HAL
open science

Black-Box Data-efficient Policy Search for Robotics

Konstantinos Chatzilygeroudis, Roberto Rama, Rituraj Kaushik, Dorian Goepp, Vassilis Vassiliades, Jean-Baptiste Mouret

To cite this version:

Konstantinos Chatzilygeroudis, Roberto Rama, Rituraj Kaushik, Dorian Goepp, Vassilis Vassiliades, et al.. Black-Box Data-efficient Policy Search for Robotics. IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Sep 2017, Vancouver, Canada. hal-01576683

HAL Id: hal-01576683

https://inria.hal.science/hal-01576683v1

Submitted on 23 Aug 2017

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Black-Box Data-efficient Policy Search for Robotics

Konstantinos Chatzilygeroudis, Roberto Rama, Rituraj Kaushik, Dorian Goepp,
Vassilis Vassiliades and Jean-Baptiste Mouret*

***Abstract*— The most data-efficient algorithms for reinforcement learning (RL) in robotics are based on uncertain dynamical models: after each episode, they first learn a dynamical model of the robot, then they use an optimization algorithm to find a policy that maximizes the expected return given the model and its uncertainties. It is often believed that this optimization can be tractable only if analytical, gradient-based algorithms are used; however, these algorithms require using specific families of reward functions and policies, which greatly limits the flexibility of the overall approach. In this paper, we introduce a novel model-based RL algorithm, called Black-DROPS (Black-box Data-efficient RObot Policy Search) that: (1) does not impose any constraint on the reward function or the policy (they are treated as *black-boxes*), (2) is as data-efficient as the state-of-the-art algorithm for data-efficient RL in robotics, and (3) is as fast (or faster) than analytical approaches when several cores are available. The key idea is to replace the gradient-based optimization algorithm with a parallel, black-box algorithm that takes into account the model uncertainties. We demonstrate the performance of our new algorithm on two standard control benchmark problems (in simulation) and a low-cost robotic manipulator (with a real robot).**

***Index Terms*— Learning and Adaptive Systems, Data-Efficient Learning**

I. Introduction

Reinforcement Learning (RL) can help robots adapt to unforeseen situations, such as being damaged [1], [2], [3] or stranded [4]. Rather than aborting their mission when something goes wrong, they could carry on by discovering new behaviors autonomously. Nevertheless, to be useful in such situations, learning has to happen in a few minutes, typically within a few trials. This scarcity of data makes it difficult to exploit the many recent machine learning techniques (e.g., deep learning) that rely on the availability of very large datasets or fast simulations[1] [6]. As a consequence, robot learning has to consider other approaches, with the explicit goal of requiring as little *interaction time* as possible between the robot and the environment.

When data are scarce, a general principle is to extract as much information as possible from them. In the case of robotics, this means that all the state variables that are available should be collected at every time-step and be used by the learning algorithm. This contrasts with many direct policy search approaches (e.g., policy gradient algorithms [7], [8] or Bayesian optimization [2], [9], [10]) which only use the (cumulative) reward at the end of each episode.

One of the best ways to take advantage of this sequential state recording is to learn a dynamical model of the robot [11], and then exploit it either for model-predictive control [12] or to find an optimal policy offline [7]. However, such approaches assume that the model is "good enough" to predict future states for all the possible states. This is often not the case when only a few episodes have been performed, as many states have not been observed yet. Learning with a dynamical model therefore often requires acquiring enough points to learn an accurate model, which, in turn, increases the interaction time.

This challenge can be overcome by taking into account the uncertainty of the dynamical model: if the algorithm "knows" that a prediction is unreliable, it can balance the risks of trying something that might fail with the potential benefits. The PILCO (Probabilistic Inference for Learning COntrol) algorithm [13], which is one of the state-of-the-art algorithms for data-efficient model-based policy search, follows this strategy by alternating between two steps, (1) learning a dynamical model with Gaussian processes [14], (2) using a gradient-based optimizer to search for a policy that maximizes the expected reward, taking the uncertainty of the model into account. Thanks to this process, PILCO achieves remarkable data-efficiency.

Nevertheless, analytical algorithms like PILCO have two main issues that may not be apparent at first sight. First, they impose several constraints on the reward functions and policies that prevent the use of arbitrary rewards (e.g., PILCO can only be used with distance-based rewards so far) and of non-derivable policies (e.g., parameterized state automata, like in [9]). Second, they require a large *computation time* to optimize the policy (e.g., typically more than 5 minutes on a modern computer between each episode for the cart-pole benchmark), because they rely on computationally expensive methods to do approximate inference for each step of the policy evaluation [13].

In this paper, we introduce a novel policy search algorithm that tackles these two problems while maintaining the data-efficiency of analytical algorithms. Our main insight is that while the analytic approach is efficient on a sequential computer, it cannot take advantage of the multi-core archi-

*Corresponding author: jean-baptiste.mouret@inria.fr
All authors have the following affiliations:
- Inria, Villers-lès-Nancy, F-54600, France
- CNRS, Loria, UMR 7503, Vandœuvre-lès-Nancy, F-54500, France
- Université de Lorraine, Loria, UMR 7503, Vandœuvre-lès-Nancy, F-54500, France
This work received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (GA no. 637972, project "ResiBots") and the European Commission through the project H2020 AnDy (GA no. 731540).

[1]As an illustration, the deep Q-learning algorithm needed about 38 days of interaction to learn to play Atari 2600 games [5] (only four possible actions), which would be hardly conceivable to a achieve with a robot.

tectures now present in every computer. By contrast, Monte Carlo approaches and population-based black-box optimizers like CMA-ES [15] (1) do not put any constraint on the reward functions and policies, and (2) are straightforward to parallelize, which can make them competitive with analytical approaches when several cores are available. Our second insight is that it is not necessary to explicitly compute accurate approximations of the expected reward when the optimization is performed with rank-based algorithms designed for noisy functions (e.g., CMA-ES [15]), which saves a lot of computation: only the ranking of potential solutions matters. Thus, it is possible to define a data-efficient, black-box policy search algorithm that is competitive with gradient-based, analytical approaches.

We call our algorithm Black-DROPS, for Black-box Data-efficient RObot Policy Search. It is a model-based policy search algorithm which:

- takes into account the uncertainty of the dynamical model when searching for a policy;
- is as data-efficient as state-of-the-art, analytical algorithms, that is, it requires similar *interaction time*;
- performs a more global search than gradient-based algorithms, that is, it can escape from some local optima;
- is at least as fast as state-of-the-art, analytical methods when several cores are used, that is, it requires similar or lower *computation time*; in addition, it is likely to be faster with future computers with more cores;
- does not impose any constraint on the reward function (in particular, the reward function can be learned);
- does not impose any constraint on the policy representation (any parameterized policy can be used).

We demonstrate these features with two families of policies, feed-forward neural networks and Gaussian processes, applied to two classic control benchmarks in simulation, the inverted pendulum and the cart-pole swing-up, as well as a physical 4-DOF robotic arm.

