Bayesian Optimization with Adaptive Kernels for Robot Control

Ruben Martinez-Cantin

***Abstract*— Active policy search combines the trial-and-error methodology from policy search with Bayesian optimization to actively find the optimal policy. First, policy search is a type of reinforcement learning which has become very popular for robot control, for its ability to deal with complex continuous state and action spaces. Second, Bayesian optimization is a sample efficient global optimization method that uses a surrogate model, like a Gaussian process, and optimal decision making to carefully select each sample during the optimization process. Sample efficiency is of paramount importance when each trial involves the real robot, expensive Monte Carlo runs, or a complex simulator. Black-box Bayesian optimization generally assumes a cost function from a stationary process, because nonstationary modeling is usually based on prior knowledge. However, many control problems are inherently nonstationary due to their failure conditions, terminal states and other abrupt effects. In this paper, we present a kernel function specially designed for Bayesian optimization, that allows nonstationary modeling without prior knowledge, using an adaptive local region. The new kernel results in an improved local search (exploitation), without penalizing the global search (exploration), as shown experimentally in well-known optimization benchmarks and robot control scenarios. We finally show its potential for the design of the wing shape of a UAV.**

I. INTRODUCTION

When a baby is learning a new behavior, like walking or grasping, she performs it several times, trying to improve the outcome of the behavior in each trial or exploring new strategies. This trial and error methodology share many points with the model-free direct policy search methodology that has been quite successful in robotics [1]. These algorithms are designed to maximize or minimize a respective reward or cost function as a function of a parametrization of the policy or behavior that the agent is following. Traditionally, the most popular methods were variations of policy gradient methods [2]. Recently, there has been an increasing number of methods that have been applying Bayesian optimization in the context of policy search for robotics. The advantages of Bayesian optimization for policy search are: a) ability to find the global optimum, b) after testing few policies, c) without gradient information and d) without model or instant reward. Recent studies have found connections between Bayesian optimization and the way biological systems adapt and search in nature [3].

Bayesian optimization is a classic global optimization method [4] which has become quite popular recently for being sample efficient [5] and applied with great success for machine learning applications [6], etc. In the context of robotics it has been applied for robot planning [7], [8], control [9], [10], task optimization [11], grasping [12], model-free reinforcement learning [13], [14], model-based reinforcement learning [15], sensor networks [16], etc.

Bayesian optimization is the combination of two main components: a surrogate model which captures all prior and observed information and a decision process which performs the optimal action, i.e.: where to sample next, based on the previous model. Thus, the quality of the surrogate model is of paramount importance as it also affects the optimality of the decision process. Earliest versions of Bayesian optimization used Wiener processes [4] as surrogate models. It was the seminal paper of Jones et al. [5] that introduced the use of Gaussian processes (GP). The GP model is the most popular due to its accuracy, robustness and flexibility, because Bayesian optimization is mainly used in black or grey-box scenarios. The range of applicability of a GP is defined by its kernel function, which sets the family of functions that is able to represent through the reproducing kernel Hilbert space (RKHS) [17]. From a practical point of view, the standard procedure is to select a generic kernel function, such as the Gaussian (square exponential) or Matérn kernels, and estimate the kernel hyperparameters from data. One property of these kernels is that they are stationary. Although it might be a reasonable assumption in a black box setup, we show in Section II-B that this reduces the efficiency of Bayesian optimization in most situations. It also limits the potential range of applications. Moreover, nonstationay methods usually require extra knowledge of the function (e.g.: the global trend or the space partition). Being global properties, gathering this knowledge from data requires global sampling, which is contrary to the Bayesian optimization methodology.

The main contribution of the paper is a new set of adaptive kernels for Gaussian processes that are specifically designed to model functions from nonstationary processes but focused on the local region near the optimum. Thus, the new model maintains the philosophy of global exploration/local exploitation. This idea results in an improved sample efficiency of any Bayesian optimization based on Gaussian processes. We call this new method *Spartan Bayesian Optimization* (SBO). The algorithm has been extensively evaluated in many scenarios and applications. Besides some standard optimization benchmarks, we show the applicability to optimal policy learning in reinforcement learning scenarios. Furthermore, to highlight the general purpose of the method, we also show an application of autonomous wing design of a UAV.

II. BAYESIAN OPTIMIZATION WITH GAUSSIAN PROCESSES

Consider the problem of finding the minimum of an unknown real valued function $f : \mathbb{X} \to \mathbb{R}$, where $\mathbb{X}$

*This work was supported in part by projects DPI2015-65962-R (MINECO/FEDER, UE), CUD2013-05 and CUD2016-17.

R. Martinez-Cantin is at Centro Universitario de la Defensa, Zaragoza, Spain. and SigOpt, Inc. rmcantin@unizar.es

is a compact space, $\mathbb{X} \subset \mathbb{R}^d, d \geq 1$. In order to find the minimum, the algorithm has a maximum budget of N evaluations of the target function f. The purpose of the algorithm is to select the best query points at each iteration such that the *optimization gap* or *regret* is minimum for the available budget.

