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 can typically only be achieved after significant tuning effort. 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 safe 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 tune parameters.

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

1. INTRODUCTION

Optimizing controller parameters is a difficult and recurring problem in controller design. While many methods have been proposed in the field of reinforcement learning, few consider the practical 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 optimizations over only a few parameters in practice.

In this paper, we consider the problem of scaling up safe 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. The resulting method retains the safety guarantees of safe Bayesian optimization, at a reduced computational cost. We show that good results can be achieved at the example of tuning an $\mathcal{L}_1$ adaptive controller, which 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) tuned the weight matrices of an LQR controller, and Calandra et al. (2014) optimizes gaits for bipedal locomotion.

In the Bayesian optimization setting, safety has been considered in terms of constraints on the optimization problem that need to be satisfied for every parameter that is evaluated on the physical system. This problem has been considered 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 (Jones et al., 1993) algorithm to determine parameters, which can spend a lot of time evaluating unsafe parameters, while the second one uses a discretization of the parameter space that suffers from the curse of dimensionality.

We use ideas from particle swarm optimization (Kiranyaz et al., 2014) to overcome this problem and speed up safe Bayesian optimization. The method considers particles moving through parameter space, where the particle's dynamics depend both on the performance of other particles

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

Preprint submitted to 20th IFAC World Congress. Received November 11, 2016.

and its own function value. While it is difficult to provide guarantees, 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).

In this paper, we use particle swarms as a heuristic to adaptively discretize the safe parameter space for constrained Bayesian optimization. We adapt the definitions from (Sui et al., 2015) to be suitable for particle swarms, without losing safety guarantees, and show that the adaptive discretization drastically decreases the amount of computation time required to determine promising controller parameters. We evaluate our approach on a challenging $\mathcal{L}_1$ adaptive controller tuning problem, which demonstrates 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 some 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 determined.

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, $g_i(\theta_n) \geq 0$, $i \in \mathcal{I} = \{1, \dots, q\}$. For example, these constraints may correspond to state constraints that ensure operational safety. Importantly, the safety constraint must be satisfied for any controller parameters that are evaluated. The resulting optimization problem is

$$\max_{\boldsymbol{\theta}\in\mathcal{D}} J(\boldsymbol{\theta}), \tag{1}$$

$$\text{s.t: } g_i(\theta_n) \geq 0 \,\forall i \in \mathcal{I} \,\forall n > 0. \tag{2}$$

Solving (2) is difficult, since the functional dependence on the controller parameters is unknown *a priori*. Existing algorithms that solve (2), 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 cheap computational resources, which makes them increasingly difficult to apply when more parameters are involved. In this paper, we focus on heuristics to make these methods applicable in practice. For ease of exposition, we consider a single constraint function $g(\theta) = J(\theta)$ 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 (2) 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. The goal is to approximate the nonlinear map $J(\boldsymbol{\theta})\colon \mathcal{D} \mapsto \mathbb{R}$ 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; that is, $\hat{J}(\boldsymbol{\theta}) = J(\boldsymbol{\theta}) + \omega$ 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

$$\mu_n(\boldsymbol{\theta}^*) = m(\boldsymbol{\theta}^*) + \mathbf{k}_n(\boldsymbol{\theta}^*)\mathbf{K}_n^{-1}\hat{\mathbf{y}}_n, \tag{3}$$

$$\sigma_n^2(\boldsymbol{\theta}^*) = k(\boldsymbol{\theta}^*, \boldsymbol{\theta}^*) - \mathbf{k}_n(\boldsymbol{\theta}^*)\mathbf{K}_n^{-1}\mathbf{k}_n^{\mathrm{T}}(\boldsymbol{\theta}^*), \tag{4}$$

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 with certain controller parameters on the robot cause wear in the system and may take a long time to perform. Bayesian optimization uses the mean, (3), and variance, (4), information provided by the GP to determine

