Using Parameterized Black-Box Priors to Scale Up Model-Based Policy Search for Robotics

Konstantinos Chatzilygeroudis, Jean-Baptiste Mouret

Using Parameterized Black-Box Priors to Scale Up Model-Based Policy Search for Robotics

Konstantinos Chatzilygeroudis and Jean-Baptiste Mouret*

***Abstract*— The most data-efficient algorithms for reinforcement learning in robotics are model-based policy search algorithms, which alternate between learning a dynamical model of the robot and optimizing a policy to maximize the expected return given the model and its uncertainties. Among the few proposed approaches, the recently introduced Black-DROPS algorithm exploits a black-box optimization algorithm to achieve both high data-efficiency and good computation times when several cores are used; nevertheless, like all model-based policy search approaches, Black-DROPS does not scale to high dimensional state/action spaces. In this paper, we introduce a new model learning procedure in Black-DROPS that leverages parameterized black-box priors to (1) scale up to high-dimensional systems, and (2) be robust to large inaccuracies of the prior information. We demonstrate the effectiveness of our approach with the "pendubot" swing-up task in simulation and with a physical hexapod robot (48D state space, 18D action space) that has to walk forward as fast as possible. The results show that our new algorithm is more data-efficient than previous model-based policy search algorithms (with and without priors) and that it can allow a physical 6-legged robot to learn new gaits in only 16 to 30 seconds of interaction time.**

I. Introduction

Robots have to face the real world, in which trying something might take seconds, hours, or even days [1]. Unfortunately, the current state-of-the-art learning algorithms (*e.g.*, deep learning [2]) either rely on the availability of very large data sets (*e.g.*, 1.2 millions labeled images in the ImageNet database [3]) or only make sense in simulated environments (*e.g.*, 38 days of learning for Atari games [4]). This scarcity of data calls for algorithms that are highly data-efficient, that is, that minimize the *interaction time* between the robot and the world, even if it means a considerable computation cost.

In reinforcement learning for robotics, the most data-efficient algorithms are model-based policy search algorithms [5], [6]: after each episode, the algorithm updates a model of the dynamics of the robot, then it searches for the best policy according to the model. To improve the data-efficiency, the current algorithms take the uncertainty of the model into account in order to avoid overfitting the model [7], [8]. The PILCO algorithm [7] implements these ideas, but (1) it imposes several constraints on the reward functions and policies (because it needs to compute gradients analytically), and (2) it is a slow algorithm that cannot benefit from multi-core computers (typically about an hour to complete 15 episodes on the cart-pole benchmark) [8].

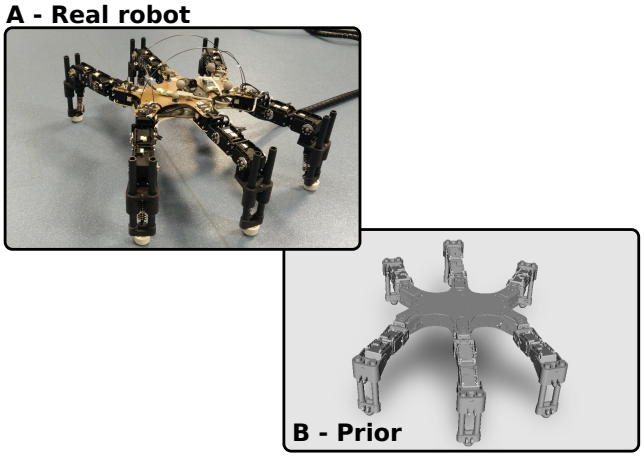

Fig. 1. **A.** The physical hexapod robot used in the experiments (48D state space and 18D action space). **B.** The simulated hexapod that is used as a prior model for our approach in the experiments.

The recently introduced Black-DROPS algorithm [8] is one of the first model-based policy search algorithms for robotics that is purely black-box and can extensively take advantage of parallel computations. Black-DROPS achieves similar data-efficiency to state-of-the-art approaches like PILCO (*e.g.*, less than 20 s of interaction time to solve the cart-pole swing-up task), while being faster on multi-core computers, easier to set up, and much less limiting (*i.e.*, it can use any policy and/or reward parameterization; it can even learn the reward model).

However, while Black-DROPS scales well with the number of processors, the main challenge of model-based policy search is scaling up to complex problems: as the algorithm models the transition function between full state/action spaces (joint positions, environment, joint velocities, *etc.*), the complexity of the model increases substantially with each new degree of freedom; unfortunately, the quantity of data required to learn a good model scales most of the time exponentially with the dimension of the state space [9]. As a consequence, the data-efficiency of model-based approaches greatly suffers from the increase of the dimensionality of the model. In practice, model-based policy search algorithms can currently be employed only with simple systems up to 10-15D state and action space combined (*e.g.*, double cart-pole or a simple manipulator).

One way of tackling the problem raised by the "curse of

*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).

dimensionality" is to use prior information about the system that is modeled; for instance, dynamic simulators of the robot can be effective priors and are often available. The ideal model-based policy search algorithm with priors for robotics should, therefore:

- scale to high dimensional and complex robots (*e.g.*, walking or soft robots);
- take advantage of multi-core architectures to speed-up computation times;
- perform the search in the full policy space (*i.e.*, the more real trials, the better expected reward);
- make as few assumptions as possible about the type of robot and the prior information (*i.e.*, require no specific structure or differentiable models);
- be able to select among several prior models or to tune the prior model.

A few algorithms leverage prior information to speed-up learning on the real system [10], [11], [12], [13], [14], [15], but none of them fulfills all of the above properties. In this paper, we propose a novel, purely black-box, flexible and data-efficient model-based policy search algorithm that combines ideas from the Black-DROPS algorithm, from simulation-based priors, and from recent model learning algorithms [16], [17]. We show that our approach is capable of learning policies in about 30 seconds to control a damaged physical hexapod robot (48D state space, 18D action space) and outperforms state-of-the-art model-based policy search algorithms without (PILCO [7], Black-DROPS [8]) and with priors (PILCO with priors [10]), as well as prior-based Bayesian optimization (IT&E [14]).