The surrogate model is a Gaussian process $\mathcal{GP}(\mathbf{x}|\mu, \sigma^2, \boldsymbol{\theta})$ with inputs $\mathbf{x} \in \mathbb{X}$, scalar outputs $y \in \mathbb{R}$ and an associate kernel or covariance function $k(\cdot,\cdot)$ with hyperparameters $\boldsymbol{\theta}$. The hyperparameters are estimated from data using Markov Chain Monte Carlo (MCMC) resulting in m samples $\boldsymbol{\Theta} = \{\boldsymbol{\theta}_i\}_{i=1}^m$. Given a dataset at step n of query points $\mathbf{X} = \{\mathbf{x}_{1:n}\}$ and its respective outcomes $\mathbf{y} = \{y_{1:n}\}$, then the prediction of the Gaussian process at a new query point $\mathbf{x}_q$, with kernel k_i conditioned on the i-th hyperparameter sample $k_i = k(\cdot,\cdot|\boldsymbol{\theta}_i)$ is a normal distribution such as $y_q \sim 1/m \sum_{i=1}^m \mathcal{N}(\mu_i, \sigma_i^2|\mathbf{x}_q)$ where:

$$\begin{aligned} \mu_i(\mathbf{x}_q) &= \mathbf{k}_i(\mathbf{x}_q, \mathbf{X})\mathbf{K}_i(\mathbf{X}, \mathbf{X})^{-1}\mathbf{y} \\ \sigma_i^2(\mathbf{x}_q) &= k_i(\mathbf{x}_q, \mathbf{x}_q) \\ &\quad - \mathbf{k}_i(\mathbf{x}_q, \mathbf{X})\mathbf{K}_i(\mathbf{X}, \mathbf{X})^{-1}\mathbf{k}_i(\mathbf{X}, \mathbf{x}_q) \end{aligned} \tag{1}$$

being $\mathbf{k}_i(\mathbf{x}_q, \mathbf{X})$ the corresponding cross-correlation vector of the query point $\mathbf{x}_q$ with respect to the dataset $\mathbf{X}$

$$\mathbf{k}_i(\mathbf{x}_q, \mathbf{X}) = [k_i(\mathbf{x}_q, \mathbf{x}_1), \ldots, k_i(\mathbf{x}_q, \mathbf{x}_n)]^T$$

and $\mathbf{K}_i(\mathbf{X}, \mathbf{X}) = \mathbf{K}_G + \sigma_n^2\mathbf{I}$ is the Gram matrix $\mathbf{K}_G$ corresponding to kernel k_i for the dataset $\mathbf{X}$, and σ_n^2 is a noise term to represent stochastic functions or surrogate mismodeling. The prediction at any point $\mathbf{x}$ is a mixture of Gaussians because we use a sampling distribution of $\boldsymbol{\theta}$.

For the decision process, first we rely on an initial design of p points based on *Latin Hypercube Sampling* (LHS) [5], to avoid initialization bias. Subsequent points are selected using the *expected improvement* criterion [4] which is defined as the expectation of the improvement function $I(\mathbf{x}) = \max(0, \rho - f(\mathbf{x}))$. This improvement is defined over a incumbent target ρ, which in many applications is considered to be the *best outcome* until that iteration $\rho = y_{best}$. Taking the expectation over the mixture of Gaussians of the predictive distribution, we can compute the expected improvement as:

$$EI(\mathbf{x}) = \sum_{i=1}^{m} \left[(\rho - \mu_i(\mathbf{x}))\,\Phi(z_i) + \sigma_i(\mathbf{x})\phi(z_i)\right] \tag{2}$$

where ϕ and Φ are the corresponding Gaussian probability density function (PDF) and cumulative density function (CDF), being $z_i = (\rho - \mu_i(\mathbf{x}))/\sigma_i(\mathbf{x})$. At iteration n, we select the next query at the point that maximizes the expected improvement $\mathbf{x}_n = \arg\max_{\mathbf{x}} EI(\mathbf{x})$

A. Kernels for Bayesian optimization

Many applications of Gaussian process regression, including Bayesian optimization, are based on the assumption that the process is stationary. This is a reasonable assumption for black-box optimization as it does not assume any extra information on the evolution of the function in the space. For example, the use of the squared exponential (SE) kernel in GPs is quite frequent: $k_{SE}(\mathbf{x}, \mathbf{x}') = \exp\left(-\frac{1}{2}r^2\right)$ where r is the weighted L^2 norm between $\mathbf{x}$ and $\mathbf{x}'$. The hyperparameters of the kernel are the components of the weighting matrix of the norm Λ. In many applications, the matrix is a simple constant $\Lambda = \theta^{-1}\mathbf{I}$ (isotropic process). If we use a diagonal matrix $\Lambda = diag(\theta_1^{-1}, \ldots, \theta_d^{-1})$ (anisotropic process), it is called *automatic relevance determination* (ARD).

Because the SE kernel is infinitely differentiable, it tends to over-smooth functions. For Bayesian optimization, a more suitable kernel is the Matérn kernel family, specifically the the Matérn kernel with $\nu = 5/2$, as it provides a good trade off of differentiability/smoothness:

$$k_{M5}(\mathbf{x}, \mathbf{x}') = \exp\left(-\sqrt{5}r\right)\left(1 + \sqrt{5}r + \frac{5}{3}r^2\right) \tag{3}$$

For these kernels, the hyperparameters $\boldsymbol{\theta}_l$ represent the length-scales that captures the smoothness or variability of the function in the corresponding dimension [17]. Small values of $\boldsymbol{\theta}_l$ will be more suitable to capture signals with high frequency components; while large values of $\boldsymbol{\theta}_l$ result in a model for low frequency signals or flat functions. This effect is very important in Bayesian optimization. For the same distance between points, a kernel with smaller length-scale will result in higher predictive variance, therefore the exploration will be more aggressive. This idea was previously explored in Wang et al. [18] by forcing smaller scale parameters to improve the exploration. More formally:

Proposition 1: [18] Given two kernels k_l and k_s with large and small length scale hyperparameters respectively, any function f in the RKHS characterized by a kernel k_l is also an element of the RKHS characterized by k_s.