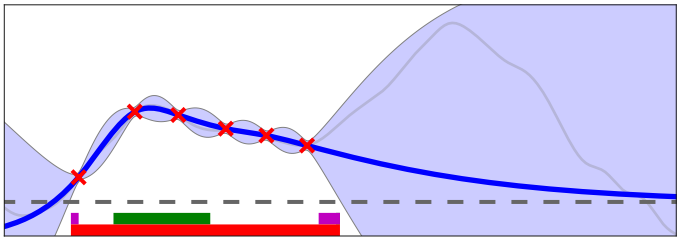


Fig. 1. Example of SafeOpt. The confidence intervals of the GP model (blue shaded) are used to determine controller safe parameters (red set) that are above the safety threshold (gray dashed) with high probability. To optimize the function, the most uncertain point from the expanders (purple) or maximizers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

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 (2). 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 $\mathcal{S}_n$ 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,

$$\boldsymbol{\theta}_n = \underset{\boldsymbol{\theta} \in \mathcal{M}_n \cup \mathcal{G}_n}{\operatorname{argmax}} \ \sigma_{n-1}(\boldsymbol{\theta}), \tag{5}$$

where $\sigma_{n-1}(\boldsymbol{\theta})$ is the standard deviation of the GP model, (4). Repeatedly evaluating the objective at parameters given by (5) 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 the discretization of the parameter set.

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 $f(\cdot)$. It draws inspiration from bird flocks, which can exhibit surprisingly complex behaviours as a whole, even though each individual bird has a limited computational power.

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 goverend by the standard equations of motion,

$$\mathbf{x}_i(t+1) = \mathbf{x}_i(t) + \mathbf{v}_i(t). \tag{6}$$

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 it's 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, $\mathbf{x}_i(t)$,

$$\mathbf{v}_i(t) = \alpha(t)\mathbf{v}_i(t-1) + r_1(\mathbf{x}_i(t) - \mathbf{z}_i(t)) + r_2(\mathbf{x}_i(t) - \bar{\mathbf{z}}(t)). \tag{7}$$

Here, $\mathbf{r}_1$ and $\mathbf{r}_2$ are samples from Gaussian random variables. 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 standard deviations of $\mathbf{r}_1$ and $\mathbf{r}_2$ are problem-specific tuning parameters that trade off exploraion for local or global exploitation.

Constraints in the optimization can be incorporated by only updating the optimum estimates $\mathbf{z}_i$ when the constrained 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 computational bottleneck of Bayesian optimization methods is the optimization of the acquisition function in (5). In the case of (Berkenkamp et al., 2016), this is done via an inefficient discretization, while the method in (Schreiter et al., 2015) relied on the DIRECT algorithm (Jones et al., 1993), which spends a lot of time evaluation the GP model outside the safe set and can have troubles with the many local optima that are typically present in the acquisition function (5). Here, we use ideas from PSO instead. We exploit the fact that typical safe regions in Bayesian optimzation are connected and use the swarm to restrict the parameter search to the feasible region. The resulting algorithm is efficient and scales to higher-dimensional problems.

The main challenges to applying this method 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.

