A survey on policy search algorithms for learning robot controllers in a handful of trials

Konstantinos Chatzilygeroudis, Vassilis Vassiliades, Freek Stulp, Sylvain Calinon, Jean-Baptiste Mouret

A survey on policy search algorithms for learning robot controllers in a handful of trials

Konstantinos Chatzilygeroudis[†], Vassilis Vassiliades[†*], Freek Stulp[‡], Sylvain Calinon[◇] and Jean-Baptiste Mouret[†]

***Abstract*—Most policy search algorithms require thousands of training episodes to find an effective policy, which is often infeasible with a physical robot. This survey article focuses on the extreme other end of the spectrum: how can a robot adapt with only a handful of trials (a dozen) and a few minutes? By analogy with the word "big-data", we refer to this challenge as "micro-data reinforcement learning". We show that a first strategy is to leverage prior knowledge on the policy structure (e.g., dynamic movement primitives), on the policy parameters (e.g., demonstrations), or on the dynamics (e.g., simulators). A second strategy is to create data-driven surrogate models of the expected reward (e.g., Bayesian optimization) or the dynamical model (e.g., model-based policy search), so that the policy optimizer queries the model instead of the real system. Overall, all successful micro-data algorithms combine these two strategies by varying the kind of model and prior knowledge. The current scientific challenges essentially revolve around scaling up to complex robots, designing generic priors, and optimizing the computing time.**

***Index Terms*—Learning and Adaptive Systems, Autonomous Agents, Robot Learning, Micro-Data Policy Search**

I. Introduction

Reinforcement learning (RL) [1] is a generic framework that allows robots to learn and adapt by trial-and-error. There is currently a renewed interest in RL owing to recent advances in deep learning [2]. For example, RL-based agents can now learn to play many of the Atari 2600 games directly from pixels [3], [4], that is, without explicit feature engineering, and beat the world's best players at Go and chess with minimal human knowledge [5]. Unfortunately, these impressive successes are difficult to transfer to robotics because the algorithms behind them are highly data-intensive: 4.8 million games were required to learn to play Go from scratch [5], 38 days of play (real time) for Atari 2600 games [3], and, for example, about 100 hours of simulation time (much more for real time) for a 9-DOF mannequin that learns to walk [6].

By contrast, robots have to face the real world, which cannot be accelerated by GPUs nor parallelized on large clusters. And the real world will not become faster in a few years, contrary to computers so far (Moore's law). In concrete terms, this means that most of the experiments that are successful in simulation cannot be replicated in the real world because they would take too much time to be technically feasible. As an example, Levine et al. [7] recently proposed a large-scale algorithm for learning hand-eye coordination for robotic grasping using deep learning. The algorithm required approximately 800000 grasps, which were collected within a period of 2 months using 6-14 robotic manipulators running in parallel. Although the results are promising, they were only possible because they could afford having that many manipulators and because manipulators are easy to automate: it is hard to imagine doing the same with a farm of humanoids.

What is more, online adaptation is much more useful when it is fast than when it requires hours — or worse, days — of trial-and-error. For instance, if a robot is stranded in a nuclear plant and has to discover a new way to use its arm to open a door; or if a walking robot encounters a new kind of terrain for which it is required to alter its gait; or if a humanoid robot falls, damages its knee, and needs to learn how to limp: in most cases, adaptation has to occur in a few minutes or within a dozen trials to be of any use.

By analogy with the word "big-data", we refer to the challenge of learning by trial-and-error in a handful of trials as "micro-data reinforcement learning" [8]. This concept is close to "data-efficient reinforcement learning" [9], but we think it captures a slightly different meaning. The main difference is that efficiency is a ratio between a cost and benefit, that is, data-efficiency is a ratio between a quantity of data and, for instance, the complexity of the task. In addition, efficiency is a relative term: a process is more efficient than another; it is not simply "efficient". In that sense, many deep learning algorithms are data-efficient because they require fewer trials than the previous generation, regardless of the fact that they might need millions of time-steps. By contrast, we propose the terminology "micro-data learning" to represent an absolute value, not a relative one: how can a robot learn in a few minutes of interaction? or how can a robot learn in less than 20 trials[1]? Importantly, a micro-data algorithm might reduce the number of trials by incorporating appropriate prior knowledge; this does not necessarily make it more "data-efficient" than another algorithm that would use more trials but less prior knowledge: it simply makes them different because the two algorithms solve a different challenge.

[†]Inria, CNRS, Université de Lorraine, LORIA, F-54000 Nancy, France
[*]Research Centre on Interactive Media, Smart Systems and Emerging Technologies, Dimarcheio Lefkosias, Plateia Eleftherias, 1500, Nicosia, Cyprus
[‡]German Aerospace Center (DLR), Institute of Robotics and Mechatronics, Wessling, Germany
[◇]Idiap Research Institute, Rue Marconi 19, 1920 Martigny, Switzerland

[1]It is challenging to put a precise limit for "micro-data learning" as each domain has different experimental constraints, this is why we will refer in this article to "a few minutes" or a "a few trials". The commonly used word "big-data" has a similar "fuzzy" limit that depends on the exact domain.

