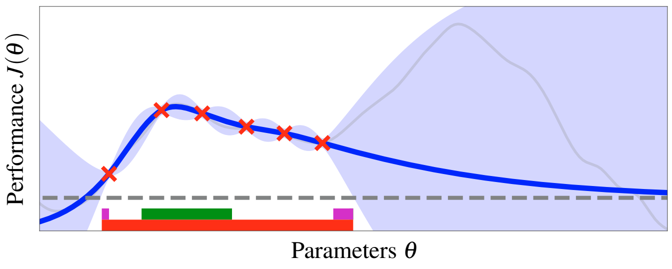


Fig. 1. Example of SafeOpt. The GP confidence intervals (blue shaded) are used to determine safe controller parameters (red set) that are above the safety threshold (gray dashed) with high probability. To optimize the function, the most uncertain point from the expanders (purple) or maximizers (green) is selected.

that satisfy the safety constraint with high probability. Within this set, the algorithm determines two sets: the set $\mathcal{M}_n \subseteq \mathcal{S}_n$ contains maximizers, parameters that could potentially obtain the maximum within the current safe set. The other set, $\mathcal{G}_n \subseteq \mathcal{S}_n$, uses a Lipschitz constant in order to determine safe parameters that could potentially enlargen the safe set given an optimistic measurement. These two sets are used to trade off exploration (expander set) and exploitation (maximizers) by choosing to evaluate the most uncertain parameter within both sets,

$$\boldsymbol{\theta}_n = \underset{\boldsymbol{\theta} \in \mathcal{M}_n \cup \mathcal{G}_n}{\operatorname{argmax}} \; \sigma_{n-1}(\boldsymbol{\theta}), \tag{4}$$

where $\sigma_{n-1}(\boldsymbol{\theta})$ is the standard deviation of the GP model (3). Repeatedly evaluating the objective at parameters given by (4) either expands the safe set or decreases uncertainty about the potential maximizers within the current safe set, so that the global optimum is found eventually. This is illustrated in Fig. 1. The main computational bottleneck of this method consists of maintaining and updating a fine discretization of the parameter set. Selecting a coarse discretization is not possible, since parameters need to be close to each other in order to use the GP model to generalize safety.

### 3.3 Particle Swarm Optimization (PSO)

Particle Swarm optimization (PSO) (Engelbrecht, 2007; Kiranyaz et al., 2014) is a heuristic method to optimize non-convex objective functions $f(\cdot)$. In this paper, we use a variant of this method to solve the optimization problem in (4). Formally, a swarm is a set of particles with positions $\mathbf{x}_i \in \mathcal{D}$ that represent individual parameters. Each particle has a velocity $\mathbf{v}_i$ and its position is governed by the standard equations of motion,

$$\mathbf{x}_i(t+1) = \mathbf{x}_i(t) + \mathbf{v}_i(t). \tag{5}$$

The velocity $\mathbf{v}_i$ is chosen so that the parameter space is explored effectively. At each iteration $t$ of the swarm, each particle $i$ evaluates the objective function at $\mathbf{x}_i(t)$ and keeps track of its best, safe objective value, $z_i(t) = \max_{t' \leq t} f(\mathbf{x}_i(t'))$. The velocity is then chosen by comparing the particle's best position, $\mathbf{z}_i$, and the global best position, $\bar{\mathbf{z}}(t) = \max_i \mathbf{z}_i(t)$, with the current position, $\mathbf{x}_i(t)$,

$$\mathbf{v}_i(t) = \alpha(t)\mathbf{v}_i(t-1) + r_1(\mathbf{z}_i(t) - \mathbf{x}_i(t)) + r_2(\bar{\mathbf{z}}(t) - \mathbf{x}_i(t)). \tag{6}$$

Here, $r_1 \in [0, c_1]$ and $r_2 \in [0, c_2]$ are samples from uniform distribution. It can be seen that particles are attracted both to their previous best position and the globally optimal solution. The inertia $\alpha(t)$ together with the with $c_1$ and $c_2$ are problem-specific tuning parameters that trade off exploration for local or global exploitation.

Constraints in the optimization can be incorporated by only updating the optimum estimates $\mathbf{z}_i$ when the constraint is fulfilled (Hu and Eberhart, 2002) or by adding a penalty for constraint violation; that is, changing the objective function to $f(\mathbf{x}) + p(\mathbf{x})$, where $p(\mathbf{x})$ is a smooth penalty for constraint violation (Parsopoulos et al., 2002; Parsopoulos and Vrahatis, 2005).

## 4. MAIN METHOD

In this section, we show how to scale safe Bayesian optimization to larger dimensions by using an adaptive discretization. We provide a Python implementation of our method at https://github.com/befelix/SafeOpt .

The main challenges to applying PSO to optimize the acquisition function (4) of safe Bayesian optimization are designing a sparse approximation for the safe set as initial points for the particles, defining suitable objective functions for the maxmimizers and expanders subject to the safety constraints, and selecting the tuning parameters of the PSO to be generally applicable.