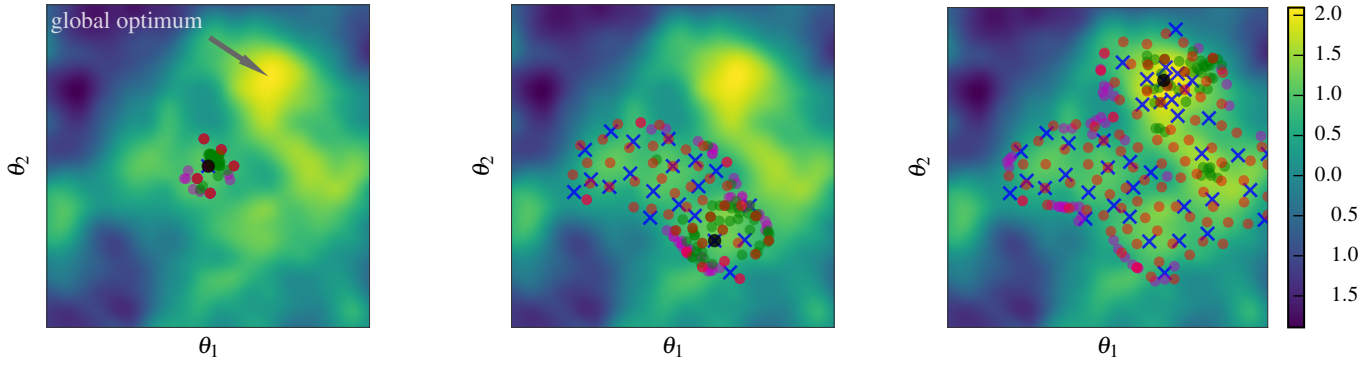
4.1 Safety, Maximizers, and Expanders

As a first step, we need determine for each particle whether it fulfills the safety constraints with high probability. To this end, we use high-probability confidence intervals of the GP model of (2),

$$l_n(\mathbf{x}) = \mu_n(\mathbf{x}) - \beta_n \sigma_n(\mathbf{x}), \tag{8}$$

$$u_n(\mathbf{x}) = \mu_n(\mathbf{x}) + \beta_n \sigma_n(\mathbf{x}), \tag{9}$$

where l_n and u_n are the lower and upper bound of the GP model after n data points, respectively, with μ_n from (3) and σ_n from (4). 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;



a: After 1 evaluation. b: After 20 evaluations. c: After 40 evaluations.

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 (blue cross). The estimate of the optimum (black dot) converges to the true, safely reachable optimum. Eventually only the optimum is evaluated.

typical choices for beta are values of two or three. Based on these confidence intervals, we can determine parameters that are safe with high probability as $l_n \geq 0$; that is, parameters whos worst-case function values lie above the safety threshold.

For now, let us assume that the initial positions of the particles satisfy this safety constraint. 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 eventually trade off exploration for exploitation by selecting the most uncertain parameter as in (5). Thus, the swarm should additionally 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. Here, separate objectives for different swarms, which 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 $p(l_n(\mathbf{x})) \leq 0$ for the violation of the safety constraint are multiplied by the interest function $I(\mathbf{x})$ that differs for both swarms. Intuitively, the objective function encodes looking for high-variance parameters that do not violate the safety constraint significantly, while the interest function further limits the choice to parameters that are expanders are 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

$$I_{\text{expander}}(\mathbf{x}) = \exp(-l_n(\mathbf{x})^2/0.2), \tag{10}$$

which is an exponential function centered on the safety threshold.

For the maximizers, relevant parameters are those that achieve an upper bound that is better than the best lower bound, because the best lower bound 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 smoothed out step unfction instead:

$$I_{\text{maximizers}}(\mathbf{x}) = \frac{1}{1 + \exp(l_{\max} - u_n(\mathbf{x}))}, \tag{11}$$

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, so that we use a third swarm with objective $f(\mathbf{x}) = l(\mathbf{x})$ 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 (7) with the corresponding interest function. Based on this, the next parameters to evaluate on the real system are computed according to

$$\boldsymbol{\theta_n} = \operatorname*{argmax}_{\boldsymbol{\theta} \in \{\boldsymbol{\theta}_{\text{exp}}, \boldsymbol{\theta}_{\max}\}} \sigma_n(\boldsymbol{\theta}), \tag{12}$$

where $\boldsymbol{\theta}_{\text{exp}}$ and $\boldsymbol{\theta}_{\max}$ are the best parameters of the expander and maximizer swarms, respectively.

4.2 Approximation of Safe Set

So far, we have assumed that the initial positions of the particles are safe. In practice, while we assume that an initial set of safe parameters is known, the true safe set is rarely convex, which makes it difficult to approximate. We use a sparse set of previously safe parameters as initial positions for future parameters. At the end of each run of the PSO, we test whether any of the best, safe particle positions are sufficiently far away from any other particles in the safe set. 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. 2. 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 this method implies that the safe set grows over time, this growth is clearly bounded by the maximum size

of the safe set. In our experiments, the computational cost of this sparse approximation was not significant relative to the cost of the GP evaluations.