II. Background

A. Policy Search for Robotics

Model-free policy search (PS) methods have been successful in robotics as they can easily be applied in high-dimensional continuous state-action RL problems [5], [18], [19]. The PoWER algorithm [20] uses probability-weighted averaging, which has the property of following the natural gradient without computing it. The PI^2 [21] algorithm has very similar performance with PoWER, but puts no constraint on the reward function. Natural Evolution Strategies (NES) [22] and Covariance Matrix Adaptation ES (CMA-ES) [23] families of algorithms are population-based black-box optimizers that iteratively update a search distribution by calculating an estimated gradient on the distribution parameters (mean and covariance). At each generation, they sample a set of policy parameters and rank them based on their expected return. NES performs gradient ascent along the natural gradient, whereas CMA-ES updates the distribution by exploiting the technique of evolution paths.

Although, model-free policy search methods are promising, they require a few hundreds or thousands of episodes to converge to good solutions [5], [6]. The data-efficiency of such methods can be increased by learning the model (*i.e.*, transition and reward function) of the system from data and inferring the optimal policy from the model [5], [6]. For example, state-of-the-art model-free policy gradient methods (*e.g.*, TRPO [19] or DDPG [18]) require more than $500\,s$ of interaction time to solve the cart-pole swing-up task [18] whereas state-of-the-art model-based policy search algorithms (*e.g.*, PILCO or Black-DROPS) require less than $20\,s$ [8], [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 [7], [8], [6], [5].

Black-DROPS [8] and PILCO [7] are two of the most data-efficient model-based policy search algorithms for robot control. They essentially differ in how they use the uncertainty of the model and in how they optimize the policy given the model: PILCO uses moment matching and analytical gradients [7], whereas Black-DROPS uses Monte-Carlo rollouts and a black-box optimizer.

Black-DROPS adds two main benefits to PILCO: (1) any reward function or policy parameterization can be used (including non-differentiable policies like finite automata), and (2) it is a highly-parallel algorithm that takes advantages of multi-core computers. Black-DROPS achieves similar data-efficiency to PILCO and escapes local optima faster in standard control benchmarks (inverted pendulum and cart-pole swing-up) [8]. It was also able to learn from scratch a high dimensional policy (neural network with 134 parameters) in only 5-6 trials on a physical low-cost manipulator [8].

B. Accelerating Policy Search using Priors

Model-based policy search algorithms reduce the required interaction time, but for more complex or higher dimensional systems, they still require dozens or even hundreds of episodes to find a working policy; in some systems, they might also fail to find any good policy because of the inevitable model errors and biases [24].

One way to reduce the interaction time without learning models is to begin with a meaningful initial policy (coming from demonstration or simulation) and then search locally to improve it. Usually this is done by human demonstration and movement primitives [25]: a human either tele-operates or moves the robot by hand trying to achieve the task and then a model-free RL method is applied to improve the initial policy [20], [26]. However, these approaches still suffer from the data inefficiency of model-free approaches and require dozens or hundreds of episodes to find good policies.

Another way to reduce the interaction time in model-free approaches is to pre-compute archives/libraries of policies/controllers [27], [28] and then search online for the one that works best on the real system [14], [29]. The Intelligent Trial-and-Error (IT&E) algorithm [14] first uses an evolutionary algorithm called MAP-Elites [30], [31] off-line to create an archive of diverse and locally high-performing behaviors and then utilizes a modified version of Bayesian optimization (BO) [32] to quickly find a compensatory behavior. Although IT&E can allow, for instance, a damaged 6-legged robot to find a new gait in about a dozen trials (less than 2 minutes) and a robotic arm to overcome several blocked joints in a few minutes, it is not searching in the

full policy space and as such there is no guarantee that the optimal policy can be found.

Reducing the interaction time in model-based policy search can be achieved by using priors on the models [10], [11], [12], [13], [33]; *i.e.*, starting with an initial guess of the dynamics and then learning the residual model. PILCO with priors [10] and PI-REM [12] are closely related as they both use the policy search procedure of PILCO. PILCO with priors uses simulated data to create a Gaussian process prior, whereas PI-REM uses analytic equations for the prior model. The main limitation of PILCO with priors is that it implicitly requires the task to be solved in the prior model with PILCO (in order to get the speed-up shown in the original paper [10]). GP-ILQG [11] also learns the residual model like PI-REM and then uses a modified version of ILQG [34] to find a policy given the uncertainties of the model. GP-ILQG, however, requires the prior model to be differentiable.

C. Model Identification and Learning

The traditional way of exploiting analytic equations is model identification [35]. Most approaches for model identification rely on two main ingredients: (a) proper excitation of the system [36], [35], [37] and (b) parametric models. Recently, Xie et. al. [38] proposed a method that combines model identification and RL. More specifically, their approach relies on a Model Predictive Control (MPC) scheme with optimistic exploration on a parametric model that is estimated from the collected data using least-squares.

However, these approaches assume that the analytical equations can fully capture the system, which is often not the case when dealing with unforeseen effects like, for example, complex friction effects or when there exists severe model mismatch (*i.e.*, no parameters can explain the data) like, for instance, when the robot is damaged.

A few methods have been proposed to combine model identification and model learning [16], [17]. Nevertheless, these methods are based on the manipulator equation exploiting it in different ways and it is not straight-forward how they can be used with more complicated robots that involve complex collisions and contacts (*e.g.*, walking or complex soft robots).

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 $\mathbf{u} \in \mathbb{R}^U$, i.i.d. Gaussian system noise $\mathbf{w}$, and unknown transition dynamics F. We assume that we have an initial guess of the dynamics, the function $M(\mathbf{x}_t, \mathbf{u}_t)$, that may not be accurate either because we do not have a very precise model of our system (*i.e.*, what is called the *"reality-gap"* [39]) or because the robot is damaged in an unforeseen way (*e.g.*, a blocked joint or faulty motor/encoder) [14], [40].