### 4.1 Safety, Maximizers, and Expanders

As a first step, we need to determine whether each particle fulfills the safety constraints. To this end, we use high-probability confidence intervals of the GP model of (1),

$$l_n(\mathbf{x}) = \mu_n(\mathbf{x}) - \beta_n\sigma_n(\mathbf{x}), \tag{7}$$

$$u_n(\mathbf{x}) = \mu_n(\mathbf{x}) + \beta_n\sigma_n(\mathbf{x}), \tag{8}$$

where $l_n$ and $u_n$ are the lower and upper bound of the GP model after $n$ data points, with $\mu_n$ from (2) and $\sigma_n$ from (3). The true function value $J(\mathbf{x})$ lies within the confidence interval $[l_n(\mathbf{x}), u_n(\mathbf{x})]$ with high probability. The exact probability depends on the choice of $\beta_n$; typical choices are values of two or three. Based on these confidence intervals, we can determine parameters that are safe with high probability, $l_n(\mathbf{x}) \geq 0$; that is, parameters with worst-case function values above the safety threshold.

To use the particles for safe Bayesian optimization, we need to encode the objectives of expanding the safe set and maximizing the function within the safe set. At the same time, as in SafeOpt, we trade off exploration and exploitation by selecting the most uncertain parameter as in (4). Thus, the swarm should aim for high-variance parameters, while staying within the safe set.

Maximizers and expanders have fundamentally different objectives, which are difficult to unify into one cost function. Instead, we use different swarms with separate objectives that explore the parameter space simultaneously. We specify the objective functions of the particle swarms as $f(\mathbf{x}) = I(\mathbf{x})(\sigma_n(\mathbf{x}) + p(l_n(\mathbf{x}))$, where the GP's predictive standard deviation and a penalty for the violation of the safety constraint ($p(l_n(\mathbf{x})) \leq 0$ for violations) are multiplied by the interest function $I(\mathbf{x})$, which differs for both swarms. Intuitively, the objective function encodes that parameters with high variance that do not violate

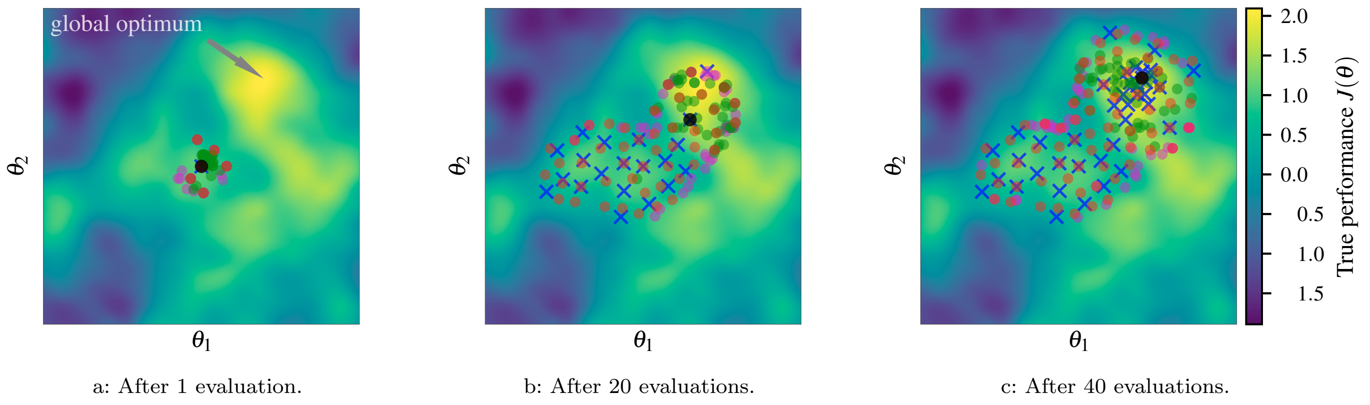


Fig. 2. Swarm optimization example. We maintain a coarse approximation of the safe set (red dots), at which the particle swarms are initialized. Starting from these, the expanders (magenta dots) and maximizers (green dots) search and eventually the objective function is evaluated at one data point (real experiment, blue cross). The estimate of the optimum (black dot) converges to the true, safely reachable optimum. Eventually only the optimum is evaluated.

the safety constraint significantly are desirable, while the interest function is used to select parameters that are expanders or maximizers.

For expanders, this interest function needs to be attractive to parameters that are close to the boundary of the safe set. As can be seen in Fig. 1, these boundary points have a lower bound that is close to the safety threshold. Thus, parameters with high uncertainty on the boundary of the safe set are promising candidates for expansion of the safe set. To encourage aiming close to the safety threshold, the interest function is defined as

$$I_{\exp}(\mathbf{x}) = \exp(-5 l_n^2(\mathbf{x})), \tag{9}$$

which is an exponential function centered on the safety threshold. The value five is chosen experimentally, see Sec. 4.3.