Thus, using k_s instead of k_l is safer in terms of guaranteeing convergence. However, if the small kernel is used everywhere, it might result in unnecessary sampling of smooth areas.

B. Nonstationary Gaussian processes

Consider the problems of Section V-B.1 where a biped robot (agent) is trying to learn the walking pattern (policy) that maximizes the walking speed (reward). In this setup, there are some policies that reach undesirable states or result in a failure condition, like the robot falling or losing the upright posture. Then, the system returns a null reward or arbitrary penalty. In cases where finding a stable policy is difficult, the reward function may end up being almost flat, except for a small region of successful policies where the reward is actually informative in order to maximize the speed.

Modeling these kind of functions with Gaussian processes require kernels with different length scales for the flat/non-flat regions or specially designed kernels to capture that behavior. Furthermore, Bayesian optimization is inherently a local stationary process depending on the acquisition function. It has a dual behavior of global exploration and local exploitation. Ideally, both samples and uncertainty estimation end up being distributed unevenly, with many samples and small uncertainty near the local optima and sparse samples and large uncertainty everywhere else.

Definition 1: Let $f : \mathbb{R}^d \to \mathbb{R}$ be a function and $\mathcal{H}_k$ be the reproducing kernel Hilbert space generated by kernel $k(\cdot, \cdot)$.

- We say that a function $f(\mathbf{x})$ is *stationary* if $\exists\ k(\mathbf{x}, \mathbf{x}') = k(\tau)$ where $\tau = \mathbf{x} - \mathbf{x}'$ and $f \in \mathcal{H}_k$.
- In contrast, we say that a function $f(\mathbf{x})$ is *nonstationary* if $\nexists\ k(\mathbf{x}, \mathbf{x}') = k(\tau)$ where $\tau = \mathbf{x} - \mathbf{x}'$ and $f \in \mathcal{H}_k$.
- Finally, we say that a function $f(\mathbf{x})$ is *local stationary* if there is a subset $\mathcal{X} \subset \mathbb{R}^d$ so that the function is stationary $\forall \mathbf{x} \in \mathcal{X}$ and nonstationary $\forall \mathbf{x} \in \mathbb{R}^d \setminus \mathcal{X}$.

According to the previous definition, most applications of Bayesian optimization are nonstationary or local stationary. Also, even for stationary problems, we might want different levels of exploration in different regions, which might require using two or more length-scales as seeing in Section II-A.

There have been several attempts to model nonstationarity in Bayesian optimization. Bayesian treed GPs were used in Bayesian optimization combined with an auxiliary local optimizer [19]. An alternative is to project the input space through a warping function to a stationary latent space [20]. Later, Assael et al. [21] built treed GPs where the warping model was used in the leaves. These methods were direct application of regression methods, that is, they model the nonstationary property in a global way. However, as pointed out before, sampling in Bayesian optimization is uneven, thus the global model might end up being inaccurate.

III. Spartan Bayesian Optimization

Our approach to nonstationarity is based on the model presented in Krause & Guestrin [22] where the input space is partitioned in different regions such that the resulting GP is the linear combination of local GPs: $\xi(\mathbf{x}) = \sum_j \lambda_j(\mathbf{x})\xi_j(\mathbf{x})$. Each local GP has its own specific hyperparameters, making the final GP nonstationary even when the local GPs are stationary. In order to achieve smooth interpolation between regions, the authors suggest the use of a weighting function $\omega_j(\mathbf{x})$ for each region, having the maximum in region j and decreasing its value with distance to region j [22]. Then, we can set $\lambda_j(\mathbf{x}) = \sqrt{\omega_j(\mathbf{x}) / \sum_p \omega_p(\mathbf{x})}$. In practice, the mixed GP can be obtained by a combined kernel function of the form: $k(\mathbf{x}, \mathbf{x}'|\boldsymbol{\theta}) = \sum_j \lambda_j(\mathbf{x})\lambda_j(\mathbf{x}')k_j(\mathbf{x}, \mathbf{x}'|\boldsymbol{\theta})$. A related approach of additive GPs was used by Kandasamy et al. [23] for Bayesian optimization of high dimensional functions under the assumption that the actual function is a combination of lower dimensional functions.

For Bayesian optimization, we propose the combination of a local and a global kernels and with multivariate normal distributions as weighting functions. We have called this kernel, the Spartan kernel:

$$\begin{aligned} k(\mathbf{x}, \mathbf{x}'|\boldsymbol{\theta}^S) = {} & \lambda^{(g)}(\mathbf{x})\lambda^{(g)}(\mathbf{x}')k^{(g)}(\mathbf{x}, \mathbf{x}'|\boldsymbol{\theta}^g) \\ & + \lambda^{(l)}(\mathbf{x}|\boldsymbol{\theta}^p)\lambda^{(l)}(\mathbf{x}'|\boldsymbol{\theta}^p)k^{(l)}(\mathbf{x}, \mathbf{x}'|\boldsymbol{\theta}^l) \end{aligned} \tag{4}$$

where the normalized local weight $\lambda^{(l)}(\mathbf{x}|\boldsymbol{\theta}^p)$ includes parameters to move the influence region. The unnormalized weights ω are defined as:

$$\begin{aligned} \omega^{(g)} &= \mathcal{N}\left(\psi, \mathbf{I}\sigma_g^2\right) \\ \omega^{(l)} &= \mathcal{N}\left(\boldsymbol{\theta}^p, \mathbf{I}\sigma_l^2\right) \end{aligned} \tag{5}$$