Contrary to previous works [11], [16], [17], we assume no structure or specific properties of our initial dynamics model

Algorithm 1 Model-based policy search with priors

1: Optimize $\boldsymbol{\theta}^*$ on the prior model according to $J(\boldsymbol{\theta})$ and the initial reward function r
2: Apply policy $\pi_{\boldsymbol{\theta}^*}$ on the robot and record data
3: **repeat**
4: Learn the immediate reward function r from the gathered data — if necessary
5: Learn a model that approximates the actual underlying system's dynamics using the gathered data *and the prior model*
6: Optimize $\boldsymbol{\theta}^*$ on the model according to $J(\boldsymbol{\theta})$ and the (learned) reward function r
7: Apply policy $\pi_{\boldsymbol{\theta}^*}$ on the robot and record data
8: **until** Task is solved

M (*i.e.*, we treat it as a black-box function), other than it has some tunable parameters, $\boldsymbol{\phi}_M$, which change its behavior. Examples of these parameters can be some optimization parameters (*e.g.*, type of optimizer) of a dynamic simulator involving contacts and collisions or some internal parameters of the robot (*e.g.*, masses of the bodies). Finally, we add a non-parametric model, f (with associated hyper-parameters $\boldsymbol{\phi}_K$), to model whatever is not possible to capture with M:

$$\mathbf{x}_{t+1} = \mathbf{x}_t + M(\mathbf{x}_t, \mathbf{u}_t, \boldsymbol{\phi}_M) + f(\mathbf{x}_t, \mathbf{u}_t, \boldsymbol{\phi}_K) + \mathbf{w} \tag{2}$$

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

$$J(\boldsymbol{\theta}) = \mathbb{E}\left[\sum_{t=1}^{T} r(\mathbf{x}_t)\middle|\boldsymbol{\theta}\right] \tag{3}$$

where $r(\mathbf{x}_t)$ is the immediate reward of being in state $\mathbf{x}_t$. We assume that π is a function parameterized by $\boldsymbol{\theta} \in \mathbb{R}^{\Theta}$.

In model-based policy search with priors, we begin by optimizing the policy on the prior model (that is, *there is no prior information on the policy parameters*) and applying it on the real system to gather the initial data. Afterwards, a loop is iterated where we first learn a model using the prior model and the collected data and then optimize the policy given this newly learned model (Algo. 1). Finally, the policy is applied on the real system, more data is collected and the loop re-iterates until the task is solved.

IV. Approach

A. Gaussian processes with the simulator as the mean function

We would like to have a model $\hat{F}$ that approximates as accurately as possible the unknown dynamics F of our system given some initial guess, M. We rely on Gaussian processes (GPs) to do so as they have been successfully used in many model-based reinforcement learning approaches [7], [8], [41], [42], [5], [40], [6]. A GP is an extension of the multivariate Gaussian distribution to an infinite-dimension stochastic process for which any finite combination of dimensions will be a Gaussian distribution [43].

As inputs, we use tuples made of the state vector $\mathbf{x}_t$ and the action vector $\mathbf{u}_t$, that is, $\tilde{\mathbf{x}}_t = (\mathbf{x}_t, \mathbf{u}_t) \in \mathbb{R}^{E+U}$; 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 $\boldsymbol{\Delta}_{\mathbf{x}_t}$. Assuming $D_{1:t} = \{F(\tilde{\mathbf{x}}_1), ..., F(\tilde{\mathbf{x}}_t)\}$ is a set of observations and $M(\tilde{\mathbf{x}})$ being the simulator function (*i.e.*, our initial guess of the dynamics — tunable or not; we drop the $\boldsymbol{\phi}_M$ parameters here for brevity), we can query the GP at a new input point $\tilde{\mathbf{x}}_*$:

$$p(\hat{F}(\tilde{\mathbf{x}}_*)|D_{1:t}, \tilde{\mathbf{x}}_*) = \mathcal{N}(\mu(\tilde{\mathbf{x}}_*), \sigma^2(\tilde{\mathbf{x}}_*)) \tag{4}$$

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

$$\begin{aligned} \mu(\tilde{\mathbf{x}}_*) &= M(\tilde{\mathbf{x}}_*) + \boldsymbol{k}^T K^{-1}(D_{1:t} - M(\tilde{\mathbf{x}}_{1:t})) \\ \sigma^2(\tilde{\mathbf{x}}_*) &= k(\tilde{\mathbf{x}}_*, \tilde{\mathbf{x}}_*) - \boldsymbol{k}^T K^{-1} \boldsymbol{k} \end{aligned} \tag{5}$$

The formulation above allows us to combine observations from the simulator and the real-world smoothly. In areas where real-world data is available, the simulator's prediction will be corrected to match the real-world ones. On the contrary, in areas far from real-world data, the predictions resort to the simulator [14], [11], [40].

This model learning procedure has been used in several articles [33], [16], [42] and in particular to learn the cumulative reward model for a BO procedure highlighted in the IT&E approach [14]. GP-ILQG [11] and PI-REM [12] formulate a similar model learning procedure for optimal control (under model uncertainty) and policy search respectively. GP-ILQG additionally assumes that the prior model M is differentiable, which is not always true and might be too slow to perform via finite differences (*e.g.*, when using black-box simulators for M). PILCO with priors [10] utilizes a similar scheme but assumes that the prior model M is a GP learned from simulation data that is gathered from running PILCO on the prior system.

We use the exponential kernel with automatic relevance determination [43] ($\boldsymbol{\phi}_K$ are the kernel hyper-parameters). When searching for the best kernel hyper-parameters through Maximum Likelihood Estimation (MLE) for a GP with a non-tunable mean function M, we seek to maximize [43]:

$$p(D_{1:t}|\tilde{\mathbf{x}}_{1:t}, \boldsymbol{\phi}_K) = \frac{1}{\sqrt{(2\pi)^t |K|}} e^{-\frac{1}{2}(D_{1:t} - M(\tilde{\mathbf{x}}_{1:t}))^T K^{-1} (D_{1:t} - M(\tilde{\mathbf{x}}_{1:t}))} \tag{6}$$

The gradients of this likelihood function can be analytically computed, which makes it possible to use any gradient based optimizer (we use Rprop [44]). Since we have E independent GPs, we have E independent optimizations. We use the limbo C++11 library for GP regression [45].