For the maximizers, relevant parameters are those that achieve an upper bound that is better than the best lower bound, which is the current estimate for the optimal parameters. Ideally, we would use a step function to cut off non-interesting states. However, in order to provide more information to the particles, we use the logistic function as a smooth approximation instead:

$$I_{\max}(\mathbf{x}) = \frac{1}{1 + \exp(l_{\max} - u_n(\mathbf{x}))}, \tag{10}$$

where $l_{\max} = \max_{\mathbf{x} \in \mathcal{D}} l(\mathbf{x})$ is the best lower bound. Note that this quantity itself is not easy to obtain. We use a third swarm with objective $f(\mathbf{x}) = l_n(\mathbf{x})$ in order to approximate it efficiently. An additional advantage of this scheme is that an estimate of the optimal parameters is computed as a by-product.

Based on these interest functions, the estimates for the best expander and maximizer can be found by running PSO based on the dynamics in (6) with the corresponding interest function. Based on this, the next parameters to evaluate on the real system are

$$\boldsymbol{\theta}_n = \underset{\boldsymbol{\theta} \in \{\boldsymbol{\theta}_{\exp}, \boldsymbol{\theta}_{\max}\}}{\operatorname{argmax}} \sigma_n(\boldsymbol{\theta}), \tag{11}$$

where $\boldsymbol{\theta}_{\exp}$ and $\boldsymbol{\theta}_{\max}$ are the optimal parameters with respect to $f(\mathbf{x})$ that are returned by the expander and maximizer swarms, respectively.

### *4.2 Approximation of Safe Set*

To make the algorithm efficient, particles must start in safe positions. We use a sparse set of previously safe parameters as initial positions. At the end of each run of the PSO, we test whether any safe particle positions are sufficiently 'far away' from positions in the sparse set of safe parameters. If such a 'new' candidate is found, it is added to the safe set. We discuss how to select an appropriate distance measure in the next section. The resulting safe set is illustrated by the red circles in Fig. 5. The initial positions of the particles are choosen uniformly at random from this sparse set of parameters at the beginning of each iteration $n$.

While the safe set grows over time, this growth is bounded. In our experiments, the computational cost of this sparse approximation was not significant relative to the cost of evaluating the GP posterior distribution.

### *4.3 Parameter Selection*

The performance of the algorithm depends on the choice of parameters for the swarm optimization. Specifically, the objective functions for the particle swarms in Sec. 4.1 are not invariant with respect to the magnitude of the function values of $J(\cdot)$ and $g(\cdot)$. Moreover, to approximate of the safe set in Sec. 4.2 and for the swarm dynamics (6) we have to compute distances. To ensure that particles explore within the safe region of parameters, these distances must depend on the rate of change of the constraints $g(\boldsymbol{\theta})$.

In order to make the algorithm applicable to a general class of problems, we have to make the swarm optimization invariant with respect to the properties of the functions that we optimize in (1). To this end, we use the assumptions about the objective and constraint functions that are encoded within the kernel.

In particular, let $\sigma_\eta$ be the prior standard deviation of the kernel of the GP model. In order to encode invariance towards function value magnitude, we scale $\sigma_n$, $l_n$, and $u_n$ in the swarm objective functions in Sec. 4.1 as well as the selection criterion in (4) by $\sigma_\eta$. For multiple constraints, this value varies for each GP model. Similarly, the prior covariance $k(\boldsymbol{\theta}, \boldsymbol{\theta})$, between two data points encodes a distance measure. We conduct a simple bisection line search to determine a distance scale: we selecte a distance scale $\mathbf{d}$ such that $k(\boldsymbol{\theta}, \boldsymbol{\theta} + \mathbf{d})/\sigma_\eta \simeq 0.95$. This distance measure is used to scale the distances in the computation of the swarm velocities, see (6). Moreover, we set the initial velocities of the particles equal to $\mathbf{d}$, in order to encourage a velocity that takes particles away from current positions, but without risking moving to far away from existing data points and leaving the safe set. In the case of multiple constraints, the smallest initial velocity is used.

Following (Homaifar et al., 1994), we use a multi-stage penalty function that is typical for swarm optimization,

$$p(l_n(\mathbf{x})) = \begin{cases} 0 & \text{if } l_n(\mathbf{x}) > 0, \\ 2l_n(\mathbf{x}) & \text{if } l_n(\mathbf{x}) \in [-0.001, 0], \\ 5l_n(\mathbf{x}) & \text{if } l_n(\mathbf{x}) \in [-0.1, -0.001], \\ 10l_n(\mathbf{x}) & \text{if } l_n(\mathbf{x}) \in [-1, -0.1], \\ -300l_n(\mathbf{x})^2 & \text{if } l_n(\mathbf{x}) < -1. \end{cases} \tag{12}$$

This function provides a strong incentive for particles to remain inside the safe set. For multiple constraints, the individual penalties are added up. In our experiments, the algorithm was robust towards the choice of penalty used.