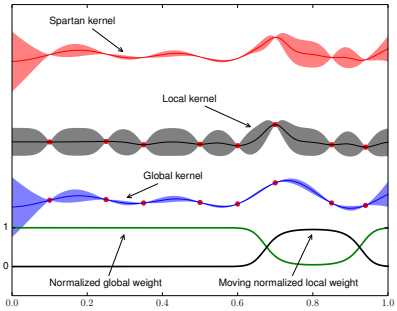

Fig. 1. Representation of the Spartan kernel in SBO. Typically, the local and global kernels have a small and large length-scale respectively. The influence of each kernel is represented by the normalized weight at the bottom of the plot. Note how the kernel with small length-scale produces larger uncertainties which is an advantage for fast exploitation, but it can perform poorly for global exploration as it tends to sample equally almost everywhere. On the other hand, the kernel with large length-scale provides a better global estimate, but it can be too constrained locally.

where ψ and $\boldsymbol{\theta}^p$ can be seen as the centers of the influence region of each kernel while σ_g^2 and σ_l^2 represents the area of influence. The Spartan kernel is shown in Figure 1.

1) Global weight parameters: Unless we have prior knowledge of the function, the parameters of the global weight are mostly irrelevant. In most applications, we can use a uniform weight, which can be easily approximated with a large σ_g^2. For example, assuming a normalized input space $\mathcal{X} = [0, 1]^d$, we can set $\psi = [0.5]^d$ and $\sigma_g^2 = 10$.

2) Local weight parameters: For the local weight, we propose to consider the center of the influence area of the local kernel $\boldsymbol{\theta}^p$ as part of the hyperparameters for the Spartan kernel, that also includes the parameters of the local and global kernels, that is:

$$\boldsymbol{\theta}^S = [\boldsymbol{\theta}^g, \boldsymbol{\theta}_1^l, \ldots, \boldsymbol{\theta}_M^l, \boldsymbol{\theta}^p]$$

Thus, $\boldsymbol{\theta}^p$ is also estimated from data. The variance of the weight of the local kernel (extension of the influence area) could also be adapted including the terms σ^2 as part of the kernel hyperparameters $\boldsymbol{\theta}^S$. However, in that case the problem becomes ill-posed, resulting in overfitting. Instead of adding regularization terms, we found simpler to fix the value of σ_l^2 or fix the number of samples within $2\sigma_l^2$. The second method has the advantage that, while doing exploitation, as the number of local samples increase, the area gets narrower, allowing better local modeling.

3) Learning the parameters: As commented in Section II, when new data is available, all the parameters are updated using MCMC. Therefore, the position of the local kernel $\boldsymbol{\theta}^p$ is moved each iteration to represent the posterior, as can be seen in Figure 2. Due to the sampling behavior in Bayesian optimization, we found that it intrinsically moves more likely towards the more densely sampled areas in many problems, which corresponds to the location of the function minima. Furthermore, as we have m MCMC samples, there are m different positions for the local kernel $\boldsymbol{\Theta}^p = \{\boldsymbol{\theta}_i^p\}_{i=1}^m$.

It is important to note that, although we have described SBO relying on GPs and EI, the Spartan kernel also works with other popular models such as Student-t processes, variational GPs; other criteria such as *upper confidence bound*

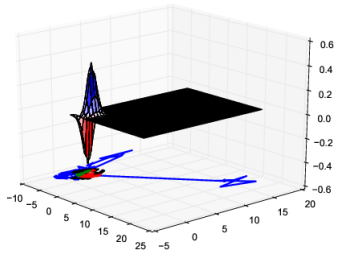

Fig. 2. Gramacy function [21]. The path bellow the surface represents the location of the local kernel as being sampled by MCMC for each BO iteration. Clearly, it moves towards the nonstationary section of the function. For visualization, the path is colored depending on the iteration (start → blue → black → green → red → end).

[16], *relative entropy* [24]; and specific configuration such as trajectory aware kernels [15], [8].

The intuition behind SBO is the same of the sampling strategies in Bayesian optimization: *the aim of the model is not to approximate the target function precisely in every point, but to provide information about the location of the minimum*. Many optimization problems are difficult due to the fact that the region near the minimum is heteroscedastic, i.e.: it has higher variability than the rest of the space, like the function in Figure 2. In this case, SBO greatly improves the performance of the state of the art in Bayesian optimization.

IV. Active policy search

Reinforcement learning algorithms usually rely on variants of the Bellman equation to optimize the policy step by step considering each instantaneous reward r_t separately. Some algorithms also rely on partial or total knowledge of the transition model. Other methods tackle the optimization problem directly, considering the problem of finding the optimal policy as a stochastic optimization problem, being called *direct policy search*. In that way, the use of Bayesian optimization for reinforcement learning falls in the family of direct policy search, being called *active policy search* [13] for its connection with active learning and how samples are carefully selected based on current information.

The main advantage of using Bayesian optimization to compute the optimal policy is that it can be done with very little information. In fact, as soon as we are able to simulate scenarios and return the total reward $\sum_{t=1}^{T} r_t$, we do not need to access the dynamics, the instantaneous reward or the current state of the system. Furthermore, there is no need for space or action discretization, building complex features or *tile coding* [25]. We found that for many control problems, a simple, low dimensional policy is able to achieve state-of-the-art performance if properly optimized. We also solve stochasticity by running each episode several times and returning the average reward, as an approximation of the expected reward.