4.3 Parameter Selection

The performance of the algorithm depends on the choice of parameters for the swarm optimization and the cost/distance measures in (7), (10), (11), and Sec. 4.2. There are two main challenges, the meaning of distance varies significantly with respect to the Lipschitz constants of the functions that we consider, while the objective functions specified in Sec. 4.1 depend on the magnitude of the function values.

In order to tune these parameters only once for a general class of problems, these parameter need to depend on properties of the functions that we optimize in (2). We exploit the fact that the kernel encodes a distance measure as well as a prior over the magnitude of function values.

In particular, let σ_p be the prior standard deviation of the kernel of the GP model. In order to encode invariances towards function value magnitude, we scale σ_n, l_n, and u_n in the swarm objective functions in Sec. 4.2 as well as the selection criterion in (5) by σ_p. 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; in our case, we selected the distance scale $\mathbf{d}$ such that $k(\boldsymbol{\theta}, \boldsymbol{\theta} + \mathbf{d})/\sigma_\mathrm{p} \simeq 0.95$. This distance measure is used to scale the distances in the computation of the swarm velocities, (7). 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 point and leaving the safe set. In the case of multiple constraints, the smallest initial velocity is used to ensure safety.

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

$$p(l_n(\mathbf{x})) = \begin{cases} 0 & \text{if } l_n(\mathbf{x}) > 0, \\ 2l_n(\mathbf{x}) & \text{if } l_n(\mathbf{x}) \in [-0.001, 0], \\ 5l_n(\mathbf{x}) & \text{if } l_n(\mathbf{x}) \in [-0.1, -0.001], \\ 10l_n(\mathbf{x}) & \text{if } l_n(\mathbf{x}) \in [-1, -0.1], \\ -300l_n(\mathbf{x})^2 & \text{if } l_n(\mathbf{x}) < -1. \end{cases} \tag{13}$$

While this function is discontinuous, it provides strong incentives to the 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 simple two-dimensional optimization problem can be seen in Fig. 2. 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 prior*, the particles start to explore the space in Fig 2a. Eventually the most uncertain parameter from the two best candidates of the expander and maximizer swarms is evaluated. As the safe set increases, expander particles (magenta) stay close to the boundary of the safe set, while maximizers (green) aim for areas with high function values, as can be seen in Fig. 2c. The estimate of the best known parameters (black) eventually converges to the global optimum. Overall, the computation time is significantly lower than running vanilla SafeOpt from (Berkenkamp et al., 2016). Moreover, computation time scales with the number of particles, rather than the number of dimensions in the parameters space. This allows one to actively trade off accuracy for computation time.

5. EXPERIMENTS

To demonstrate the feasibility of the proposed optimization method, the parameters of the $\mathcal{L}_1$ adaptive output feedback controller are tuned using this approach. The controller is implemented on a Parrot AR.Drone 2.0 quadrotor, and is tuned to optimize the tracking performance of a circular trajectory with a diameter of 3 m in 15 s.

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 adaptation gains to be chosen for fast adaptation. In the experiments, the typical $\mathcal{L}_1$ adaptive output feedback controller for single-input single-output systems from (Hovakimyan and Cao, 2010) is nested within a proportional controller, as shown in Fig. 3. This extended architecture is identical to (Michini and How, 2009) and (Pereida et al., 2016). The $\mathcal{L}_1$ adaptive output feedback controls the quadrotor translational velocity and the outer-loop proportional position controller ensures that the system remains within the physical limitations of the flight area.

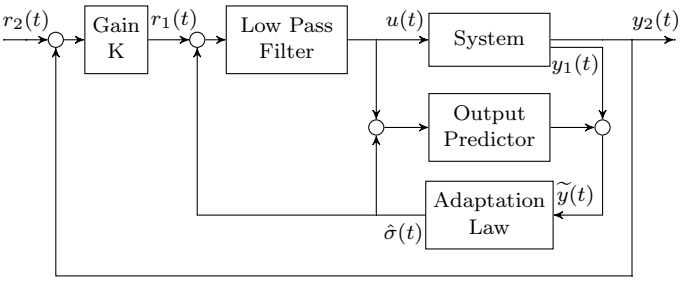


Fig. 3. The extended $\mathcal{L}_1$ adaptive controller that is optimized using the

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 $r_2(t)$. Any uncertainties in the form of unknown disturbances and unmodelled dynamics of the system are compensated for by the $\mathcal{L}_1$ adaptive controller by using the adaptive estimate $\hat{\sigma}$ in the control law. The theoretical details of this architecture are provided in (Pereida et al., 2016), while further details of $\mathcal{L}_1$ adaptive control theory are discussed in (Hovakimyan and Cao, 2010).