B. Mean functions with tunable parameters

We would like to use a mean function $M(\tilde{\mathbf{x}}, \boldsymbol{\phi}_M)$, where each vector $\boldsymbol{\phi}_M \in \mathbb{R}^{n_M}$ corresponds to a different prior model of our system (*e.g.*, different lengths of links). Searching for the $\boldsymbol{\phi}_M$ that best matches the observations can be seen as a model identification procedure, which could be solved via minimizing the mean squared error; nevertheless, the GP framework allows us to jointly optimize for the kernel hyper-parameters and the mean parameters, which allows the modeling procedure to balance between non-parametric and parametric modeling. We can easily extend Eq. (6) to include parameterized mean functions:

$$p(D_{1:t}|\tilde{\mathbf{x}}_{1:t}, \boldsymbol{\phi}_K, \boldsymbol{\phi}_M) = \frac{1}{\sqrt{(2\pi)^t |K|}} e^{-\frac{1}{2}(D_{1:t} - M(\tilde{\mathbf{x}}_{1:t}, \boldsymbol{\phi}_M))^T K^{-1} (D_{1:t} - M(\tilde{\mathbf{x}}_{1:t}, \boldsymbol{\phi}_M))} \tag{7}$$

This time, even though we have E independent GPs (one for each output dimension), all of them need to share the same mean parameters $\boldsymbol{\phi}_M$ (contrary to the kernel parameters, which are typically different for each dimension), because the model of the robot should be consistent in all of the output dimensions. Thus, we have to jointly optimize for the mean parameters and the kernel hyper-parameters of all the GPs. Since most dynamic simulators are not differentiable (or too slow to differentiate by finite differences), we cannot resort to gradient-based optimization to optimize Eq. (7) jointly for all the GPs. A black-box optimizer like CMA-ES [23] could be employed instead, but this optimization was too slow to converge in our preliminary experiments.

To combine the benefits of both gradient-based and gradient-free optimization, we use gradient-based optimization for the kernel hyper-parameters (since we know the analytical gradients) and black-box optimization for the mean parameters. Conceptually, we would like to optimize for the mean parameters, $\boldsymbol{\phi}_M$, given the optimal kernel hyper-parameters for each of them. Since we do not know them before-hand, we use two nested optimization loops: (a) an outer loop where a gradient-free local optimizer searches for the best $\boldsymbol{\phi}_M$ parameters (we use a variant of the Subplex algorithm [46] provided by NLOpt [47] for continuous spaces and exhaustive search for discrete ones), and (b) an inner optimization loop where given a mean parameter vector $\boldsymbol{\phi}_M$, a gradient-based optimizer searches for the best kernel hyper-parameters (each GP is independently optimized since $\boldsymbol{\phi}_M$ is fixed in the inner loop) and returns a score that corresponds to $\boldsymbol{\phi}_M$ for the optimal $\boldsymbol{\phi}_K$ (Algo. 2).

One natural way of combining the likelihoods of the independent GPs to form the objective function of the outer loop is to take the product, which would be equivalent to taking the joint probability of the likelihoods of the independent GPs (since the likelihood is a probability density function). However, we observed that taking the sum or the harmonic mean of the likelihoods instead yielded more robust results. This comes from the fact that the product can be dominated by a few terms only and thus if some parameters explain one output dimension perfectly and all the others not as well it would still be chosen. In addition, in practice we observed that taking the sum of the likelihoods proved to be numerically more stable than the harmonic mean.

Algorithm 2 GP-MI Learning process

procedure GP-MI($D_{1:t}$)
  Optimize $\boldsymbol{\phi}_M^*$ according to EVALUATEMODEL($\boldsymbol{\phi}_M$, $D_{1:t}$) using a gradient-free local optimizer
  return $\boldsymbol{\phi}_M^*$
procedure EVALUATEMODEL($\boldsymbol{\phi}_M$, $D_{1:t}$)
  Initialize E GPs $f_1, \ldots, f_E$ as $f_i(\tilde{\mathbf{x}}) \sim \mathcal{N}(M_i(\tilde{\mathbf{x}}, \boldsymbol{\phi}_M), k_i(\tilde{\mathbf{x}}, \tilde{\mathbf{x}}))$ ▷ M^i queries M and returns the i-th element of the return vector, k_i is the kernel function of the i-th GP
  for i from 1 to E **do** ▷ This can also be done in parallel
    Optimize the kernel hyper-parameters, $\boldsymbol{\phi}_K^i$, of f_i given $D_{1:t}^i$ assuming $\boldsymbol{\phi}_M$ fixed ▷ $D_{1:t}^i$ is the i-th column of $D_{1:t}$
    $\text{lik}_i = p(D_{1:t}^i | \tilde{\mathbf{x}}_{1:t}, \boldsymbol{\phi}_K^i, \boldsymbol{\phi}_M)$ ▷ Eq. (7)
  return $\sum_{i=1}^{E} \text{lik}_i$ ▷ Sum of the independent likelihoods

Our model learning approach, which we call *GP-MI* (Gaussian Process Model Identification), that combines non-parametric model learning and parametric model identification is related to the approach in [16], but there are some key differences between them. Firstly, the model learning procedure in [16] depends on the manipulator equation and cannot easily be used with robots that do not directly comply to the equation (one example would be the hexapod robot in our experiments or a soft robot with complex dynamics), whereas GP-MI imposes no structure on the prior model, other than providing some tunable parameters (continuous or discrete). Furthermore, the approach in [16] is tied to inverse dynamics models and cannot be used with forward models in the general case (necessary for long-term forward predictions); on the contrary, GP-MI can be used with inverse or forward dynamics models and in general with any black-box tunable prior model.