An illustrative run of the algorithm on a two-dimensional optimization problem is shown in Fig. 5. The safe, global optimum is indicated close to the top-right corner. Starting from a single parameter evaluation that is known to be safe *a priori*, the particles start to explore the space in Fig. 2a. At the end of the particle swarm optimization, we select $\boldsymbol{\theta}_n$ according to (11) and evaluate its performance on the physical system. As more data becomes available, we update the GP model and the safe set increases. It can be seen in Fig. 2c that expander particles (magenta) stay close to the boundary of the safe set, while maximizers (green) aim for areas with high function values. The estimate of the best known parameters (black) eventually converges to the global optimum.

Overall, our method determines good parameters even for a small number of particles, since the particle swarms focus on exploring within the safe set. This provides the opportunity to actively trade off accuracy for computation time. In contrast, the non-adaptive optimization in (Berkenkamp et al., 2016) stalls on coarse discretizations. As a consequence swarm-based SafeOpt can be applied to higher-dimensional problems.

## 5. EXPERIMENTS

To demonstrate the effectiveness of the proposed optimization method, we use our approach to tune the parameters of an $\mathcal{L}_1$ adaptive output feedback controller. The controller is implemented on a Parrot AR.Drone 2.0 quadrotor, and optimized for maximum tracking performance of a circular trajectory with a diameter of 3 m. First, two parameters are optimized followed by optimization over four parameters to show the computational benefits.

### *5.1 Overview of the Extended $\mathcal{L}_1$ Adaptive Controller*

The $\mathcal{L}_1$ adaptive controller is based on the model reference adaptive control (MRAC) architecture with the addition of a low-pass filter that decouples robustness from adaptation (Hovakimyan and Cao, 2010). This allows arbitrarily high gains to be chosen for fast adaptation. In the experiments, the typical $\mathcal{L}_1$ adaptive output feedback controller for single-input single-output (SISO) systems in (Hovakimyan and Cao, 2010) is nested within a proportional controller, as shown in Fig. 3. The SISO architecture is implemented separately in each direction and combined to control the multi-input multi-output (MIMO) quadrotor system. This architecture is identical to the one by Michini and How (2009) and Pereida et al. (2017). The $\mathcal{L}_1$ adaptive output feedback controls the quadrotor's translational velocity, while the outer-loop proportional position controller ensures that the system remains within the physical limitations of the flight area.

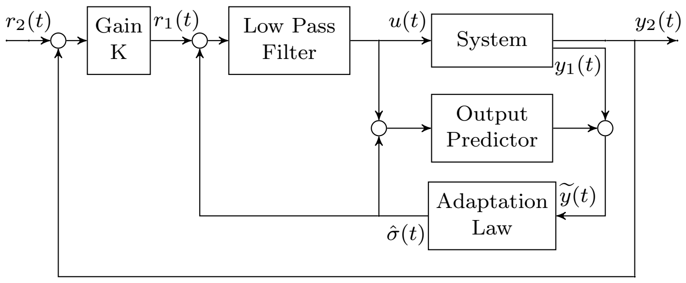


Fig. 3. The extended $\mathcal{L}_1$ adaptive controller that is optimized using the swarm-based SafeOpt algorithm.

The objective of the extended $\mathcal{L}_1$ adaptive output feedback controller is to determine a control input $u(t)$ such that the quadrotor system position $y_2(t)$ tracks a bounded, piecewise continuous reference position trajectory $r_2(t)$. The $\mathcal{L}_1$ adaptive controller uses the adaptive estimate $\hat{\sigma}$ in the control law to compensate for any unknown disturbances and unmodeled dynamics of the system. The theoretical details of this architecture are provided in (Pereida et al., 2017), while $\mathcal{L}_1$ adaptive control theory is discussed in (Hovakimyan and Cao, 2010).

The individual components of the SISO $\mathcal{L}_1$ architecture are introduced next. With the exception of the proportional negative feedback loop, this architecture from $r_1$ to $y_1$ is identical to (Hovakimyan and Cao, 2010). The equations describing the components of the extended $\mathcal{L}_1$ output feedback architecture are (13), (14), (15), and (17) below.

*Output Predictor* We use a first-order, continuous-time output predictor in the frequency domain within the $\mathcal{L}_1$ adaptive output feedback architecture,

$$\hat{y}_1(s) = \frac{m}{m+s}(u(s) + \hat{\sigma}(s)), \tag{13}$$

where $\hat{\sigma}(t)$ is the adaptive estimate, and $m$ is the eigenvalue or pole location of the output predictor. Larger magnitudes of $m$ demand a faster response from the system and contributes to the high frequency signal in the adaptive estimate that enables fast adaptation.