II. Related Work

Direct policy search (PS) methods have been successful in robotics as they can easily be applied in high-dimensional continuous state-action RL problems [7]. REINFORCE [16] is an early policy gradient method which performs exploration of the action space using probabilistic policies. It suffers, however, from slow convergence due to the high variance in its gradient estimates. Policy gradients with parameter-based exploration (PGPE) [17] address this problem by transferring exploration to parameter space. In particular, PGPE samples deterministic policies at the start of each episode by maintaining a separate Gaussian distribution for each parameter of the policy, whose mean and variance are adapted during training. The PoWER (Policy learning by Weighting Exploration with the Returns) algorithm [18] uses probability-weighted averaging, which has the property of following the natural gradient without computing it [18]. PoWER, however, assumes that the immediate rewards sum to a constant number and are always positive, which complicates the design of reward functions. The Policy Improvements with Path Integrals (PI^2) [19] algorithm does not make such an assumption. When the reward function is compatible with both PoWER and PI^2, the algorithms have identical performance [19].

A limitation of PGPE is that it does not consider any correlations between dimensions in parameter space. This can be addressed by the Natural Evolution Strategies (NES) [20] and Covariance Matrix Adaptation ES (CMA-ES) [15] families of algorithms, which are population-based Black-Box Optimizers (BBO). Both NES and CMA-ES iteratively update a search distribution by calculating an estimated gradient on the distribution parameters (mean and covariance matrix). At each generation, they sample a set of solutions (i.e., policy parameters) and rank them based on their fitness (i.e., expected return). NES performs gradient ascent along the natural gradient, which normalizes the update with respect to uncertainty. CMA-ES updates the distribution by exploiting the technique of evolution paths to average-out random effects over the generations. NES and CMA-ES are closely related, as the latter performs an approximate natural gradient ascent [21]. Interestingly, a variant of PI^2 with a simplified parameter perturbation and update method outperforms PI^2 and was shown to be a special case of CMA-ES [22].

In general, any BBO can be used for direct PS. Bayesian optimization [23] is a particular family of BBO that can be very data-efficient by building a surrogate model of the objective function (i.e., the expected return) and exploring this model in a clever way (e.g., using upper confidence bounds [24]). It can drastically decrease the evaluation time when optimizing gaits [10] or when finding compensatory behaviors for damaged robots [2].

The data-efficiency of direct PS can be further increased by learning the model (i.e., transition and reward function) of the system from data and inferring the optimal policy from the model [7]. Probabilistic models have been more successful than deterministic ones, as they provide an estimate about the uncertainty of their approximation which can be incorporated into long-term planning [13]. For example, local linear models have been used in [25], [26], [27], Gaussian processes (GPs) in [28], [13], [29] and least-squares conditional density estimation in [30].

Early examples of such model-based PS include applications on helicopter hovering [25], [26] and blimp control [28]. These works employ the PEGASUS algorithm which can transform a stochastic Markov Decision Process (MDP) or partially-observable MDP (POMDP) into a deterministic POMDP [31]. It does so by fixing in advance the sequence of random numbers associated with the state transitions. This simple modification significantly reduces the time needed to optimize the policy, as it removes the noise from the evaluation of an initially noisy objective function.

Both the model-based PGPE [30] and the PILCO [13] algorithm use gradient-based policy updates. Rather than using Monte Carlo sampling, as in model-based PGPE, PILCO performs deterministic approximate inference by explicitly incorporating the model uncertainty into long-term predictions. This procedure is done by approximating the

probability distribution over trajectories with a Gaussian that has the same mean and covariance (moment matching). The gradient of the expected return is then computed analytically with respect to the policy parameters. This makes PILCO dependent on differentiable reward and policy functions.

Gradient-free methods, such as the Model-Based Relative Entropy PS (M-REPS) [29] and the Model-Based Guided PS (M-GPS) [27], do not have these requirements. Both algorithms place a KL-divergence constraint on the cost function to bound the distance between the old trajectory distribution and the newly estimated one at each policy improvement step. This constraint limits the information loss of the updates [32]. M-GPS turns the policy optimization problem into a supervised learning one, allowing the use of high-dimensional policy representations such as deep neural networks. However, M-GPS makes strong assumptions about the task at hand, by assuming that time-varying Gaussians can approximate the local dynamics. In contrast, M-REPS uses GPs for model learning, and the REPS algorithm (which can be seen as a BBO) for policy search.

Overall, the current consensus [7] is that (1) model-based algorithms are more data-efficient than direct PS, (2) in model-based PS, it is crucial to account for potential model errors during policy learning, and (3) deterministic approximate inference and analytic computation of policy gradients is required to make model-based PS computationally tractable. In this paper, we focus on the latter and explore a parallel BBO algorithm for policy optimization.

III. Problem Formulation

We consider dynamical systems of the form:

$$\mathbf{x}_{t+1} = \mathbf{x}_t + f(\mathbf{x}_t, \mathbf{u}_t) + \mathbf{w} \tag{1}$$

with continuous-valued states $\mathbf{x} \in \mathbb{R}^E$ and controls $\mathbf{u} \in \mathbb{R}^F$, i.i.d. Gaussian system noise $\mathbf{w}$, and unknown transition dynamics f.

Our objective is to find a deterministic *policy* π, $\mathbf{u} = \pi(\mathbf{x}|\boldsymbol{\theta})$, which maximizes the *expected long-term reward* when following policy π for T time steps:

$$J(\boldsymbol{\theta}) = \mathbb{E}\left[\sum_{t=1}^{T} r(\mathbf{x}_t)\Big|\boldsymbol{\theta}\right] \tag{2}$$

where $r(\mathbf{x}_t)$ is the immediate reward of being in state $\mathbf{x}$ at time t. We assume that π is a function parameterized by $\boldsymbol{\theta} \in \mathbb{R}^{\Theta}$ and that the immediate reward function $r(\mathbf{x}) \in \mathbb{R}$ might be unknown to the learning algorithm.

IV. Approach

A. Learning dynamics model with Gaussian processes

We would like to have a model $\hat{f}$ that approximates as accurately as possible the unknown dynamics f of our systems and provides uncertainty information. We rely on Gaussian processes (GPs) to do so. A GP is an extension of multivariate Gaussian distribution to an infinite-dimension stochastic process for which any finite combination of dimensions will be a Gaussian distribution [14].