A frequent issue for applying general purpose optimization algorithms for policy search is that, in many problems, the occurrence of *failure* states or scenarios results in large discontinuities or flat regions due to large penalties for all failing policies. This is opposed to the behavior of the reward near the optimal policy where small variations on a suboptimal policy can considerably change the performance achieved. Therefore, the resulting reward function presents a nonstationary behavior with respect to the policy.

V. Evaluation and results

We have selected a variety of benchmarks from the optimization, RL/control and robotics literature. For evaluation purposes and to highlight the robustness of SBO, we took the simpler approach to fix the variance of ω_l. We found that a value of $\sigma_l^2 = 0.05$ was robust enough in all the experiments once the input space was normalized to the unit hypercube.

Although this method allows for any combination of local and global kernels, for the purpose of evaluation, we used the Matérn kernel from equation (3) with ARD for both –local and global– kernels. Furthermore, the length-scales were initialized with the same prior for the both kernels. Therefore, we let the data determine which kernel has smaller length-scale. We found that the typical result is the behavior from Figure 1. However, in some problems, the method may learn a model where the local kernel has a larger length-scale (i.e.: smoother and smaller variance) than the global kernel, which may also improve the convergence in plateau-like functions. Besides, if the target function is stationary, the system might end up learning a similar length-scale for both kernels, thus being equivalent to a single kernel. We can say that standard BO is a special case of SBO where the local and global kernels are the same.

Given that for a single Matérn kernel with ARD, the number of kernel hyperparameters is the dimensionality of the problem, d, the number of hyperparameters for the Spartan kernel in this setup is $3d$. As we will see in the experiments, this is the only drawback of SBO compared to standard BO, as it requires running MCMC in a larger dimensional space, which results in higher computational cost. However, because SBO is more efficient, the extra computational cost can be easily compensated by a reduced number of samples.

We implemented Spartan Bayesian Optimization (SBO) using the BayesOpt library [26]. This allowed us to evaluate the setup for many surrogates and criteria. For comparison, we also implemented the input warping (WARP) method from Snoek et al. [20]. To our knowledge, this is the only Bayesian optimization algorithm that has deal with nonstationarity using GPs in a fully correlated way.

For the experiments reported here we used: a Gaussian process with unit mean function like in Jones et al. [5]. For BO and WARP we also used a Matérn kernel $\nu = 5/2$ with ARD. The kernel hyperparameters, including $\boldsymbol{\theta}^p$ for SBO and (α, β) for the warping functions were estimated using MCMC (i.e.: slice sampling). Due to the computational burden of MCMC, we used a small number of samples (i.e.: 10), while trying to decorrelate every resample with large burn-in periods (i.e.: 100 samples) as in Snoek et al. [6]. All experiments were repeated 20 times using common random numbers. The starting function evaluations of the plots represents the initial design using *latin hypercube sampling*. Plots show the average results over all runs with 95% confidence intervals.

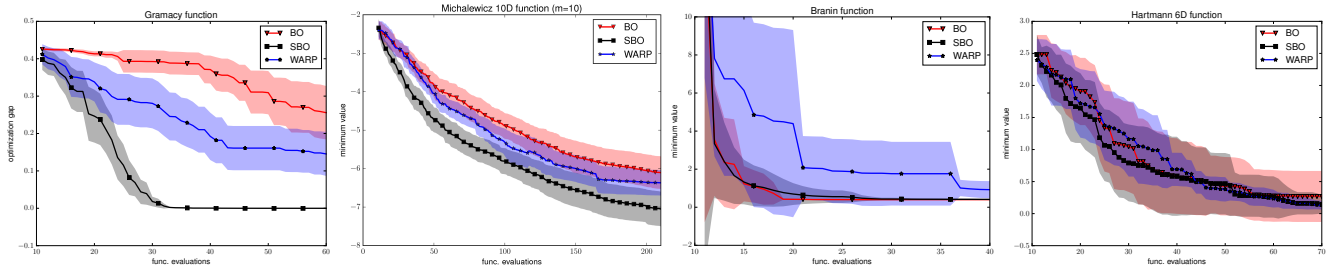

Fig. 3. a) Gramacy function. b) Michalewicz 10D function with m=10. c) Branin-Hoo function, d) Hartmann 6D function. For the nonstationary functions, a) and b), the proposed SBO method results in an outstanding convergence speed compared to the state of the art. For the Gramacy function, SBO finds the minimum in about 30 function evaluations in all tests. For the stationaty functions, c) and d) BO and SBO are barely identical, with SBO producing more accurate results and with smaller uncertainty. The WARP method sometimes improves over standard BO (a,b and d) or produces worse results (c).

A. *Optimization benchmarks*

We evaluated the algorithms on a set of well-known test functions for global optimization both smooth or with sharp drops (see Figure 3). The selected functions have become a standard in the Bayesian optimization literature.

First, we show the results for the optimization benchmarks for functions with sharp drops where local optimization is fundamental. Figure 3a shows the results of the Gramacy function found in [21]. Our method (SBO) provides excellent results, by reaching the optimum in less than 35 samples for all tests. Because the function is nonstationary, the WARP method outperforms standard BO, but its convergence is much slower than SBO. Figure 3b shows the results for the Michalewicz function. This function has a parameter to define the dimensionality d and the steepness m. This function is known to be a hard benchmarks in global optimization due to the many local minima ($d!$) and steep drops. We used $d = 10$ and $m = 10$, resulting in 3628800 minima with very steep edges. For this problem, SBO clearly outperforms the rest of the methods by a large margin.