*Adaptation Law* We update the adaptive estimate $\hat{\sigma}(t)$ according to the update law

$$\dot{\hat{\sigma}}(t) = \Gamma \text{Proj}(\hat{\sigma}(t), -mP\tilde{y}(t)), \qquad \hat{\sigma}(0) = 0, \tag{14}$$

where $\tilde{y}(t) \triangleq \hat{y}_1(t) - y_1(t)$ and $P > 0$ solves the algebraic Lyapunov equation $mP + Pm = 2mP = -Z$ for $Z > 0$.

The variable $\Gamma \in \mathbb{R} > 0$ is the adaptation rate. It has a lower bound as discussed in (Hovakimyan and Cao, 2010) to guarantee stability. Typically in $\mathcal{L}_1$ adaptive control, $\Gamma$ is set very large. Our experiments were carried out with an adaptation rate of $\Gamma = 5000$. The projection operator defined in (Hovakimyan and Cao, 2010) guarantees that the estimation of $\sigma$ remains within a specified convex set.

*Control Law* The control input signal is the difference between the $\mathcal{L}_1$ desired trajectory signal $r_1$ and the adaptive estimate $\hat{\sigma}$ after being filtered by the low-pass filter $C(s)$:

$$u(s) = C(s)\big(r_1(s) - \hat{\sigma}(s)\big). \tag{15}$$

The low-pass filter used in our experiments is the following third-order filter:

$$C(s) = \frac{3\omega_c^2 s + \omega_c^3}{(s + \omega_c)^3}, \tag{16}$$

where $\omega_c$ is the cutoff frequency of the filter. The filter ensures that only the low frequencies of the error signal that the system actuators are capable of counteracting are passed through. The high-frequency portion is attenuated by the low-pass filter.

*Closed-Loop Feedback* The following equation describes the closed-loop feedback acting on the input to the $\mathcal{L}_1$ adaptive output feedback controller $r_1$ based on the position of the system $y_1$:

$$r_1(s) = K(r_2(s) - y_2(s)), \tag{17}$$

where the objective is for $y_2$ to track $r_2$. The proportional gain is $K$.

The SISO $\mathcal{L}_1$ adaptive control architecture discussed above is extended to the MIMO quadrotor system by implementing $(3 \times 3)$ diagonal transfer function matrices for the low-pass filter and first-order output predictor. The signals $r_1(t)$, $r_2(t)$, $y_1(t)$, and $y_2(t)$ are the desired translational velocity, desired position, quadrotor translational velocity and quadrotor position, respectively. Each element of the three-dimensional signals and each diagonal element of the transfer function matrices correspond to the $x$, $y$ and $z$ inertial directions, respectively. Since the transfer function matrices are diagonal, any coupling between the $x$ and $y$ dynamics are compensated for by the controller.

### 5.2 Optimization Parameters

In the following, we present the results of two experiments. The first optimization problem considers the two main $\mathcal{L}_1$ adaptive control tuning parameters of the horizontal velocity controller, namely the first-order output predictor pole location and the low-pass filter cutoff frequency, such that $\boldsymbol{\theta} = \{\omega_{xy}, m_{xy}\}$. This two-dimensional problem shows the ability of the algorithm to safely optimize the nonlinear underlying cost function of the $\mathcal{L}_1$ adaptive controller, and allows the exploration strategy of the algorithm to be visualized. Since the quadrotor is assumed to behave identically in roll and pitch, the optimization is conducted in the horizontal plane over the low-pass filter cutoff frequency $\omega_{xy}$, and the first-order output predictor $m_{xy}$ in both the $x$ and $y$ directions simultaneously. For this reason, a circular trajectory was chosen in the horizontal plane with a diameter of 3 m. The performance metric used for this optimization penalizes the radial position error according to the cost function

$$J(\boldsymbol{\theta}) = \sum_{i=1}^{N} \frac{1}{N}\big(-500\tilde{r_i}^2\big), \tag{18}$$

where $i$ denotes the timestep within a particular experiment, $N$ is the total number of timesteps within an iteration of the optimization, and $\tilde{r}_i \triangleq r_{\text{des}} - \sqrt{{x_i}^2 + {y_i}^2}$ is the radial position error at the $i$th timestep. The safety constraint is defined to be $g(\boldsymbol{\theta}) = J(\boldsymbol{\theta}) \geq -25$ determined experimentally to produce safe results with this cost function.

The main computational advantage of the proposed method is demonstrated through the second optimization problem over four parameters of the $\mathcal{L}_1$ adaptive controller. These are the cutoff frequency and output predictor eigenvalue of the $z$ controller as well as the horizontal controller as in the first optimization, such that $\boldsymbol{\theta} = \{\omega_{xy}, m_{xy}, \omega_z, m_z\}$. The trajectory is changed to include a total altitude change of 0.5 m over the same circular trajectory. To capture the altitude tracking performance in the cost function, an additional term is added for the vertical position error:

$$J(\boldsymbol{\theta}) = \sum_{i=1}^{N} \frac{1}{N}\big(-500\tilde{r_i}^2 - 1000\tilde{z_i}^2\big), \tag{19}$$

where $\tilde{z}_i \triangleq z_{\text{des},i} - z_i$ is the altitude error at the $i$th timestep at which the radial error is also computed. The safety constraint is adjusted to $g(\boldsymbol{\theta}) = J(\boldsymbol{\theta}) \geq -30$, which reflects the additional altitude error cost term.