As inputs, we use tuples made of the state vector $\mathbf{x}_t$ and the action vector $\mathbf{u}_t$, that is, $\tilde{\mathbf{x}}_t = (\mathbf{x}_t, \mathbf{u}_t) \in \mathbb{R}^{E+F}$; as training targets, we use the difference between the current state vector and the next one: $\boldsymbol{\Delta}_{\mathbf{x}_t} = \mathbf{x}_{t+1} - \mathbf{x}_t \in \mathbb{R}^E$. We use E independent GPs to model each dimension of the difference vector $\boldsymbol{\Delta}_{\mathbf{x}_t}$. For each dimension $d = 1 \dots E$ of $\boldsymbol{\Delta}_{\mathbf{x}_t}$, the GP is computed as ($k_{\hat{f}_d}$ is the kernel function):

$$\hat{f}_d(\tilde{\mathbf{x}}) \sim \mathcal{GP}(\mu_{\hat{f}_d}(\tilde{\mathbf{x}}), k_{\hat{f}_d}(\tilde{\mathbf{x}}, \tilde{\mathbf{x}}')) \tag{3}$$

Assuming $D^d_{1:t} = \{f_d(\tilde{\mathbf{x}}_1), ..., f_d(\tilde{\mathbf{x}}_t)\}$ is a set of observations, we can query the GP at a new input point $\tilde{\mathbf{x}}_*$:

$$p(\hat{f}_d(\tilde{\mathbf{x}}_*)|D^d_{1:t}, \tilde{\mathbf{x}}_*) = \mathcal{N}(\mu_{\hat{f}_d}(\tilde{\mathbf{x}}_*), \sigma^2_{\hat{f}_d}(\tilde{\mathbf{x}}_*)) \tag{4}$$

The mean and variance predictions of this GP are computed using a kernel vector $\boldsymbol{k}_{\hat{f}_d} = k(D^d_{1:t}, \tilde{\mathbf{x}}_*)$, and a kernel matrix $K_{\hat{f}_d}$, with entries $K^{ij}_{\hat{f}_d} = k_{\hat{f}_d}(\tilde{\mathbf{x}}_i, \tilde{\mathbf{x}}_j)$:

$$\begin{aligned} \mu_{\hat{f}_d}(\tilde{\mathbf{x}}_*) &= \boldsymbol{k}^T_{\hat{f}_d} K^{-1}_{\hat{f}_d} D^d_{1:t} \\ \sigma^2_{\hat{f}_d}(\tilde{\mathbf{x}}_*) &= k_{\hat{f}_d}(\tilde{\mathbf{x}}_*, \tilde{\mathbf{x}}_*) - \boldsymbol{k}^T_{\hat{f}_d} K^{-1}_{\hat{f}_d} \boldsymbol{k}_{\hat{f}_d} \end{aligned} \tag{5}$$

In this paper, we use the exponential kernel with automatic relevance determination [14]:

$$\begin{aligned} k_{\hat{f}_d}(\tilde{\mathbf{x}}_p, \tilde{\mathbf{x}}_q) = \sigma^2_d \exp(-\frac{1}{2}(\tilde{\mathbf{x}}_p - \tilde{\mathbf{x}}_q)^T \boldsymbol{\Lambda}^{-1}_d (\tilde{\mathbf{x}}_p - \tilde{\mathbf{x}}_q)) \\ + \delta_{pq}\sigma^2_{n_d} \end{aligned} \tag{6}$$

where δ_{pq} equals to 1 when $p = q$ and 0 otherwise, and $[\boldsymbol{\Lambda}_d, \sigma^2_d, \sigma^2_{n_d}]$ is the vector of hyper-parameters of the kernel (length scales for each dimension of the observations, signal variance and noise) found through Maximum Likelihood Estimation [14]. We use the limbo C++11 library for GP regression [33].

B. Learning the immediate reward function with a GP

Similarly to the dynamical model, we use a GP to learn the immediate reward function, which associates a reward $r(\mathbf{x}) \in \mathbb{R}$ to each state $\mathbf{x}$:

$$\hat{r}(\mathbf{x}) \sim \mathcal{GP}(\mu_r(\mathbf{x}), k_r(\mathbf{x}, \mathbf{x}')) \tag{7}$$

The GP predictions are calculated similarly to Eq. 4 and 5.

C. Policy Evaluation

Our goal is to maximize the expected cumulative reward (Eq. 2), which requires predicting the state evolution given an uncertain transition model and an uncertain reward model. To do so in a deterministic way[2], PILCO proposes to approximate the distribution of state $\mathbf{x}_{t+1}$ given the distribution of state $\mathbf{x}_t$ and the action $\mathbf{u}_t$ using moment matching [13], and then propagates from state to state until reaching the end of the episode; however, this sequential approach accumulates errors over time, is not easy to parallelize, and is computationally expensive. As an alternative, we can compute a Monte Carlo approximation of the final distribution: at each step we sample a new state according to the GP of the model and its reward according to the reward model, query the

[2]Additionally, PILCO requires the reward function to be known a priori.

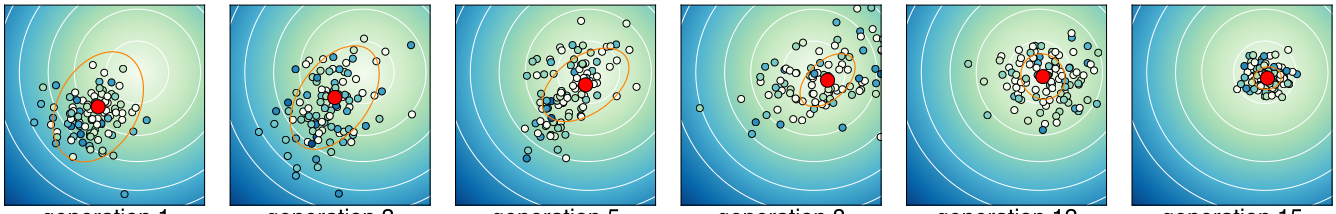

Fig. 1. Illustration of an actual run with CMA-ES on a simple 2-D, noisy problem. The performance landscape is pictured on the background. Each colored disk is a candidate from the population (the color represents its performance with the same color scale as the landscape in the background). If the color of a disk is the same as the background, then noise did not change the performance. The mean m_k of the best μ candidates is pictured as a red disk, and the covariance of the μ best individuals as an orange ellipse. In only a few generations, and in spite of the noise, CMA-ES identifies the optimum of the function. Please note that in this example we use a bigger population than in our work, as advised by the authors of CMA-ES [15].

policy to choose the actions, and use this new state to sample the next state. By performing this process many times, we can get a good estimate of the expected cumulative reward, but many samples are needed to obtain a good estimate [29].

Here we adopt a different approach. Like in Monte Carlo estimation, we propagate from state to state by sampling according to the models. However, we consider that each of these rollouts is a measurement of a function $G(\boldsymbol{\theta})$ that is the actual function $J(\boldsymbol{\theta})$ perturbed by a noise $N(\boldsymbol{\theta})$:

$$\begin{aligned} G(\boldsymbol{\theta}) &= J(\boldsymbol{\theta}) + N(\boldsymbol{\theta}) \\ &= \sum_{t=1}^{T} \hat{r}(\mathbf{x}_{t-1} + \hat{f}(\mathbf{x}_{t-1}, \mathbf{u}_{t-1})) \end{aligned} \tag{8}$$

where $\hat{f}(\mathbf{x}_{t-1}, \mathbf{u}_{t-1}) \sim \mathcal{N}(\mu_{\hat{f}}(\tilde{\mathbf{x}}_{t-1}), \Sigma_{\hat{f}}(\tilde{\mathbf{x}}_{t-1}))$ is a realization of a normally distributed random vector according to Eq. 4, $\hat{r}(\mathbf{x}) \sim \mathcal{N}(\mu_r(\mathbf{x}), \sigma_r^2(\mathbf{x}))$ is a realization of a normally distributed random value according to Eq. 7 and $\mathbf{u}_{t-1} = \pi(\mathbf{x}_{t-1}|\boldsymbol{\theta})$. We would like to maximize its expectation:

$$\begin{aligned} \mathbb{E}\Big[G(\boldsymbol{\theta})\Big] &= \mathbb{E}\Big[J(\boldsymbol{\theta}) + N(\boldsymbol{\theta})\Big] \\ &= \mathbb{E}\Big[J(\boldsymbol{\theta})\Big] + \mathbb{E}\Big[N(\boldsymbol{\theta})\Big] \end{aligned} \tag{9}$$