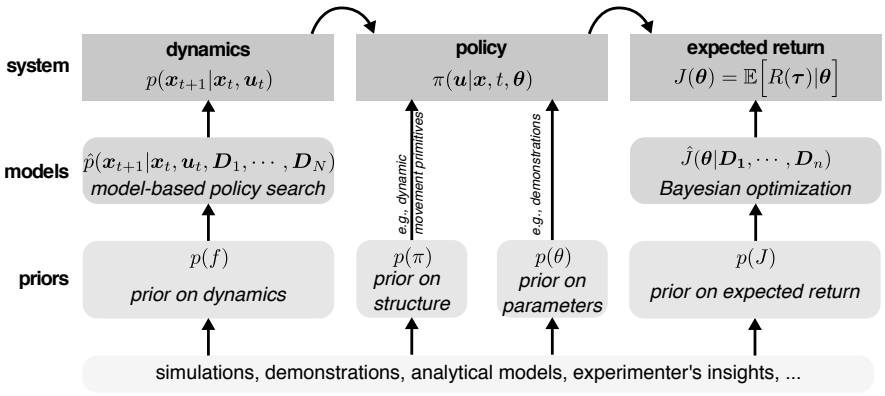

Fig. 1: Overview of possible strategies for Micro-Data Policy Search (MDPS). The first strategy (bottom) is to leverage prior knowledge on the dynamics, on the policy parameters, on the structure of the policy, or on the expected return. A second strategy is to learn surrogate models of the dynamics or of the expected return.

Among the different approaches for RL, most of the recent work in robotics focuses on *Policy Search* (PS), that is, on viewing the RL problem as the optimization of the parameters of a given policy [10] (see the problem formulation, Section II). A few PS algorithms are explicitly focused on requiring very little *interaction time* with the robot, which often implies that they authorize themselves to substantially increase the *computing time* and the amount of *prior knowledge*. The purpose of this paper is to survey such existing micro-data policy search techniques that have been successfully used for robot control [2], and to identify the challenges in this emerging field. In particular, we focus on policy search approaches that have the *explicit goal* of reducing the interaction time between the robot and the environment to a few seconds or minutes [3].

Most published algorithms for micro-data policy search implement and sometimes combine two main strategies (Fig. 1): leveraging prior knowledge (Sections III, IV-B, and V-B) and building surrogate models (Sections IV and V).

Using prior knowledge requires balancing carefully between what can be realistically known before learning and what is left to be learnt. For instance, some experiments assume that demonstrations can be provided, but that they are imperfect [13], [14]; some others assume that a damaged robot knows its model in its intact form, but not the damaged model [15]–[17]. This knowledge can be introduced at different places, typically in the structure of the policy (e.g., dynamic movement primitives [18], Section III), in the reward function (e.g., reward shaping, Section IV-B), or in the dynamical model [17], [19] (Section V-B).

The second strategy is to create models from the data gathered during learning and utilize them to make better decisions about what to try next on the robot. We can further categorize these methods into (a) algorithms that learn a surrogate model of the expected return (i.e., long-term reward) from a starting state [20], [21] (Section IV); and (b) algorithms that learn models of the transition dynamics and/or the immediate reward function (e.g., learning a controller for inverted helicopter flight by first learning a model of the helicopter's dynamics [13], Section V). The two strategies — priors and surrogates — are often combined (Fig. 2); for example, most works with a surrogate model impose a policy structure and some of them use prior information to shape the initial surrogate function, before acquiring any data.

This article surveys the literature along these three axes: priors on policy structure and parameters (Section III), models of expected return (Section IV), and models of dynamics (Section V). Section VI lists the few noteworthy approaches for micro-data policy search that do not fit well into the previous sections. Finally, Section VII sketches the challenges of the field and Section VIII proposes a few "precepts" and recommendations to guide future work in this field.

II. Problem formulation

We model the robots as discrete-time dynamical systems that can be described by transition probabilities of the form:

$$p(\boldsymbol{x}_{t+1}|\boldsymbol{x}_t, \boldsymbol{u}_t) \tag{1}$$

where the robot is at state $\boldsymbol{x}_t \in \mathbb{R}^E$ at time t, takes control input $\boldsymbol{u}_t \in \mathbb{R}^F$ and ends up at state $\boldsymbol{x}_{t+1}$ at time $t+1$.