C. Policy Search with the Black-DROPS algorithm

We use the Black-DROPS [8] algorithm for policy search because it allows us to use the type of priors discussed in Section IV-B and to leverage specific policy parameterizations that are suitable for different cases (*e.g.*, we use a neural network policy for the pendubot task and an open-loop periodic policy for the hexapod). We assume *no prior information on the policy parameters* and we begin by optimizing the policy on the prior model. Moreover, we took advantage of multi-core architectures to speed-up our experiments. Contrary to Black-DROPS, PILCO [7] cannot take advantage of multiple cores[1] and the need for deriving all the gradients for a different policy/reward makes it difficult (or even impossible) to try new ideas/policies.

To take the uncertainties of the model into account, the core idea of Black-DROPS is to avoid to compute the expected reward of policy parameters, which is what most approaches do and is usually either computationally expensive [48] or requires some approximation to be made [7]. Instead it treats each Monte-Carlo rollout as a noisy measurement of a function $G(\boldsymbol{\theta})$ that is the actual function $J(\boldsymbol{\theta})$ perturbed by a noise $N(\boldsymbol{\theta})$ and tries to maximize its expectation:

$$\begin{aligned}\mathbb{E}\Big[G(\boldsymbol{\theta})\Big] &= \mathbb{E}\Big[J(\boldsymbol{\theta}) + N(\boldsymbol{\theta})\Big] = \mathbb{E}\Big[J(\boldsymbol{\theta})\Big] + \mathbb{E}\Big[N(\boldsymbol{\theta})\Big] \\ &= J(\boldsymbol{\theta}) + \mathbb{E}\Big[N(\boldsymbol{\theta})\Big] (\text{since } \mathbb{E}\Big[\mathbb{E}[x]\Big] = \mathbb{E}[x]) \quad (8)\end{aligned}$$

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 $J(\boldsymbol{\theta})$ (see Eq. (3)). The second main idea of Black-DROPS, is to use a population-based black-box optimizer that (1) can optimize noisy functions and (2) can take advantage of multi-core computers. Here we use BIPOP-CMAES [23], [8].

PI-REM [12] is close to our approach as it leverages priors to learn the residual model and then performs policy search on the model. However, PI-REM assumes that the prior information is fixed and cannot be tuned, whereas our approach has the additional flexibility of being able to change the behavior of the prior. In addition, PI-REM utilizes the policy search procedure of PILCO that can be limiting in many cases as already discussed. Nevertheless, as Black-DROPS and PILCO have been shown to perform similarly when PILCO's limitations are not present [8], we include in our experiments a variant of our approach that resembles PI-REM (Black-DROPS with priors).

V. EXPERIMENTAL RESULTS

A. Pendubot swing-up task

We first evaluate our approach in simulation with the pendubot swing-up task. The pendubot is a two-link under-actuated robotic arm (with lenghts l_1, l_2 and masses m_1, m_2) and was introduced by [49] (Fig. 2). The inner joint (attached to the ground) exerts a torque $|u| \leq 3.5$, but the outer joint cannot (both of the joints are subject to some friction with coefficients b_1, b_2). The system has four continuous state variables: two joint angles and two joint angular velocities. The angles of the joints, θ_1 and θ_2, are measured anti-clockwise from the upright position. The pendubot starts hanging down and the goal is to find a policy such that the pendubot swings up and then balances in the upright position. Each episode lasts 2.5 s and the control rate is 20 Hz. We use a distance based reward function as in [8].

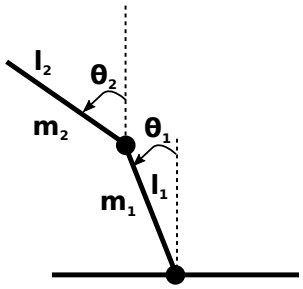

Fig. 2. The pendubot system

We chose this task because it is a fairly difficult problem and forces slower convergence on model-based techniques without priors, but not too hard (*i.e.*, it can be solved without priors in reasonable interaction time); a fact that allowed us to make a rather extensive evaluation with meaningful

[1]For reference, each run of PILCO with priors (26 episodes + model learning) in the pendubot task took around 70 hours on a modern computer with 16 cores, whereas each run of Black-DROPS with priors and Black-DROPS with GP-MI took around 15 hours and 24 hours respectively.

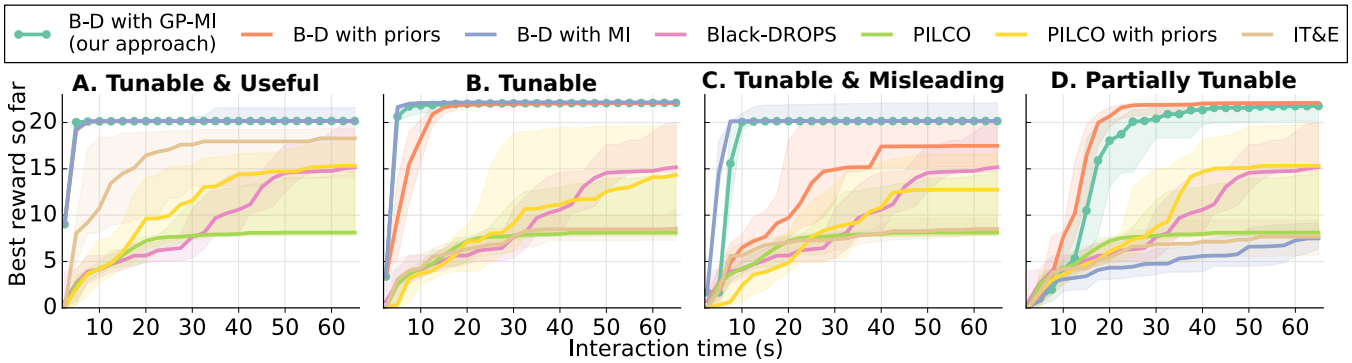

Fig. 3. Results for the pendubot task (30 replicates of each scenario). The lines are median values and the shaded regions the 25^{th} and 75^{th} percentiles. See Table I for the description of the priors. Black-DROPS with GP-MI always solves the task and achieves high rewards at least as fast as all the other approaches in all the cases that we considered. Black-DROPS with MI achieves good rewards whenever the parameters it can tune are the ones that are wrong (**A,B,C**) and bad rewards otherwise (**D**). Black-DROPS with priors performs very well whenever the prior model is not too far away from the real one (**A,B**) and not so well whenever the prior is misleading (**C**). Black-DROPS with priors and MI have very similar performance in **A** and as such are not easily distinguishable. IT&E and PILCO with priors are not able to reliably solve the task across different prior models.