And since $\forall x \ \mathbb{E}\Big[\mathbb{E}[x]\Big] = \mathbb{E}[x]$:

$$\mathbb{E}\Big[G(\boldsymbol{\theta})\Big] = J(\boldsymbol{\theta}) + \mathbb{E}\Big[N(\boldsymbol{\theta})\Big] \tag{10}$$

We assume that $\mathbb{E}[N(\boldsymbol{\theta})] = 0$ for all $\boldsymbol{\theta} \in \mathbb{R}^{\Theta}$ and therefore maximizing $\mathbb{E}[G(\boldsymbol{\theta})]$ is equivalent to maximizing $J(\boldsymbol{\theta})$.

D. Policy search

Seeing the maximization of $J(\boldsymbol{\theta})$ as the optimization of a noisy function allows us to maximize it without computing or estimating it explicitly: we only use the noisy measurements in the optimization algorithm. To do so, we build on all the work about noisy function optimization [34], [35], and especially on CMA-ES, one of the most successful black-box optimizer for noisy functions [15]. CMA-ES (Fig. 1) performs four steps at each generation k:

(1) sample λ new candidates according to a multi-variate Gaussian distribution of mean m_k and covariance $\sigma_k^2 C_k$, that is, $\boldsymbol{\theta}_i \sim \mathcal{N}(m_k, \sigma_k^2 C_k)$ for $i \in 1, \cdots, \lambda$;
(2) rank the λ sampled candidates based on their (noisy) performance $G(\boldsymbol{\theta}_i)$;
(3) compute m_{k+1} by averaging the μ best candidates: $m_{k+1} = \frac{1}{\mu}\sum_{i=1}^{\mu} \boldsymbol{\theta}_i$;
(4) update the covariance matrix to reflect the distribution of the μ best candidates.

Overall, these steps are only marginally impacted by noise in the performance function, as confirmed by empirical experiments with noisy functions [34]. More precisely, the only decision that matters is whether a solution belongs to the μ best ones (step 2), that is, a precise ranking is not needed and errors can only happen at the boundaries between the low-performing and high-performing solutions. In addition, if a candidate is misclassified because of the noise, the impact of this error will be smoothed out by the average when computing m_{k+1} (step 3). One can also observe that because CMA-ES samples several solutions around a mean m_k, it performs many evaluations of similar parameters, which are then averaged: this *implicit averaging* [34], [36] has many similarities with re-evaluating noisy solutions to estimate their expectation.

Modern implementations of CMA-ES add several refinements to compute the covariance matrix, to take into account successive steps, and to restart the process with more exploration when it reaches an optimum. In this work, we use BIPOP-CMA-ES with restarts [37], which is one of best CMA-ES variants on benchmarks with both noiseless and noisy functions [37], [38].

On top of these refinements, we follow the strategy proposed by Hansen et al. [35] to improve the behavior of CMA-ES with noisy functions (called UH-CMA-ES). The starting idea is that uncertainty is a problem for a rank-based algorithm if and only if, for two potential candidates $\boldsymbol{\theta}_1$ and $\boldsymbol{\theta}_2$ the variation due to $N(\boldsymbol{\theta}_1)$ and $N(\boldsymbol{\theta}_2)$ exceeds the difference $|J(\boldsymbol{\theta}_1) - J(\boldsymbol{\theta}_2)|$ and thus their ordering is changed. If the variation tends to exceed this difference, we cannot conclude only from two measurements $G(\boldsymbol{\theta}_1)$, $G(\boldsymbol{\theta}_2)$, whether $J(\boldsymbol{\theta}_1) > J(\boldsymbol{\theta}_2)$ or $J(\boldsymbol{\theta}_1) < J(\boldsymbol{\theta}_2)$ holds. If we view $|J(\boldsymbol{\theta}_1) - J(\boldsymbol{\theta}_2)|$ as the signal and the variations due to $N(\boldsymbol{\theta})$ as noise, then it follows that one way to improve the quality of the ranking without re-evaluating solutions many times (which would reduce noise) is to increase the signal.

We therefore implement the following strategy: (1) at each generation, we quantify the uncertainty of the ranking by re-evaluating $\lambda_{\text{reev}} < \lambda$ randomly selected candidates from the population and count the number of rank changes (see [35] for a detailed description of uncertainty quantification), (2) if the uncertainty is above a user-defined threshold, then we increase the variance of the population (σ_k in step 1 of CMA-

ES). In addition of reducing the uncertainty of the ranking when needed, this strategy has an interesting consequence: in uncertain search-space regions, CMA-ES moves faster (it makes bigger steps), which means that the algorithm favors regions that are more certain (when they are as promising as uncertain regions) and is not "trapped" in uncertain regions. We use a modified version of the *libcmaes* C++11 library[3].

E. Black-box Data-efficient Robot Policy Search

Putting everything together, we get the Black-DROPS algorithm (Alg. 1). Firstly, N_R random episodes of T time steps are conducted on the robot (Alg. 1: lines 4-12). In the learning loop, first we learn a probabilistic model of the dynamics and a model of the reward function, and then we optimize $\mathbb{E}\big[G(\boldsymbol{\theta})\big]$ given this learned models using BIPOP-CMAES with uncertainty handling (Alg. 1: lines 14-16). Lastly, the best policy $\pi_{\boldsymbol{\theta}^*}$ is executed on the robot, more data is collected and the main loop continues until the task is learned.