The individual components of the $\mathcal{L}_1$ 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 (15), (16), (17), and (19).

Output Predictor The following first-order output predictor is used within the $\mathcal{L}_1$ adaptive output feedback architecture:

$$\dot{\hat{y}}_1(t) = -m\hat{y}_1(t) + m(u(t) + \hat{\sigma}(t)) \ , \qquad \hat{y}_1(0) = 0 \ , \quad (14)$$

where $\hat{\sigma}(t)$ is the adaptive estimate. In the Laplace domain, this is equivalent to:

$$\hat{y}_1(s) = \frac{m}{m+s}(u(s) + \hat{\sigma}(s)) \ . \quad (15)$$

The eigenvalue, or pole location of the output predictor, m is one of the tuning parameters being optimized.

Adaptation Law The adaptive estimate, $\hat{\sigma}(t)$, is updated according to the following update law:

$$\dot{\hat{\sigma}}(t) = \Gamma \text{Proj}(\hat{\sigma}(t), -mP\tilde{y}(t)) \ , \qquad \hat{\sigma}(0) = 0 \ , \quad (16)$$

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 subject to the lower bound as specified in (Hovakimyan and Cao, 2010). Typically in $\mathcal{L}_1$ adaptive control, Γ is set very large. Experiments with this controller were carried out with an adaptation rate of $\Gamma = 5000$.

The projection operator defined in (Hovakimyan and Cao, 2010) ensures that the estimation of σ is guaranteed to remain 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 $C(s)$:

$$u(s) = C(s)(r_1(s) - \hat{\sigma}(s)) \ . \quad (17)$$

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

$$C(s) = \frac{3\omega_c^2 s + \omega_c^3}{(s+\omega_c)^3} \ , \quad (18)$$

where ω_c is the cut-off frequency of the filter, tuned by the optimization. The filter ensures that only the low frequencies, which the system is capable of counteracting, are compensated for. 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:

$$r_1(s) = K(r_2(s) - y_2(s)) \ , \quad (19)$$

where the objective is for y_2 to track r_2. The proporional gain, K is the final tuning parameter being optimized in the three dimensional optimization.

The single-input single-output $\mathcal{L}_1$ adaptive control architecture discussed above is extended to the multi-input multi-output quadrotor system by implementing 3×3 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 GP Kernel

The kernel used in the optimization is the Matèrn kernel with parameter $\nu = 3/2$ (Rasmussen and Williams, 2006):

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

$$r(\boldsymbol{\theta}_i, \boldsymbol{\theta}_j) = \sqrt{(\boldsymbol{\theta}_i - \boldsymbol{\theta}_j)^{\mathrm{T}} \mathbf{M}^{-2} (\boldsymbol{\theta}_i - \boldsymbol{\theta}_j)} \ , \quad (21)$$

where σ_η^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 σ^2 defined in Section 3.1, the prior variance σ_η^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. The length-scales were dependent on the specific optimization and are provided below.