Variable	*Actual*	Tunable & Useful Prior	Tunable Prior	Tunable & Misleading Prior	Partially Tunable Prior
m_1	*0.5*	**0.65** (30% incr.)	0.5	0.5	**0.65** (30% incr.)
m_2	*0.5*	0.5	**0.75** (50% incr.)	0.5	**0.35** (30% decr.)
l_1	*0.5*	0.5	0.5	0.5	0.5
l_2	*0.5*	**0.4** (20% decr.)	0.5	**0.25** (50% decr.)	0.5
b_1 non-tunable	*0.1*	0.1	0.1	0.1	**0.** (100% decr.)
b_2 non-tunable	*0.1*	0.1	0.1	0.1	**0.** (100% decr.)

TABLE I

ACTUAL SYSTEM AND PRIORS FOR THE PENDUBOT TASK.

comparisons (4 different prior models, 7 different algorithms, 30 replicates of each combination). We assume that we have 4 priors available; we tried to capture easy and difficult cases and cases where all the wrong parameters can be tuned or not (see Table I): **Tunable & Useful:** a fully tunable prior that is very close to the actual one; **Tunable:** a fully tunable prior that is not very close to the actual; **Tunable & Misleading:** a prior that can be fully tuned, but is very far from the actual; **Partially tunable:** a prior that cannot be fully tuned, but not very far from the actual.

We compare 7 algorithms: **1.** Black-DROPS [8]; **2.** Black-DROPS with priors, which is close to PI-REM [12] and GP-ILQG [11][2]; **3.** Black-DROPS with GP-MI (*our approach*); **4.** Black-DROPS with MI (Black-DROPS where model learning is replaced by model identification — via mean squared error); **5.** PILCO [7]; **6.** PILCO with priors [10]; **7.** IT&E [14].

For Black-DROPS with GP-MI and the MI variant, we additionally assume that the parameters m_1, m_2, l_1 and l_2 can be tuned, but the parameters b_1 and b_2 are fixed and cannot be changed. Since the adaptation part of IT&E is a deterministic algorithm (given the same prior) and our system has no uncertainty, for each prior we generated 30 archives with different random seeds and then ran the adaptation part of IT&E once for each archive. We used 3 equally spread in time end-effector positions as the behavior descriptor for the archive generation with MAP-Elites. For all the Black-DROPS variants and for IT&E we used a neural network policy with one hidden layer (10 hidden neurons) and the hyperbolic tangent as the activation function.

Similarly to IT&E, since PILCO with priors is a deterministic algorithm given the same prior, for each prior we ran PILCO 30 times with different random seeds on the prior model (for 40 episodes in order for PILCO to converge to a good policy and model) and then ran PILCO with priors on the actual system once for each different model. We used priors both in the policy and the dynamics model when learning in the actual system (as advised in [10]). We also used a GP policy with 200 pseudo-observations [7][3].

Black-DROPS with GP-MI always solves the task and achieves high rewards at least as fast as all the other approaches in the cases that we considered (Fig. 3). Black-DROPS with MI performs very well when the parameters it can tune are the ones that are wrong (Fig. 3**A,B,C**), and badly otherwise (Fig. 3**D** — *i.e.*, no parameters of the prior model can explain the data). Black-DROPS with priors performs very well whenever the prior model is not far away from the real one (Fig. 3**A,B**) and not so well whenever the prior is misleading (Fig. 3**C**). Both Black-DROPS and PILCO cannot solve the task in less than $65\ s$ of interaction time, but Black-DROPS shows a faster learning curve (Fig. 3).

Interestingly, PILCO with priors is not able to always achieve better results than Black-DROPS and is always worse than Black-DROPS with priors. This can be explained by the fact that PILCO without priors learns slower than Black-DROPS and is a more local search algorithm and as such needs more interaction time to achieve good results. On the contrary, Black-DROPS uses a modified version of CMA-ES that can more easily escape local optima [8]. Moreover, the initial prior model for PILCO with priors is an approximated model, whereas Black-DROPS with priors uses the actual

[2]The algorithm in this specific form is first formulated in this paper (*i.e.*, the Black-DROPS policy search procedure with a prior model), but, as discussed above, it is close in spirit with GP-ILQG [11] and PI-REM [12]. Therefore, we assume that the performance of Black-DROPS with priors is representative of what could be achieved with PI-REM and GP-ILQG, although Black-DROPS with priors should be more effective because it performs a more global search [8].

[3]These are the parameters that come with the original code of PILCO. We used the code from: https://bitbucket.org/markjcutler/gaussian-process.

prior model to begin with. Lastly, the GP policy, that PILCO is mainly used with[4], creates really high dimensional policy spaces compared to the simple neural network policy that Black-DROPS is using (*i.e.*, 1400 vs 81 parameters) and as such causes the policy search to converge slower.

IT&E is not able to reliably solve the task and achieve high rewards. This is because IT&E assumes that (a) the system is redundant enough so that the task can be solved in many different ways and (b) there is a policy/controller in the pre-computed archive that can solve the task (*i.e.*, IT&E cannot search outside of this archive) [14]. Obviously, these assumptions are violated in the pendubot scenario: (a) the system is underactuated and thus does not have the required redundancy, and (b) the system is inherently unstable and as such precise policy parameters are needed (it is highly unlikely that one of them exists in the pre-computed archive).