Algorithm 1 Black-DROPS

1: **procedure** BLACK-DROPS
2: Define policy $\pi : \mathbf{x} \times \boldsymbol{\theta} \rightarrow \mathbf{u}$
3: $D = \emptyset$
4: **for** $i = 1 \rightarrow N_R$ **do** ▷ N_R random episodes
5: Set robot to initial state $\mathbf{x}_0$
6: **for** $j = 0 \rightarrow T-1$ **do** ▷ perform the episode
7: $\mathbf{u}_j$ = random_action()
8: $\mathbf{x}_{j+1}, r(\mathbf{x}_{j+1})$ = execute_on_robot($\mathbf{u}_j$)
9: $D = D \cup \{\tilde{\mathbf{x}}_j \rightarrow \boldsymbol{\Delta}_{\mathbf{x}_j}\}$
10: $R = R \cup \{\mathbf{x}_{j+1} \rightarrow r(\mathbf{x}_{j+1})\}$
11: **end for**
12: **end for**
13: **while** $task \neq solved$ **do**
14: Model learning: train E GPs given data D
15: Reward learning: train 1 GP given data R
16: $\boldsymbol{\theta}^* = \text{argmax}_{\boldsymbol{\theta}} \mathbb{E}\Big[G(\boldsymbol{\theta})\Big]$ using BIPOP-CMA-ES ▷ Sec. IV-D
17: Set robot to initial state $\mathbf{x}_0$
18: **for** $j = 0 \rightarrow T-1$ **do** ▷ perform the episode
19: $\mathbf{u}_j = \pi(\mathbf{x}_j | \boldsymbol{\theta}^*)$
20: $\mathbf{x}_{j+1}, r(\mathbf{x}_{j+1})$ = execute_on_robot($\mathbf{u}_j$)
21: $D = D \cup \{\tilde{\mathbf{x}}_j \rightarrow \boldsymbol{\Delta}_{\mathbf{x}_j}\}$
22: $R = R \cup \{\mathbf{x}_{j+1} \rightarrow r(\mathbf{x}_{j+1})\}$
23: **end for**
24: **end while**
25: **end procedure**

V. Experimental Setup

A. Policy Representations

To highlight the flexibility of Black-DROPS, we use a GP-based policy [13] and a feed-forward neural network-based one. Any other parameterized policy can be used (e.g., dynamic movement primitives).

1) GP Policy: If we only consider the mean, a Gaussian process can be used to map states to actions, that is, to define a policy:

$$\pi(\mathbf{x}) = \mathbf{u}_{\text{max}} \kappa(\mu(\mathbf{x})) = \mathbf{u}_{\text{max}} \kappa(\boldsymbol{k}^T (K + \sigma_n^2 I)^{-1} \mathbf{x}) \quad (11)$$

where $\mathbf{u}_{\text{max}}$ is the maximum value of $\mathbf{u}$ (different for each action dimension), κ is a squashing function like the one used in [13], $\mathbf{x}$ is the input state vector to the policy, K is the covariance matrix and its elements are computed using the exponential kernel with automatic relevance determination as in Eq. 6. Here, we set signal noise, $\sigma_n^2 = 0.01$. The vector $[\boldsymbol{\Lambda}, \sigma_f^2]$ and the pseudo-observations (*inputs* & *targets* when learning the GP) constitute the parameters of the policy.

2) Neural Network Policy: The network function of the ith layer of the network is given by $\mathbf{y}_i = \phi_i(\mathbf{W}_i \mathbf{y}_{i-1} + \mathbf{b}_i)$, where $\mathbf{W}_i$ and $\mathbf{b}_i$ are the weight matrix and bias vector, $\mathbf{y}_{i-1}$ and $\mathbf{y}_i$ are the input and output vector and ϕ_i is the activation function. Throughout the paper, we use configurations with one hidden layer and the hyperbolic tangent as the activation function ϕ for all the layers, leading to:

$$\pi(\mathbf{x}) = \mathbf{u}_{\text{max}} \mathbf{y}_1 = \mathbf{u}_{\text{max}} \phi(\mathbf{W}_1 \mathbf{y}_0 + \mathbf{b}_1)$$
$$\text{and } \mathbf{y}_0 = \phi(\mathbf{W}_0 \mathbf{x} + \mathbf{b}_0) \quad (12)$$

B. Metrics

1) Reward as interaction time increases: This metric assesses the quality of the solutions and the data-efficiency of each algorithm.

2) Speed-up when more cores are available: This metric assesses how well each algorithm scales as the available hardware resources increase, independently of the particular implementation (e.g., MATLAB vs C++).

C. Remarks

We evaluate Black-DROPS on the pendulum and cart-pole tasks and compare it to PILCO using 120 replicates over different CPU configurations. As an additional baseline, we evaluate a variant of our approach using deterministic GP models of the dynamics (i.e., using only the mean of the GPs) to quantify the importance of considering the uncertainty (variance) of the model in policy optimization. For Black-DROPS and the baseline we use two different policies: a neural network policy (with one hidden layer and 10 hidden units) and a GP policy (with 10 and 20 pseudo-observations for the pendulum and the cart-pole task respectively). For PILCO we used only the GP policy with the same parameters as for the other algorithms.

We additionally evaluate Black-DROPS on a 4-DOF arm task to validate that it can be used with more complex and interesting robots, that it can be used when the reward function is unknown, and that it works on a real robotic platform. We use only the neural network policy for this task, as it performed better in the simpler benchmarks.

For all the tasks, an episode corresponds to applying the same policy for a duration of $4\,s$ and the sampling/control rate is $10Hz$. The source code of the experiments can be found at `https://github.com/resibots/blackdrops`.

VI. Results

A. Task 1: Inverted Pendulum

This simulated system consists of a freely swinging pendulum with mass $m = 1\,kg$ and length $l = 1\,m$. The objective is to learn a controller to swing the pendulum up and to balance it in the inverted position applying a torque.

[3] `https://github.com/beniz/libcmaes`

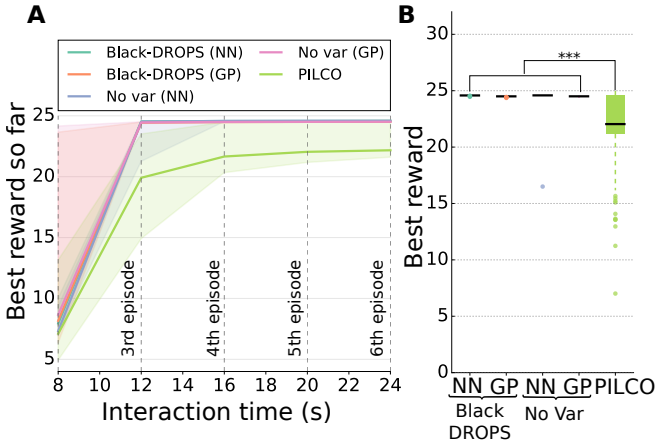

Fig. 2. Results for the pendulum task (120 replicates): (**A**) Best reward found per episode. The lines are median values and the shaded regions the 25th and 75th percentiles. Black-DROPS converges to higher quality solutions in fewer episodes than PILCO and has considerably less variance. (**B**) Best reward after 4 episodes. The box plots show the median (black line) and the interquartile range (25th and 75th percentiles); the whiskers extend to the most extreme data points not considered outliers, and outliers are plotted individually. Our approach outperforms PILCO in the quality of the controllers found. The number of stars indicates that the p-value of the Mann-Whitney U test is less than 0.05, 0.01, 0.001 and 0.0001 respectively.