We have also evaluated stationary and smooth functions with large valleys near the global minimum. Bayesian optimization is more suitable for these functions and standard kernels perform well in general. Therefore, there is barely room from improvement. However, we show that, even in this situation, SBO is equal or better than standard BO. In terms of accuracy, there is no penalty for the extra complexity of the SBO model, while the WARP method may require more samples due to the extra complexity. For the Branin-Hoo function (see Figure 3c), the differences between SBO and BO are insignificant. Meanwhile, the warping function in WARP introduces an exploration bias at early stages, resulting in slower convergence. For the Hartmann 6D function (see Figure 3d), the differences are small, which imply that the function is most likely stationary and simple to exploit. However, we can see that during the final iterations, nonstationary methods (SBO and WARP) slightly improve standard BO, both in terms of average result (convergence) and variance (robustness).

B. *Reinforcement learning experiments*

We evaluated SBO with several RL/control problems. They all rely on continuous states, actions. We assume the problems are episodic, with a finite time horizon. We have compared our method in three well-known benchmarks with different level of complexity.

1) Walker: The first problem is learning the controller of a three limb robot walker presented in Westervelt et al. [27] using their Matlab code. The controller modulates the walking pattern of a simple biped robot. The desired behavior is a fast upright walking pattern, the reward is based on the walking speed with a penalty for not maintaining the upright position. The dynamic controller has 8 continuous parameters. The walker problem was already used as a Bayesian optimization benchmark [24].

2) Mountain Car: The second problem is the mountain car problem [25] based on a Python implementation from Martin H. [28]. The state of the system is the car horizontal position. The action is the horizontal acceleration $a \in [-1, 1]$. Contrary to the many solutions that discretize both the state and action space, we can directly deal with continuous states and actions. The policy is a simple linear perceptron model inspired by Brochu et al. [29]. The potentially unbounded policy parameters $\mathbf{w} = \{w_i\}_{i=1}^{7}$ are computed as $\mathbf{w} = \tan\left((\pi - \epsilon_\pi)\,\mathbf{w}_{01} - \pi/2\right)$ where $\mathbf{w}_{01}$ are the policy parameters bounded in the $[0, 1]^7$ space. The term $\epsilon_\pi > 0$ was used to avoid $w_i \to \infty$.

3) Helicopter hovering: The third problem is the hovering helicopter from the RL-competition [1]. It is one of the most challenging scenarios of the competition, being presented in all the editions since 2008. This problem is based on a simulator of the XCell Tempest aerobatic helicopter. The simulator model was learned based on actual data from the helicopter using apprenticeship learning [30]. The model was used to learn a policy that was later used in the real robot. The simulator included several difficult wind conditions. The state space is 12D (position, orientation, translational velocity and rotational velocity) and the action is 4D (forward-backward cyclic pitch, lateral cyclic pitch, main collective pitch and tail collective pitch). The reward is a quadratic function that penalizes both the state error (inaccuracy) and the action (energy). Each episode is run during 10 seconds (6000 control steps). If the simulator enters a terminal state (crash), a large negative reward is given, corresponding to

[1] http://www.rl-competition.org/

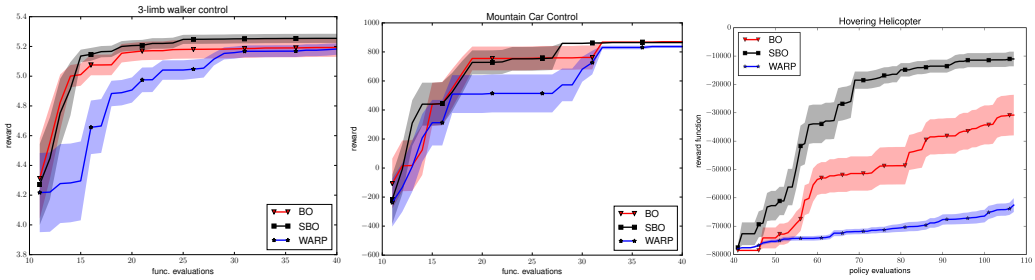

Fig. 4. Total reward for: a) the three limb walker, b) the mountain car and c) the hovering helicopter control problem. For the first problem, SBO is able to achieve higher reward, while other methods get stuck in a local maxima. For the mountain car, SBO is able to achieve maximum performance in all trials after just 27 policy trials (17 iterations + 10 initial samples). For the helicopter problem, BO and WARP have slow convergence, because many policies results in an early crash, providing almost no information. However, SBO is able to exploit good policies and quickly improve the performance.

getting the most negative reward achievable for the remaining time.

We used the weak baseline controller that was included with the helicopter model. This weak controller is a simple linear policy with 12 parameters (weights). In theory, this controller is enough to avoid crashing but is not very robust. We show how this policy can be easily improved with few iterations. In this case, initial exploration of the parameter space is specially important because the number of policies not crashing in few control steps is very small. For most policies, the reward is the most negative reward achievable. Thus, in this case, we have used Sobol sequences for the initial samples of Bayesian optimization. These samples are deterministic, therefore we guarantee that the same number of non-crashing policies are sampled for every trial and every algorithm. We also increased the number of initial points to 40 due to the higher dimensionality.