### 5.3 Gaussian Process (GP) Kernel

To use Bayesian optimization, we need to specify a GP prior over the objective functions (18) and (19). The kernel used in the optimization is the Matèrn kernel with parameter $\nu = 3/2$ (Rasmussen and Williams, 2006):

$$k(\boldsymbol{\theta}, \boldsymbol{\theta}') = \sigma_\eta^2\Big(1 + \sqrt{3}\ r(\boldsymbol{\theta}, \boldsymbol{\theta}')\Big)\exp\Big(-\sqrt{3}\ r(\boldsymbol{\theta}, \boldsymbol{\theta}')\Big), \tag{20}$$

$$r(\boldsymbol{\theta}, \boldsymbol{\theta}') = \sqrt{(\boldsymbol{\theta} - \boldsymbol{\theta}')^{\mathrm{T}}\mathbf{M}^{-2}(\boldsymbol{\theta} - \boldsymbol{\theta}')}, \tag{21}$$

where $\sigma_\eta^2$ is the prior variance and $\mathbf{M}$ is a diagonal matrix containing the positive length-scales $\mathbf{l} \in \mathbb{R}^{|\mathcal{D}|} > 0$, such that $\mathbf{M} = \text{diag}(\mathbf{l})$. This means there are the following three hyperparameters, the process noise $\sigma^2$ defined in Sec. 3.1, the prior variance $\sigma_\eta^2$, and the positive length-scales $\mathbf{l}$. For the experiments conducted for this paper, the process noise is set to 6% of the initial controller performance $J(\boldsymbol{\theta}_0)$ and the prior variance is set to 50% of the initial controller performance. In the first optimization problem, the GP length-scales were chosen to be 0.60 for both the cutoff frequency $\omega_{xy}$ and the output predictor eigenvalue $m_{xy}$ such that $\mathbf{l} = [0.6\ 0.6]$. Larger length-scales assume that the underlying cost function does not vary as much with respect to the parameters. This speeds up the optimization process since fewer data points are required to capture the behavior of the cost function.

In the second optimization problem, the GP kernel models the parameters of the $z$ controller to be additive, which makes the assumption that the controller parameters in

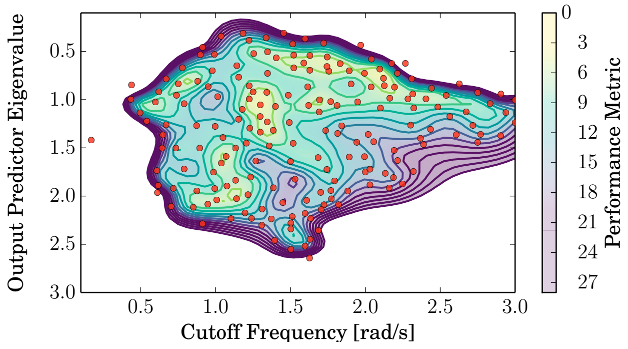


Fig. 4. Contour plot of the optimization of the low-pass filter and output predictor eigenvalue for the horizontal controller. Individual evaluations are indicated by the red dots. The contour shows the GP mean estimate of the cost function.

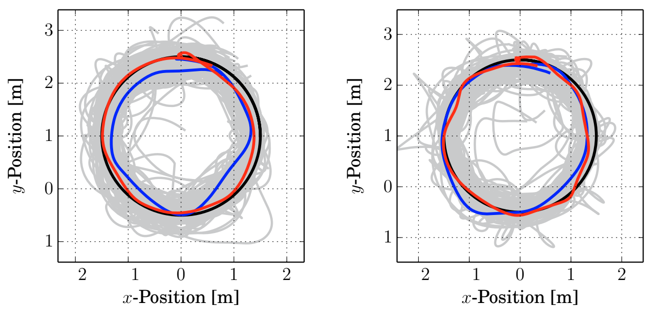
the $z$ direction independently influence the cost from the parameters of the $x$ and $y$ directions. Modeling the $z$ direction as an additive error allows for more data-efficient optimization, as this encodes additional problem structure. The length-scales were chosen to be $\mathbf{l} = [0.6\ 0.6\ 0.7\ 0.7]$ to take the two additional parameters into account and expedite the exploration of the four-dimensional parameter space.