5.3 Performance Function

The performance metric used for optimization penalizes the horizontal position error and the desired roll and pitch control inputs according to the following cost function:

$$J(\boldsymbol{\theta}) = J(\tilde{\mathbf{x}}, \mathbf{u}) = \sum_{i=1}^{N} \frac{1}{N}\big(-1.5\tilde{\mathbf{x}}_i{}^{\mathrm{T}}\mathbf{I}\tilde{\mathbf{x}}_i - 0.15\mathbf{u}_i{}^{\mathrm{T}}\mathbf{I}\mathbf{u}_i\big), \quad (22)$$

where i denotes the timestep within a particular experiment, N is the total number of timesteps within an iteration of the optimization, $\tilde{\mathbf{x}}_i \triangleq [x_{\text{des},i} - x_i \ \ y_{\text{des},i} - y_i]^{\mathrm{T}}$ is the horizontal position error at the i^{th} timestep, and $\mathbf{u}_i \triangleq [\phi_{\text{des},i} \ \theta_{\text{des},i}]^{\mathrm{T}}$ are the desired roll and pitch angles and the i^{th} timestep, respectively. The roll and pitch angles are calculated by an underlying attitude controller, which has been tuned beforehand.

5.4 Optimization Results

First, the tuning problem is constrained to 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. This two dimensional problem shows the ability of the algorithnm 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 ω_{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. The contour plot of the optimization of the low-pass filter and output

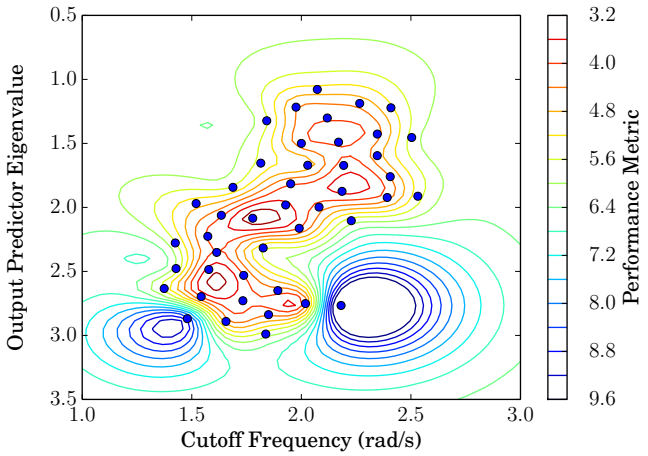


Fig. 4. Contour plot of the optimization of the low-pass filter and output predictor eigenvalue. Individual evaluations are indicated by the blue dots, and contours show the GP mean estimate of the underlying cost function.

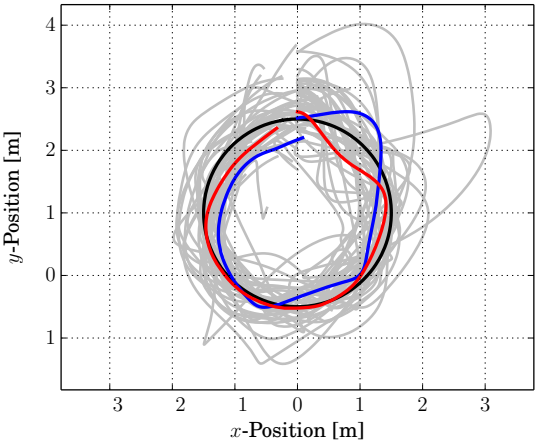


Fig. 5. Tracking performance comparison for the optimization of ω_{xy} and m_{xy}. The initial controller trajectory is shown in blue, the optimized controller in red, and the desired circular trajectory in black. The other iterations of the optimization process are shown in gray.

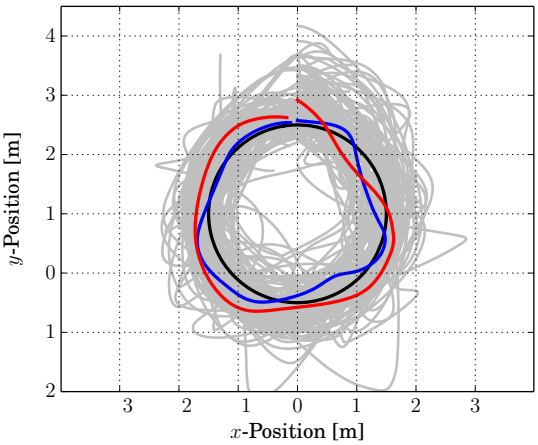


Fig. 6. Tracking performance comparison for the optimization of ω_{xy}, m_{xy}, and K_{xy}. The initial controller trajectory is shown in blue, the optimized controller in red, and the desired circular trajectory in black. The other iterations of the optimization process are shown in gray.

predictor is shown in Fig. 4, while keeping the proportional gain $K_{xy} = 0.4$ constant. The GP length-scales were chosen to be 0.3 for both the the cut-off frequency ω_{xy} and the output predictor eigenvalue m_{xy} such that $\mathbf{l} = [0.3\ 0.3]$.