Figure 4 shows the performance for the three limb walker presented, the mountain car and the helicopter problem. In all cases, the results obtained by SBO were more efficient in terms on number of trials and accuracy, with respect to standard BO and WARP. Furthermore, we found that the results of SBO were comparable to those obtained by popular reinforcement learning solvers like SARSA [25], but with much less information and prior knowledge about the problem. For the helicopter problem, other solutions found in the literature require a larger number of scenarios/trials to achieve similar performance [31].

C. Automatic wing design using CFD simulations

The next test is to find the shape of the wing that minimizes the drag while maintaining enough lift. Wing design using computational fluid dynamics (CFD) simulators is also a well known difficult optimization problem due to the chaotic nature of fluid dynamics [32]. Even though we use a commercial CFD software (XFlow) to simulate the wind tunnel, sample efficiency is mandatory as an average CFD simulation can still take days or months of CPU time.

First, as a common practice in this kind of problems, we assumed a 2D simulation of the fluid along the profile of the wing. This is a reasonable assumption for wings with large aspect ratio (large span compared to the chord), and it considerably reduces the wall time of each simulation from days to hours. For the parametrization of the profile, there

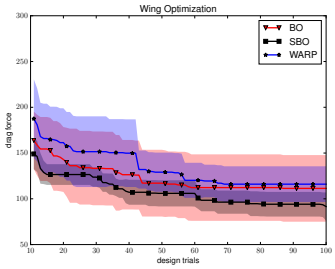

Fig. 5. Results for the wing design optimization (10 runs per plot).

are many alternatives based on geometric or manufacturing principles. In our case, we used Bezier curves for their simplicity to generate the corresponding shape. However, note that Bayesian optimization is agnostic of the geometric parametrization and any other parametrization could also be used. The Bezier curve of the wing was based on 7 control points, that is, 14 parameters which were reduced to 5 by adding some physical and manufacturing restrictions.

The problem of minimizing the drag directly is that the best solutions tend to generate flat wings that do not provide enough lift for the plane. As a simple approach, we added a large penalty to the wings without enough lift. We also found that, due to fluid dynamics, the drag value was very chaotic. For example, the flow near the trailing edge can transition from laminar to turbulent regime due to a small change in the wing shape. Thus, the resulting forces are completely different, increasing the drag and reducing the lift. Figure 5 shows how both BO and WARP fail to find the optimum wing shape, under these conditions. However, SBO finds a better wing shape and in very few iterations.

D. Computational cost

The main difference between the three methods (BO, SBO and WARP) in terms of the algorithm is within the kernel function $k(\cdot, \cdot)$, which includes the evaluation of the weights in SBO and the evaluation of the warping function (the cumulative density function of a Beta distribution) in WARP. We found that the time differences between the algorithms were mainly driven by the dimensionality and shape of the posterior distribution of the kernel hyperparameters because MCMC was the main bottleneck. Also, the evaluation of the Beta CDF was more involved and computationally expensive

TABLE I

AVERAGE CPU TIME FOR THE TOTAL OPTIMIZATION (IN SECONDS).

Time(s)	Gram.	Branin	Hartm.	Michal.	Walker	MCar
#dims	2	2	6	10	8	7
#evals	60	40	70	210	40	40
BO	120	171	460	8 360	47	38
SBO	2 481	3 732	10 415	225 313	440	797
WARP	13 929	28 474	188 942	4 445 854	20 271	18 972

than the evaluation of the Matérn kernel or the Gaussian weights. That extra cost became an important factor as the kernel function is called millions of times for each Bayesian optimization run.

Table I shows the average CPU time of the different experiments for the total number of function evaluations. We did not include the helicopter and wing problems because both rely on multiple process synchronization, convoluting the measurements, although the relative computation time were similar to other examples.

VI. CONCLUSIONS

In this paper, we have presented a new algorithm called Spartan Bayesian Optimization (SBO) in the context of *active policy search* for robot control. Our method combines a local and a global kernel in a single adaptive kernel to deal with the exploration/exploitation trade-off and the inherent nonstationarity in the search process during policy search using Bayesian optimization. For nonstationary problems, like robot control problems, the method provides excellent results compared to standard Bayesian optimization and the state of the art method for nonstationarity. Furthermore, SBO also performs well in stationary problems by improving local refinement while retaining global exploration capabilities. We evaluated the algorithm extensively in standard optimization benchmarks and control/reinforcement learning scenarios. Moreover, our contribution can be directly applied to other setups. As an example, we also evaluated SBO in an autonomous wind design problem. The results have shown that SBO increases the convergence speed and reduces the number of samples in many problems. In addition, we have shown how SBO is more efficient in terms of CPU usage than other nonstationary methods for Bayesian optimization.

REFERENCES

[1] M. Deisenroth, G. Neumann, and J. Peters, "A survey on policy search for robotics," *Foundations and Trends in Robotics*, vol. 2, no. 1-2, pp. 1–142, 2013.

[2] J. Peters and S. Schaal, "Policy gradient methods for robotics," in *Proceedings of the IEEE International Conference on Intelligent Robotics Systems (IROS 2006)*, 2006.

[3] A. Cully, J. Clune, D. Tarapore, and J.-B. Mouret, "Robots that can adapt like animals," *Nature*, vol. 521, p. 503507, 2015.

[4] J. Mockus, V. Tiesis, and A. Zilinskas, "The application of Bayesian methods for seeking the extremum," in *Towards Global Optimisation 2*, L. Dixon and G. Szego, Eds. Elsevier, 1978, pp. 117–129.