B. Physical hexapod locomotion

We also evaluate our approach on the hexapod locomotion task as introduced in the IT&E paper [14] with a physical robot (Fig. 1**A**). This scenario is where IT&E excels and achieves remarkable recovery capabilities [14]. We assume that a simulator of the intact robot is available (Fig. 1**B**)[5]; for GP-MI we also assume that we can alter this simulator by removing 1 leg of the hexapod (*i.e.*, there are 7 discrete different parameterizations). This simulator is not accurate as we assume perfect velocity actuators and infinite torque. Each leg has 3 DOF leading to a total of 18 DOF. The state of the robot consists of 18 joint angles, 18 joint velocities, a 6D Center Of Mass (COM) pose (position and orientation) and 6D COM velocities. The policy is an open-loop controller with 36 parameters that outputs 18D joint angles every 0.1 s and is similar to the one used in [14]. Each episode lasts 4 s and the robot is tracked with a motion capture system.

The task is to find a policy to walk forward as fast as possible. Due to the complexity of the problem[6], we only compare 2 algorithms (IT&E and our approach) on 2 different conditions: (a) crossing the reality-gap problem; in this case our approach cannot mostly rely on the identification part and the importance of the GP modeling will be highlighted, and (b) one rear leg is removed; the back leg removals are especially difficult as most effective gaits of the intact robot rely on them.

The results show that Black-DROPS with GP-MI is able to learn highly effective walking policies on the physical hexapod robot (Fig. 4). In particular, using the dynamics simulator as prior information Black-DROPS with GP-MI is able to achieve better (and with less variance) walking speeds than IT&E [14] on the intact physical hexapod (Fig. 4**A**). Moreover, in the rear-leg removal damage case Black-DROPS with GP-MI allows the damaged robot to walk effectively after only 16 to 30 seconds of interaction time and finds higher-performing policies than IT&E ($0.21m/s$ vs $0.15m/s$ in the 8^{th} episode) (Fig. 4**B**).

[4]So far, PILCO can only be used with linear or GP policy types [7].

[5]We use the DART simulator [50].

[6]PILCO and Black-DROPS could not find any solution in preliminary simulation experiments even after several minutes of interaction time and Black-DROPS with priors was worse than Black-DROPS with GP-MI.

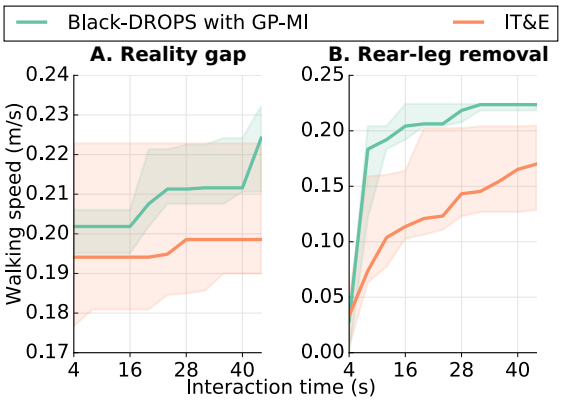

Fig. 4. Results for the physical hexapod locomotion task (5 replicates of each scenario). The lines are median values and the shaded regions the 25^{th} and 75^{th} percentiles. **A.** Improving a policy for the intact robot (crossing the reality gap): Black-DROPS with GP-MI finds a highly-effective policy (about $0.22m/s$) in less than 30 seconds of interaction time, whereas IT&E is not able to substantially improve the initial policy. **B.** Rear-leg removal damage case: Black-DROPS with GP-MI allows the damaged robot to walk effectively after only 16 to 30 seconds of interaction time and finds higher-performing policies than IT&E ($0.21m/s$ vs $0.15m/s$ in the 8^{th} episode).

Overall, Black-DROPS with GP-MI was able to successfully learn working policies even though the dimensionality of the state and the action space of the hexapod robot is 48D and 18D respectively. In addition, in the rear leg damage case, Black-DROPS always tried safer policies than IT&E that too often executed policies that would cause the robot to fall over. A video of our algorithm running on the damaged hexapod is available at the supplementary video (also at `https://youtu.be/HFkZkhGGzTo`).

VI. Conclusion and Discussion

Black-DROPS with GP-MI is one of the first model-based policy search algorithms that can efficiently learn with high-dimensional physical robots. It was able to learn walking policies for a physical hexapod (48D state and 18D action space) in less than 1 minute of interaction time, *without any prior on the policy parameters* (that is, it learns a policy from scratch). The black-box nature of our approach along with the extra flexibility of tuning the black-box prior model opens a new direction of experimentation as changing priors, robots or tasks requires minimum effort.

The way we compute the long-term predictions (*i.e.*, by chaining model predictions) requires that predicted states (the output of the GPs) are fed back to the prior simulator. This can cause the simulator to crash because there is no guarantee that the predicted state, that possibly makes sense in the real world, will make sense in the prior model; especially when the two models (prior and real) differ a lot and when there are obstacles and collisions involved. This also holds for most other prior-based methods [11], [12], [10], but it is not easily seen in simple systems. On the contrary, we observed this phenomenon a few times in our hexapod experiments. Using the prior simulator just as a reference and not mixing prior and real data is a direction of future work.

Finally, Black-DROPS with GP-MI brings closer trial-and-error and diagnosis-based approaches for robot damage recovery. It successfully combines (a) diagnosis [51] (*i.e.*, identifying the likeliest robot model from data), (b) prior knowledge of possible damages/different conditions that a robot may face and (c) trial-and-error learning.

APPENDIX

Code for replicating the experiments: `https://github.com/resibots/blackdrops`.

ACKNOWLEDGMENTS

The authors would like to thank Dorian Goepp, Rituraj Kaushik, Jonathan Spitz, and Vassilis Vassiliades for their feedback.

REFERENCES

[1] C. Atkeson *et al.*, "No falls, no resets: Reliable humanoid behavior in the DARPA robotics challenge," in *Proc. of Humanoids*, 2015.