- **State:** $\mathbf{x}_{pend} = [\dot{\theta}, \theta] \in \mathbb{R}^2$, $\mathbf{x}_0 = [0, 0]$.
- **Actions:** $\mathbf{u}_{pend} = u_{pend} \in \mathbb{R}$, $-2.5 \leq u_{pend} \leq 2.5\,N$.
- To avoid angle discontinuities, we transform the input of the GPs, the reward function, and the policy to be:

$$\mathbf{x}_{input} = [\dot{\theta}, \cos(\theta), \sin(\theta)] \in \mathbb{R}^3$$

  The MATLAB implementation of PILCO uses this transformation by default[4].
- **Reward:** We use the same reward function as PILCO[5]. This is a saturating distance-based reward function:

$$r(\mathbf{x}) = \exp(-\frac{1}{2\sigma_c^2}(\mathbf{x} - \mathbf{x}_*)^T \mathbf{Q} (\mathbf{x} - \mathbf{x}_*)) \qquad (13)$$

  where σ_c controls the width of the reward function, $\mathbf{Q}$ is a weight matrix, $\mathbf{x}_*$ is the target state and $r(\mathbf{x}) \in [0, 1]$. We set $\mathbf{x}_* = [*, \cos(\pi), \sin(\pi)]$, $\sigma_c = 0.25$ and $\mathbf{Q}$ to ignore the angular velocity $\dot{\theta}$ of the pendulum.

In this task, both Black-DROPS and PILCO solve the task in about 3 episodes ($12\,s$ of interaction time — including the random episode, Fig. 2**A**), but Black-DROPS finds higher-performing policies (Fig. 2**A-B**), with both the neural network and the GP policy. When the number of cores is increased, the computation time required by Black-DROPS decreases almost linearly (Fig. 3**A**), whereas PILCO only slightly benefits from having more than 4 cores (as PILCO is not a parallel algorithm, we think that the improvement between 1 and 4 cores stems from MATLAB's ability to parallelize some linear algebra operations). With more than 8 cores, Black-DROPS outperforms PILCO in computation speed and can be from 1.25 to 3.3 times faster when 12 cores

[4]http://mlg.eng.cam.ac.uk/pilco/

[5]PILCO uses a cost function, but it is straightforward to transform it in a reward function.

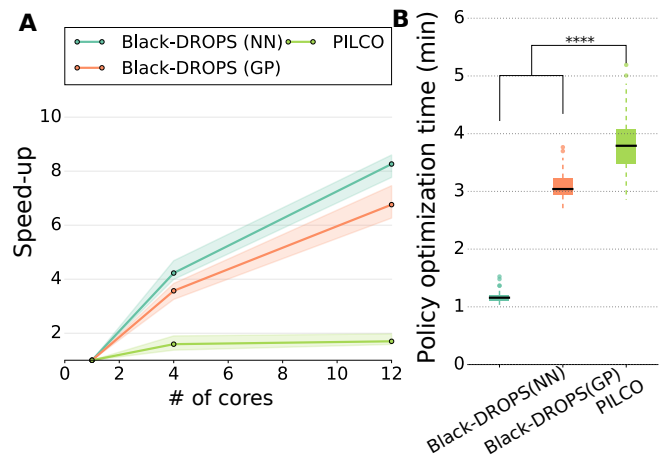

Fig. 3. Timing for the the pendulum task: (**A**) Speed-up (for total policy optimization time after 4 episodes) achieved when using multiple cores. The lines are median values over 30 runs and the shaded regions the 25th and 75th percentiles. As more cores are being used, Black-DROPS greatly benefits from it and has up to 8x speed-up when 12 cores are used. (**B**) Total policy optimization time after 4 episodes when 12 cores are available.

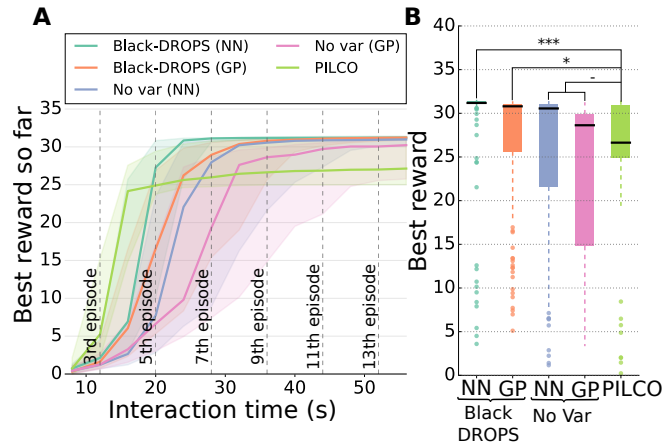

Fig. 4. Results for the cart-pole task (120 replicates): (**A**) Best reward found per episode. Black-DROPS converges to higher quality solutions in about the same number of episodes as PILCO and has less variance. (**B**) Best reward after 8 episodes. Our approach outperforms PILCO in the quality of the controllers found. See Fig. 2 for legend.

are available[6] (Fig. 3**B**). In addition, given a budget of 15 episodes, Black-DROPS succeeds more often than PILCO in finding a working policy (Table I): Black-DROPS always solves the task whereas PILCO fails once in ten runs.

Surprisingly, taking into account the uncertainty of the model does not seem to be necessary in this task (the "No Var" baselines perform the same as Black-DROPS, see Fig. 2). This result most probably stems from the fact that the dynamics of the system are simple enough for the GPs to model almost perfectly with one or two episodes.

B. Task 2: Cart-pole Swing-Up

This simulated system consists of a cart with mass $M = 0.5\,kg$ running on a track and a freely swinging pendulum

[6]While some of the runtime differences can stem from the language used (e.g., C++ being faster than MATLAB or MATLAB being faster at matrix computations), what matters is that a parallel algorithm with enough CPUs can eventually outperform a sequential gradient-based approach.

TABLE I

Algorithm	Success Rate	
	Pendulum	Cart-pole
Black-DROPS (NN)	100%	93.33%
Black-DROPS (GP)	100%	90.83%
No Var (NN)	100%	90%
No Var (GP)	100%	85%
PILCO	89.16%	92.5%

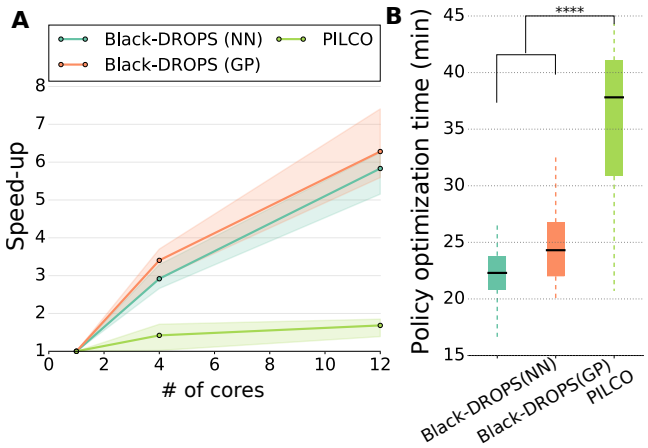

Fig. 5. Timing for the cart-pole task: (**A**) Speed-up (for total policy optimization time after 8 episodes) achieved when using multiple cores. As more cores are being used, Black-DROPS greatly benefits from it and has a 6x speed-up when 12 cores are used. (**B**) Total policy optimization time after 8 episodes when 12 cores are available. Black-DROPS is around 1.6x faster than PILCO. See Fig. 3 for legend and number of replicates.

with mass $m = 0.5\,kg$ and length $l = 0.5\,m$ attached to the cart. The state of the system contains the position of the cart, the velocity of the cart, the angle of the pendulum and the angular velocity of the pendulum. The objective is to learn a controller that applies horizontal forces on the cart to swing the pendulum up and balance it in the inverted position in the middle of the track.