### 5.4 Optimization Results

The contour plot of the first optimization problem of the low-pass filter and output predictor is shown in Fig. 4. The optimization begins with a parameter combination of $\omega_{xy} = 2.0\,\mathrm{rad/s}$ and $m_{xy} = -1.75$ with an initial performance of $-16.97$. Convergence was reached after 187 iterations, after the standard deviation within the safe region decreased below a threshold of 0.7. The optimal parameters were found to be $\omega_{xy} = 1.34\,\mathrm{rad/s}$ and $m_{xy} = -0.92$ with a performance of $-2.80$. This is an 83.5% reduction in the cost. The contour plot in Fig. 4 shows the nonlinear and non-convex cost function that confirms the challenge of manually finding the optimal parameters of the $\mathcal{L}_1$ adaptive controller. In favor of exploring the space faster, the length-scales were chosen to be relatively large. This occasionally resulted in high penalties that violated the safety constraint indicated by the parameter evaluations lying outside of the contours. This illustrates the trade-off between data-efficiency and safety.

The trajectories of the two-dimensional optimization are shown in Fig. 5a. The 83.5% improvement in the performance metric is clearly visible when comparing the optimized (red) and initial (blue) radial errors. Although the intermediate iterations have worse performance, the evaluations were still safe and resulted in no crashes, as the gray trajectories show. The time required to calculate the set of parameters for each iteration was between 0.22 and 0.53 seconds.

The second optimization problem was started with the controller parameters set to $\omega_{xy} = 1.34\,\mathrm{rad/s}$, $m_{xy} = -0.92$, $\omega_z = 0.45\,\mathrm{rad/s}$ and $m_z = -3.5$, which resulted in an initial performance of $-14.60$. The initial $x$,$y$ parameters were chosen to be the optimal parameters of the two-dimensional optimization. This allows us to examine



a: Two parameter optimization. b: Four parameter optimization.

Fig. 5. Tracking performance comparison for the two and four parameter optimization. The initial controller trajectory is shown in blue, the optimized controller in red, and the desired trajectory in black. The other iterations of the optimization are shown in gray.

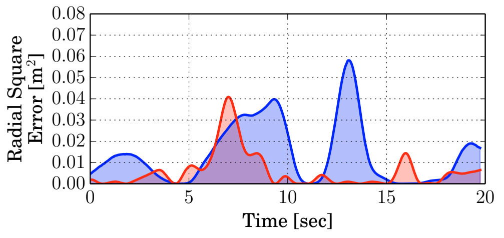


Fig. 6. Radial square error showing the initial and optimized controller in blue and red, respectively.

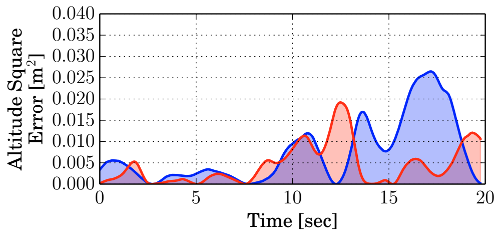


Fig. 7. Altitude square error showing the initial and optimized controller in blue and red, respectively.

the $z$ direction independence discussed above. This occurs when the optimal $x$,$y$ parameters of the four-dimensional optimization problem correspond to the same $x$ and $y$ controller parameters from the two-dimensional optimization.

The high dimensionality of the problem results in an exponential increase in the number of iterations required for convergence, because the kernel in (20) couples all the parameters. For this reason, the $z$ parameters are modelled to be additive to reduce the number of iterations. The two parameter optimization above found the optimal parameters at the 43rd iteration, so in practice, reasonable performance can be obtained in relatively few experiments and convergence is not required. For this reason, the four parameter optimization was stopped after 100 iterations. The optimized parameters after 100 iterations are $\omega_{xy} = 1.73\,\mathrm{rad/s}$, $m_{xy} = -0.68$, $\omega_z = 1,01\,\mathrm{rad/s}$ and $m_z = -2.14$ with a performance of $-7.34$. Compared to the initial controller performance of $-14.60$, the optimized trajectory represents a 49.7% improvement. The trajectories of the four-dimensional optimization are shown in Fig. 5b and the cost function error metrics composed of the radial and altitude square errors are shown in Fig. 6 and Fig. 7, respectively. The areas under the curves relate to the square error over the entire trajectory used in the cost function. The reduction in area under the optimized curves relative to the initial curves illustrates the performance improvement. In terms of computation, the time that was required to calculate the set of parameters for each iteration was between 0.32 and 0.46 seconds. This is comparable to the iterations of the two-dimensional optimization above, demonstrating the computational advantage of our approach. In contrast, without the adaptive discretization as in (Berkenkamp et al., 2016), computation time with the GP model can take several minutes for four parameters.

To analyze the independence of the $z$ controller parameters, Fig. 6 shows that the initial radial square error is lower compared to the optimal parameters. Although this suggests that the $z$ controller parameters are not independent, it can also be due to the optimization that was not completed to convergence and noise that is present in the results. Furthermore, the optimal $x$ and $y$ controller parameters $\omega_{xy} = 1.73\,\text{rad/s}$ and $m_{xy} = -0.68$ correspond to the third local maximum with magnitude $-3.30$ in the cost function shown in Fig. 4. This still represents an 80.6% improvement, and likely confirms that the $z$ controller parameters influence the cost independently.