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

[3] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *CVPR*, 2009.

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

[5] 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.

[6] A. S. Polydoros and L. Nalpantidis, "Survey of model-based reinforcement learning: Applications on robotics," *Journal of Intelligent & Robotic Systems*, pp. 1–21, 2017.

[7] 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.

[8] K. Chatzilygeroudis, R. Rama, R. Kaushik, D. Goepp, V. Vassiliades, and J.-B. Mouret, "Black-Box Data-efficient Policy Search for Robotics," in *Proc. of IROS*, 2017.

[9] E. Keogh and A. Mueen, "Curse of dimensionality," in *Encyclopedia of Machine Learning*. Springer, 2011, pp. 257–258.

[10] M. Cutler and J. P. How, "Efficient reinforcement learning for robots using informative simulated priors," in *Proc. of ICRA*, 2015.

[11] G. Lee, S. S. Srinivasa, and M. T. Mason, "GP-ILQG: Data-driven Robust Optimal Control for Uncertain Nonlinear Dynamical Systems," *arXiv preprint arXiv:1705.05344*, 2017.

[12] M. Saveriano, Y. Yin, P. Falco, and D. Lee, "Data-Efficient Control Policy Search using Residual Dynamics Learning," in *Proc. of IROS*, 2017.

[13] B. Bischoff, D. Nguyen-Tuong, H. van Hoof, A. McHutchon, C. E. Rasmussen, A. Knoll, J. Peters, and M. P. Deisenroth, "Policy search for learning robot control using sparse data," in *Proc. of ICRA*, 2014.

[14] 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.

[15] A. Marco, F. Berkenkamp, P. Hennig, A. P. Schoellig, A. Krause, S. Schaal, and S. Trimpe, "Virtual vs. Real: Trading Off Simulations and Physical Experiments in Reinforcement Learning with Bayesian Optimization," in *Proc. of ICRA*, 2017.

[16] D. Nguyen-Tuong and J. Peters, "Using model knowledge for learning inverse dynamics," in *Proc. of ICRA*, 2010.

[17] R. Camoriano, S. Traversaro, L. Rosasco, G. Metta, and F. Nori, "Incremental semiparametric inverse dynamics learning," in *Proc. of ICRA*, 2016.

[18] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 2015.

[19] J. Schulman, S. Levine, P. Moritz, M. I. Jordan, and P. Abbeel, "Trust region policy optimization," in *Proc. of ICML*, 2015.

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

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

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

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

[24] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 1998.

[25] J. Kober and J. Peters, "Imitation and reinforcement learning," *IEEE Robotics & Automation Magazine*, vol. 17, no. 2, pp. 55–62, 2010.

[26] 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.

[27] A. Cully and J.-B. Mouret, "Behavioral repertoire learning in robotics," in *GECCO*. ACM, 2013.

[28] A. Majumdar and R. Tedrake, "Funnel libraries for real-time robust feedback motion planning," *IJRR*, vol. 36, no. 8, pp. 947–982, 2017.

[29] R. Antonova, A. Rai, and C. G. Atkeson, "Sample efficient optimization for learning controllers for bipedal locomotion," in *Proc. of Humanoids*, 2016.

[30] J.-B. Mouret and J. Clune, "Illuminating search spaces by mapping elites," *arxiv:1504.04909*, 2015.

[31] V. Vassiliades, K. Chatzilygeroudis, and J.-B. Mouret, "Using centroidal voronoi tessellations to scale up the multi-dimensional archive of phenotypic elites algorithm," *IEEE Trans. on Evolutionary Computation*, 2017.

[32] 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.

[33] 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.

[34] E. Todorov and W. Li, "A generalized iterative LQG method for locally-optimal feedback control of constrained nonlinear stochastic systems," in *Proc. of ACC*, 2005.

[35] J. Hollerbach, W. Khalil, and M. Gautier, "Model identification," in *Springer Handbook of Robotics*. Springer, 2016, pp. 113–138.

[36] M. Gautier and W. Khalil, "Exciting trajectories for the identification of base inertial parameters of robots," *IJRR*, vol. 11, no. 4, pp. 362–375, 1992.

[37] F. Aghili, J. M. Hollerbach, and M. Buehler, "A modular and high-precision motion control system with an integrated motor," *IEEE/ASME Transactions on Mechatronics*, vol. 12, no. 3, pp. 317–329, 2007.

[38] C. Xie, S. Patil, T. Moldovan, S. Levine, and P. Abbeel, "Model-based reinforcement learning with parametrized physical models and optimism-driven exploration," in *Proc. of ICRA*, 2016.

[39] J.-B. Mouret and K. Chatzilygeroudis, "20 Years of Reality Gap: a few Thoughts about Simulators in Evolutionary Robotics," in *Workshop" Simulation in Evolutionary Robotics", GECCO*, 2017.

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

[41] Y. Engel, S. Mannor, and R. Meir, "Reinforcement learning with Gaussian processes," in *Proc. of ICML*. ACM, 2005.

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

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

[44] M. Blum and M. A. Riedmiller, "Optimization of Gaussian process hyperparameters using Rprop," in *Proc. of ESANN*, 2013.

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

[46] T. H. Rowan, "Functional stability analysis of numerical algorithms," 1990.

[47] G. Johnson Steven, "The NLopt nonlinear-optimization package."

[48] A. Kupcsik, M. P. Deisenroth, J. Peters, A. P. Loh, P. Vadakkepat, and G. Neumann, "Model-based contextual policy search for data-efficient generalization of robot skills," *Artificial Intelligence*, 2014.

[49] M. W. Spong and D. J. Block, "The pendubot: A mechatronic system for control research and education," in *Proc. of Decision and Control*, 1995.

[50] J. Lee *et al.*, "DART: Dynamic Animation and Robotics Toolkit," *The Journal of Open Source Software*, vol. 3, no. 22, 2018.

[51] R. Isermann, *Fault-diagnosis systems: an introduction from fault detection to fault tolerance*. Springer Science & Business Media, 2006.