- **State:** $\mathbf{x}_{cp} = [\dot{x}, x, \dot{\theta}, \theta] \in \mathbb{R}^4$, $\mathbf{x}_0 = [0, 0, 0, 0]$.
- **Actions:** $\mathbf{u}_{cp} = u_{cp} \in \mathbb{R}$, $-10 \leq u_{cp} \leq 10\,N$.
- To avoid angle discontinuities, we transform the input of the GPs, the reward, and the policy to be:

$$\mathbf{x}_{input} = [\dot{x}, x, \dot{\theta}, \cos(\theta), \sin(\theta)] \in \mathbb{R}^5$$

- **Reward:** We set $\mathbf{x}_* = [*, 0, *, \cos(\pi), \sin(\pi)]$, $\sigma_c = 0.25$, $\mathbf{Q}$ to ignore $\dot{x}$ and $\dot{\theta}$, and use Eq. 13.

The results for the cart-pole are very similar to those obtained with the inverted pendulum (Fig. 4**A-B**): Black-DROPS and PILCO have similar data-efficiency (about 4 episodes or $16\,s$ of interaction time to find a working policy with PILCO, 5 episodes or $20\,s$ with Black-DROPS) but Black-DROPS finds higher-performing policies, most probably because its search algorithm (CMA-ES with restarts) is less local than the gradient-based optimizer used in PILCO.

Using the variance helps more in this task than in the pendulum task: the variants of Black-DROPS without uncertainty handling are less data-efficient and have more variance. However, they are still able to find policies that are higher-performing than those found by PILCO (Fig. 4**B**). In terms of success rate, Black-DROPS fails as often as PILCO and, as expected, the variants fail more often than PILCO and Black-DROPS (see Table I).

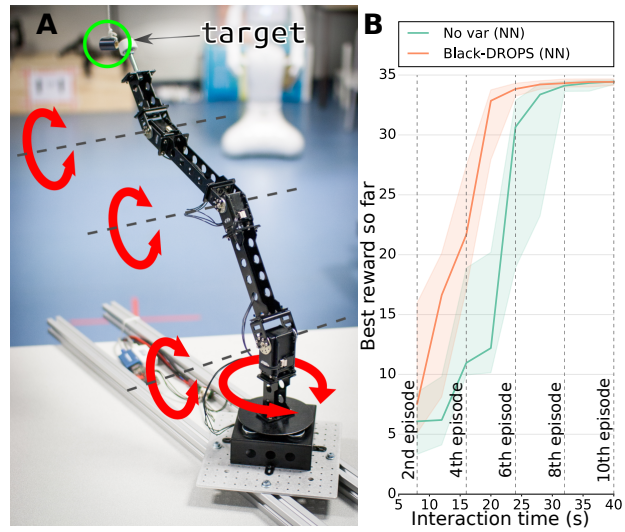

Fig. 6. Manipulator task (10 replicates for each treatment).

Similar to the pendulum task, here also Black-DROPS takes advantage of multiple cores to highly speed-up its computation and is 1.6 times faster than PILCO when 12 cores are available (Fig. 5).

C. Task 3: 4-DOF Manipulator

We applied Black-DROPS on a physical velocity-controlled 4-DOF robotic arm (Fig. 6, 10 replicates). We assume that we can only observe the angles of the joints of the arm and that the reward function r_{arm} is initially unknown. The arm begins in the up-right position and the objective is to learn a controller so that the end-effector quickly reaches a certain position (shown in Fig. 6**A**). We compare Black-DROPS with the baseline without variance.

- **State:** $\mathbf{x}_{arm} = [q_0, q_1, q_2, q_3] \in \mathbb{R}^4$, $\mathbf{x}_0 = [0, 0, 0, 0]$.
- **Actions:** $\mathbf{u}_{arm} = [v_0, v_1, v_2, v_3] \in \mathbb{R}^4$, where $-1.0 \leq v_i \leq 1.0\,rad/s, \quad i = 0, 1, 2, 3$.
- **Reward:** The unknown (to the algorithm) reward function has a form similar to Eq. 13:

$$r_{arm}(\mathbf{x}) = \exp(-\frac{1}{2\sigma_c^2}\|\mathbf{p_x} - \mathbf{p}_*\|) \tag{14}$$

  where $\sigma_c = 0.1$, $\mathbf{p_x}$ corresponds to the end-effector position in state $\mathbf{x}$, $\mathbf{p}_*$ is the goal position of the end-effector and $r_{arm}(\mathbf{x}) \in [0, 1]$.
- To avoid angle discontinuities, we transform the input to the GPs and the policy to be:

$$\mathbf{x}_{input} = [\cos(q_0), \sin(q_0), \cos(q_1), \sin(q_1), \cos(q_2), \sin(q_2), \cos(q_3), \sin(q_3)] \in \mathbb{R}^8$$

The results show that Black-DROPS is able to find a working policy within 5 episodes (including the initial random one) and outperforms the baseline which needs around 6 episodes (Fig. 6**B**). Black-DROPS, also, shows less variance and converges to high quality controllers faster (6 episodes vs 8-9). A video of a typical run is available as supplementary material (also at `https://youtu.be/kTEyYiIFGPM`).

VII. Conclusion and discussion

Black-DROPS lifts several constraints imposed by analytical approaches (reward and policy types) while being competitive in terms of data-efficiency and computation time. In three different tasks, it achieved similar results as the state-of-the-art (PILCO) while being faster when multiple cores are used. We expect that the ability of Black-DROPS to scale with the number of cores will be even more beneficial on future computers with more cores and/or with GPUs.

Using the variance in the optimization is one of the key components to learn with as little interaction time as possible. However, the learned dynamics models are only confident in areas of the state space previously visited and thus could drive the optimization into local optima when multiple and diverse solutions exist. In future work, we will investigate ways of exploring more without impacting data-efficiency.

Finally, even with 12 cores, although faster than analytical approaches, Black-DROPS still requires around 25 minutes for completing 8 episodes in the cart-pole task. The main issue is the cubic computational complexity of the prediction of the GPs (we are doing around 64,000,000 GP queries per episode). Possible solutions include using local GPs [39], [40] or to stop using GPs and make use of recent advances in neural networks with uncertain predictions [41], [42].

References

[1] J. Carlson and R. R. Murphy, "How UGVs physically fail in the field," *IEEE Trans. on Robotics*, vol. 21, no. 3, pp. 423–437, 2005.

[2] A. Cully, J. Clune, D. Tarapore, and J.-B. Mouret, "Robots that can adapt like animals," *Nature*, vol. 521, no. 7553, pp. 503–507, 2015.