The current optimization approach solves the computational burden when the number of optimization parameters is increased. However, the statistical problem that stationary kernels lead to an exponentional increase in the number of iterations required until convergence remains. In practice, this can be alleviated by encoding structure (e.g, additivity in the cost function), increasing the lengthscales, or relaxing the convergence condition.

## 6. CONCLUSION

We presented an extension of SafeOpt, a safe Bayesian optimization algorithm. We used an adaptive discretization based on particle swarms in order to decrease the computational complexity for high-dimensional problems. The resulting algorithm was applied to tuning an $\mathcal{L}_1$ adaptive controller on a quadrotor, which confirmed that this approach is flexible, practical, and can safely optimize controller parameters with low computational cost.

## REFERENCES

Abdelrahman, H., Berkenkamp, F., and Krause, A. (2016). Bayesian optimization for maximum power point tracking in photovoltaic power plants. In *Proc. of the European Control Conference (ECC)*, 2078–2083.

Berkenkamp, F., Krause, A., and Schoellig, A.P. (2016). Bayesian optimization with safety constraints: safe and automatic parameter tuning in robotics. Technical report. ArXiv:1602.04450 [cs.RO].

Calandra, R., Gopalan, N., Seyfarth, A., Peters, J., and Deisenroth, M.P. (2014). Bayesian gait optimization for bipedal locomotion. In *Learning and Intelligent Optimization*, 274–290. Springer.

Engelbrecht, A.P. (2007). *Computational intelligence: an introduction.* John Wiley & Sons, 2nd edition.

Homaifar, A., Qi, C.X., and Lai, S.H. (1994). Constrained optimization via genetic algorithms. *Simulation*, 62(4), 242–253.

Hovakimyan, N. and Cao, C. (2010). *$\mathcal{L}_1$ adaptive control theory: guaranteed robustness with fast adaptation.* Society for Industrial and Applied Mathematics.

Hu, X. and Eberhart, R. (2002). Solving constrained nonlinear optimization problems with particle swarm optimization. In *Proc. of the World Multiconference on Systemics, Cybernetics and Informatics*, volume 5, 203–206.

Jones, D.R., Perttunen, C.D., and Stuckman, B.E. (1993). Lipschitzian optimization without the Lipschitz constant. *Journal of Optimization Theory and Applications*, 79(1), 157–181.

Kiranyaz, S., Ince, T., and Gabbouj, M. (2014). *Multidimensional particle swarm optimization for machine learning and pattern recognition.* Springer.

Marco, A., Hennig, P., Bohg, J., Schaal, S., and Trimpe, S. (2016). Automatic LQR tuning based on Gaussian process global optimization. In *Proc. of the IEEE International Conference on Robotics and Automation (ICRA)*, 270–277.

Michini, B. and How, J. (2009). $\mathcal{L}_1$ adaptive control for indoor autonomous vehicles: design process and flight testing. In *Proc. of the AIAA Guidance, Navigation, and Control Conference*, 5754–5768.

Mockus, J. (2012). *Bayesian approach to global optimization: theory and applications.* Springer Science & Business Media.

Parsopoulos, K.E. and Vrahatis, M.N. (2005). Unified particle swarm optimization for solving constrained engineering optimization problems. In *Proc. of the International Conference on Natural Computation*, 582–591. Springer.

Parsopoulos, K.E., Vrahatis, M.N., and others (2002). Particle swarm optimization method for constrained optimization problems. *Intelligent Technologies–Theory and Application: New Trends in Intelligent Technologies*, 76(1), 214–220.

Pereida, K., Duivenvoorden, R.R.P.R., and Schoellig, A.P. (2017). High-precision trajectory tracking in changing environments through $\mathcal{L}_1$ adaptive feedback and iterative learning. In *Proc. of the IEEE International Conference on Robotics and Automation (ICRA)*. Accepted.

Peters, J. and Schaal, S. (2006). Policy gradient methods for robotics. In *Proc. of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2219–2225.

Rasmussen, C.E. and Williams, C.K. (2006). *Gaussian processes for machine learning.* MIT Press, Cambridge MA.

Schreiter, J., Nguyen-Tuong, D., Eberts, M., Bischoff, B., Markert, H., and Toussaint, M. (2015). Safe exploration for active learning with Gaussian processes. In *Machine Learning and Knowledge Discovery in Databases*, 133–149. Springer International Publishing.

Srinivas, N., Krause, A., Kakade, S.M., and Seeger, M. (2010). Gaussian process optimization in the bandit setting: no regret and experimental design. In *Proc. of the International Conference on Machine Learning (ICML)*, 1015–1022.

Sui, Y., Gotovos, A., Burdick, J.W., and Krause, A. (2015). Safe exploration for optimization with Gaussian processes. In *Proc. of the International Conference on Machine Learning (ICML)*, 997–1005.