The optimization started at a parameter combination of $(\omega_{xy} = 2.0, m_{xy} = -1.5)$ with an initial performance of -4.112. Convergence was reached after 43 iterations, after the standard deviation within the safe region decreased below a threshold of 0.7. The optimal parameters were found to be $(\omega_{xy} = 1.778, m_{xy} = -2.085)$ with a performance of -2.084. This is a 49.3% reduction in the cost. There were two iterations that resulted in a performance of -11.24 and -8.89 surrounded by the blue contours in Fig. 4. These were encountered because the length-scales were chosen to be relatively large in favour of exploring the space faster. The poor performance of -11.24 is primarily attributable to the large control inputs that originate from the higher frequency signals that are not within the bandwidth of the quadrotor actuators. The cost function in the parameter space is also highly nonlinear, which confirms the challenge of manually finding the optimal parameters of the $\mathcal{L}_1$ adaptive controller.

The trajectories of the two dimensional optimization are shown in Fig. 5. The 49.3% improvement in the performance metric is clearly reflected in the comparison between the optimized (red) and initial (blue) controllers.

The main computational advantage of the proposed method is demonstrated by optimizing in higher dimensions. This is conducted by optimizing over three parameters of the extended $\mathcal{L}_1$ adaptive output feedback controller. Compared to the two dimensional optimization before, the three dimensional optimization also considers the outer loop proportional gain, K_{xy}. Since the optimization still considers the x and y directions only, the performance function in (22) and the desired trajectory remain the same. The GP kernel also remains unchanged with the exception of the length-scales. These were selected to be $\mathbf{l} = [0.45\ 0.45\ 0.15]$ to take into account the third parameter and to speed up the exploration of the three dimensional parameter space. The optimization was started with the controller parameters set to $(\omega_{xy} = 2.0, m_{xy} = -1.5, K_{xy} = 0.4)$, which resulted in an initial performance of -4.630. After 103 iterations, convergence was reached because the uncertainty within the safe region reduced below the standard deviation threshold of 0.7. The trajectories of the three dimensional optimization are shown in Fig. 6.

The optimized parameters are $(\omega_{xy} = 1.628, m_{xy} = -1.303, K_{xy} = 0.709)$ with a performance of -2.454. Compared to the initial controller performance of -4.630, this represents a 47.0% improvement. Although the im-

provement in tracking performance is not immediately obvious, the cost function is also based on the control inputs which penalizes behaviour such as rapid acceleration or sharp turns. This is more evident in the initial controller shown in blue, compared to the smoother trajectory of the optimized controller in red. Better tracking performance is expected from the optimization if the cost function is changed accordingly by increasing the position error penalty.

It is also interesting to note that the optimized performance is slightly worse compared to the two dimensional optimization under the same performance function. This can be the combined result of the increased legnth-scales and standard deviation convergence threshold that could prevent further reduction of the uncertainty in areas close to local and global optima. However, reducing both the length-scales and convergence threshold will considerably increase the number of iterations required for convergence, and this will quickly grow if more optimization parameters are added. Although the current optimization approach solves the computational burden if the number of parameters to be optimized is increased, the exponentional increase in iterations required to produce the data is still applicable, and can be alleviated in practice by increasing the length-scales or relaxing the condition required for convergence,

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 and can safely optimize controller parameters with a 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 Angela P. Schoellig (2016). Bayesian optimization with safety constraints: safe and automatic parameter tuning in robotics. arXiv.

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. Citeseer.

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)*.

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. (2016). High-precision trajectory tracking in changing environments through $\mathcal{L}_1$ adaptive feedback and iterative learning. arXiv.

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*, 9286, 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.