[3] K. Chatzilygeroudis, V. Vassiliades, and J.-B. Mouret, "Reset-free Trial-and-Error Learning for Data-Efficient Robot Damage Recovery," *arXiv:1610.04213*, 2016.

[4] K. Nagatani *et al.*, "Emergency response to the nuclear accident at the Fukushima Daiichi Nuclear Power Plants using mobile rescue robots," *Journal of Field Robotics*, vol. 30, no. 1, pp. 44–63, 2013.

[5] V. Mnih *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[6] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

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

[8] N. Kohl and P. Stone, "Policy gradient reinforcement learning for fast quadrupedal locomotion," in *Proc. of ICRA*, vol. 3. IEEE, 2004, pp. 2619–2624.

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

[10] D. J. Lizotte, T. Wang, M. H. Bowling, and D. Schuurmans, "Automatic gait optimization with Gaussian process regression," in *IJCAI*, vol. 7, 2007, pp. 944–949.

[11] D. Nguyen-Tuong and J. Peters, "Model learning for robot control: a survey," *Cognitive Processing*, vol. 12, no. 4, pp. 319–340, 2011.

[12] E. F. Camacho and C. B. Alba, *Model predictive control*. Springer Science & Business Media, 2013.

[13] M. P. Deisenroth, D. Fox, and C. E. Rasmussen, "Gaussian processes for data-efficient learning in robotics and control," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 37, no. 2, pp. 408–423, 2015.

[14] C. E. Rasmussen and C. K. I. Williams, *Gaussian processes for machine learning*. MIT Press, 2006.

[15] N. Hansen and A. Ostermeier, "Completely derandomized self-adaptation in evolution strategies," *Evolutionary computation*, vol. 9, no. 2, pp. 159–195, 2001.

[16] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine learning*, vol. 8, no. 3-4, pp. 229–256, 1992.

[17] F. Sehnke *et al.*, "Policy gradients with parameter-based exploration for control," in *Proc. of Artificial Neural Networks*. Springer, 2008, pp. 387–396.

[18] J. Kober and J. Peters, "Policy search for motor primitives in robotics," *Machine Learning*, vol. 84, pp. 171–203, 2011.

[19] E. Theodorou, J. Buchli, and S. Schaal, "A generalized path integral control approach to reinforcement learning," *JMLR*, vol. 11, pp. 3137–3181, 2010.

[20] D. Wierstra *et al.*, "Natural evolution strategies," *JMLR*, vol. 15, no. 1, pp. 949–980, 2014.

[21] Y. Akimoto, Y. Nagata, I. Ono, and S. Kobayashi, "Bidirectional relation between CMA evolution strategies and natural evolution strategies," in *Proc. of PPSN*. Springer, 2010, pp. 154–163.

[22] F. Stulp and O. Sigaud, "Robot skill learning: From reinforcement learning to evolution strategies," *Paladyn, Journal of Behavioral Robotics*, vol. 4, no. 1, pp. 49–61, 2013.

[23] B. Shahriari, K. Swersky, Z. Wang, R. P. Adams, and N. de Freitas, "Taking the human out of the loop: A review of Bayesian optimization," *Proc. of the IEEE*, vol. 104, no. 1, pp. 148–175, 2016.

[24] P. Auer, "Using confidence bounds for exploitation-exploration trade-offs," *JMLR*, vol. 3, pp. 397–422, 2002.

[25] J. A. Bagnell and J. G. Schneider, "Autonomous helicopter control using reinforcement learning policy search methods," in *Proc. of ICRA*, vol. 2. IEEE, 2001, pp. 1615–1620.

[26] A. Y. Ng, A. Coates, M. Diel, V. Ganapathi, J. Schulte, B. Tse, E. Berger, and E. Liang, "Autonomous inverted helicopter flight via reinforcement learning," in *Experimental Robotics IX*. Springer, 2006, pp. 363–372.

[27] S. Levine and P. Abbeel, "Learning neural network policies with guided policy search under unknown dynamics," in *Proc. of NIPS*, 2014, pp. 1071–1079.

[28] J. Ko, D. J. Klein, D. Fox, and D. Haehnel, "Gaussian processes and reinforcement learning for identification and control of an autonomous blimp," in *Proc. of ICRA*, 2007, pp. 742–747.

[29] A. Kupcsik *et al.*, "Model-based contextual policy search for data-efficient generalization of robot skills," *Artificial Intelligence*, 2014.

[30] V. Tangkaratt, S. Mori, T. Zhao, J. Morimoto, and M. Sugiyama, "Model-based policy gradients with parameter-based exploration by least-squares conditional density estimation," *Neural Networks*, vol. 57, pp. 128–140, 2014.

[31] A. Y. Ng and M. Jordan, "PEGASUS: a policy search method for large MDPs and POMDPs," in *Proc. of Uncertainty in Artificial Intelligence*. Morgan Kaufmann, 2000, pp. 406–415.

[32] J. Peters and S. Schaal, "Reinforcement learning of motor skills with policy gradients," *Neural Networks*, vol. 21, no. 4, pp. 682–697, 2008.

[33] A. Cully, K. Chatzilygeroudis, F. Allocati, and J.-B. Mouret, "Limbo: A fast and flexible library for Bayesian optimization," *arxiv:1611.07343*, 2016.

[34] Y. Jin and J. Branke, "Evolutionary optimization in uncertain environments-a survey," *IEEE Trans. on Evolutionary Computation*, vol. 9, no. 3, pp. 303–317, 2005.

[35] N. Hansen, A. S. Niederberger, L. Guzzella, and P. Koumoutsakos, "A method for handling uncertainty in evolutionary optimization with an application to feedback control of combustion," *IEEE Trans. on Evolutionary Computation*, vol. 13, no. 1, pp. 180–197, 2009.

[36] B. L. Miller and D. E. Goldberg, "Genetic algorithms, selection schemes, and the varying effects of noise," *Evolutionary Computation*, vol. 4, no. 2, pp. 113–131, 1996.

[37] N. Hansen, "Benchmarking a BI-population CMA-ES on the BBOB-2009 function testbed," in *Proc. of GECCO*. ACM, 2009, pp. 2389–2396.

[38] ——, "Benchmarking a BI-population CMA-ES on the BBOB-2009 noisy testbed," in *Proc. of GECCO*. ACM, 2009, pp. 2397–2402.

[39] C. Park and D. Apley, "Patchwork kriging for large-scale gaussian process regression," *arXiv preprint arXiv:1701.06655*, 2017.

[40] M. P. Deisenroth and J. W. Ng, "Distributed gaussian processes," *arXiv:1502.02843*, 2015.

[41] Y. Gal, R. T. McAllister, and C. E. Rasmussen, "Improving PILCO with bayesian neural network dynamics models," in *Data-Efficient Machine Learning workshop*, 2016.

[42] Y. Gal and Z. Ghahramani, "Dropout as a bayesian approximation: Representing model uncertainty in deep learning," in *Proc. of ICML*, 2015.