[5] D. R. Jones, M. Schonlau, and W. J. Welch, "Efficient global optimization of expensive black-box functions," *Journal of Global Optimization*, vol. 13, no. 4, pp. 455–492, 1998.

[6] J. Snoek, H. Larochelle, and R. Adams, "Practical Bayesian optimization of machine learning algorithms," in *NIPS*, 2012, pp. 2960–2968.

[7] R. Martinez-Cantin, N. de Freitas, E. Brochu, J. Castellanos, and A. Doucet, "A Bayesian exploration-exploitation approach for optimal online sensing and planning with a visually guided mobile robot." *Autonomous Robots*, vol. 27, no. 3, pp. 93–103, 2009.

[8] R. Marchant and F. Ramos, "Bayesian optimisation for informative continuous path planning," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2014.

[9] R. Calandra, A. Seyfarth, J. Peters, and M. Deisenroth, "Bayesian optimization for learning gaits under uncertainty," *Annals of Mathematics and Artificial Intelligence (AMAI)*, vol. 1 1, pp. 1–19 1–19, 2015.

[10] M. Tesch, J. Schneider, and H. Choset, "Adapting control policies for expensive systems to changing environments," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2011.

[11] O. Kroemer, R. Detry, J. Piater, and J. Peters, "Combining active learning and reactive control for robot grasping," *Robotics and Autonomous Systems*, vol. 58, no. 9, pp. 1105–1116, 2010.

[12] J. Nogueira, R. Martinez-Cantin, A. Bernardino, and L. Jamone, "Unscented Bayesian optimization for safe robot grasping," in *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, 2016.

[13] R. Martinez-Cantin, N. de Freitas, A. Doucet, and J. A. Castellanos, "Active policy learning for robot planning and exploration under uncertainty," in *Robotics: Science and Systems*, 2007.

[14] S. R. Kuindersma, R. A. Grupen, and A. G. Barto, "Variable risk control via stochastic optimization," *The International Journal of Robotics Research*, vol. 32, no. 7, pp. 806–825, 2013.

[15] A. Wilson, A. Fern, and P. Tadepalli, "Using trajectory data to improve bayesian optimization for reinforcement learning," *Journal of Machine Learning Research*, vol. 15, pp. 253–282, 2014.

[16] N. Srinivas, A. Krause, S. Kakade, and M. Seeger, "Gaussian process optimization in the bandit setting: No regret and experimental design," in *Proc. International Conference on Machine Learning (ICML)*, 2010.

[17] C. E. Rasmussen and C. K. Williams, *Gaussian Processes for Machine Learning*. Cambridge, Massachusetts: The MIT Press, 2006.

[18] Z. Wang, F. Hutter, M. Zoghi, D. Matheson, and N. de Freitas, "Bayesian optimization in a billion dimensions via random embeddings," *JAIR*, vol. 55, pp. 361–387, 2016.

[19] M. A. Taddy, H. K. Lee, G. A. Gray, and J. D. Griffin, "Bayesian guided pattern search for robust local optimization," *Technometrics*, vol. 51, no. 4, pp. 389–401, 2009.

[20] J. Snoek, K. Swersky, R. S. Zemel, and R. P. Adams, "Input warping for Bayesian optimization of non-stationary functions," in *International Conference on Machine Learning*, 2014.

[21] J. M. Assael, Z. Wang, and N. de Freitas, "Heteroscedastic treed bayesian optimisation," arXiv, Tech. Rep., 2014.

[22] A. Krause and C. Guestrin, "Nonmyopic active learning of Gaussian processes: an exploration-exploitation approach," in *International Conference on Machine Learning (ICML)*, Corvallis, Oregon, June 2007.

[23] K. Kandasamy, J. Schneider, and B. Poczos, "High dimensional bayesian optimisation and bandits via additive models," in *International Conference on Machine Learning (ICML)*, 2015.

[24] J. M. Hernández-Lobato, M. W. Hoffman, and Z. Ghahramani, "Predictive entropy search for efficient global optimization of black-box functions," in *NIPS*, 2014.

[25] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. The MIT Press, 1998.

[26] R. Martinez-Cantin, "BayesOpt: A Bayesian optimization library for nonlinear optimization, experimental design and bandits," *Journal of Machine Learning Research*, vol. 15, pp. 3735–3739, 2014.

[27] E. R. Westervelt, J. W. Grizzle, C. Chevallereau, J. H. Choi, and B. Morris, *Feedback control of dynamic bipedal robot locomotion*. CRC press, 2007, vol. 28.

[28] J. A. Martin H., "A reinforcement learning environment in Python," https://jamh-web.appspot.com/download.htm.

[29] E. Brochu, V. Cora, and N. de Freitas, "A tutorial on Bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning," arXiv.org, eprint arXiv:1012.2599, December 2010.

[30] P. Abbeel, A. Coates, M. Quigley, and A. Ng, "An application of reinforcement learning to aerobatic helicopter flight," in *NIPS*, 2006.

[31] R. Koppejan and S. Whiteson, "Neuroevolutionary reinforcement learning for generalized control of simulated helicopters," *Evolutionary intelligence*, vol. 4, no. 4, pp. 219–241, 2011.

[32] A. I. Forrester, N. W. Bressloff, and A. J. Keane, "Optimization using surrogate models and partially converged computational fluid dynamics simulations," *Proceedings of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, vol. 462, no. 2071, pp. 2177–2204, 2006.