If we assume deterministic dynamics and Gaussian system noise, this equation is often written as:

$$\boldsymbol{x}_{t+1} = f(\boldsymbol{x}_t, \boldsymbol{u}_t) + \boldsymbol{w}. \tag{2}$$

[2]Planning-based and model-predictive control [11] methods do not search for policy parameters, this is why they do not fit into the scope of this paper. Although trajectory-based policies and planning-based methods share the same goal, they usually search in a different space: planning algorithms search in the state-action space (e.g., joint positions/velocities), whereas policy methods will search for the optimal parameters of the policy, which can encode a subspace of the possible trajectories.

[3]The scarcity of data in robotics makes it necessary to follow specific strategies when designing learning algorithms. The authors of the present survey organized a very successful workshop on this exact topic at IROS 2017 (Micro-Data: the new frontier of robot learning?) and we think it is the right time to summarize the recent efforts in this direction: while there have been survey articles on policy search in the past (in particular [10], [12]), there have been many exciting developments in the last years (e.g., 50% of the papers cited in our survey have been published between 2013 and 2018). Moreover, our survey focuses on policy search algorithms that have the explicit goal of minimizing the interaction time as much as possible (and not RL or PS algorithms in general), whereas previous surveys had a broader region of interest. Consequently, we can be more thorough in our review and explain the algorithms in more detail.

Here, $\boldsymbol{w}$ is i.i.d. Gaussian system noise, and f is a function that describes the unknown transition dynamics.

We assume that the system is controlled through a parameterized *policy* $\pi(\boldsymbol{u}|\boldsymbol{x}, t, \boldsymbol{\theta})$ that is followed for T steps ($\boldsymbol{\theta}$ are the parameters of the policy). Throughout the paper we adopt the episode-based, fixed time-horizon formulations for clarity and pedagogical reasons, but also because most of the micro-data policy search approaches use this formulation.

In the general case, $\pi(\boldsymbol{u}|\boldsymbol{x}, t, \boldsymbol{\theta})$ outputs a distribution (e.g., a Gaussian) that is sampled in order to get the action to apply; i.e., we have *stochastic policies*. Most algorithms utilize policies that are not time-dependent (i.e., they drop t), but we include it here for completeness. Several algorithms use *deterministic policies*; a deterministic policy means that $\pi(\boldsymbol{u}|\boldsymbol{x}, t, \boldsymbol{\theta}) \Rightarrow \boldsymbol{u} = \pi(\boldsymbol{x}, t|\boldsymbol{\theta})$.

When following a particular policy for T time-steps from an initial state distribution $p(\boldsymbol{x}_0)$, the system's states and actions jointly form *trajectories* $\boldsymbol{\tau} = (\boldsymbol{x}_0, \boldsymbol{u}_0, \boldsymbol{x}_1, \boldsymbol{u}_1, \dots, \boldsymbol{x}_T)$, which are often also called *rollouts* or *paths*. We assume that a scalar performance system exists, $R(\boldsymbol{\tau})$, that evaluates the performance of the system given a trajectory $\boldsymbol{\tau}$. This *long-term reward* (or *return*) is defined as the sum of the immediate rewards along the trajectory $\boldsymbol{\tau}$:

$$R(\boldsymbol{\tau}) = \sum_{t=0}^{T-1} r_{t+1} = \sum_{t=0}^{T-1} r(\boldsymbol{x}_t, \boldsymbol{u}_t, \boldsymbol{x}_{t+1}) \tag{3}$$

where $r_{t+1} = r(\boldsymbol{x}_t, \boldsymbol{u}_t, \boldsymbol{x}_{t+1}) \in \mathbb{R}$ is the *immediate reward* of being in state $\boldsymbol{x}_t$ at time t, taking the action $\boldsymbol{u}_t$ and reaching the state $\boldsymbol{x}_{t+1}$ at time $t+1$. We define the *expected return* $J(\boldsymbol{\theta})$ as a function of the policy parameters:

$$\begin{aligned} J(\boldsymbol{\theta}) &= \mathbb{E}\Big[R(\boldsymbol{\tau})|\boldsymbol{\theta}\Big] \\ &= \int R(\boldsymbol{\tau}) P(\boldsymbol{\tau}|\boldsymbol{\theta}) \end{aligned} \tag{4}$$

where $P(\boldsymbol{\tau}|\boldsymbol{\theta})$ is the distribution over trajectories $\boldsymbol{\tau}$ for any given policy parameters $\boldsymbol{\theta}$ applied on the actual system:

$$\underbrace{P(\boldsymbol{\tau}|\boldsymbol{\theta})}_{\text{trajectories for } \boldsymbol{\theta}} = \underbrace{p(\boldsymbol{x}_0)}_{\text{initial state}} \prod_t \underbrace{p(\boldsymbol{x}_{t+1}|\boldsymbol{x}_t, \boldsymbol{u}_t)}_{\text{transition dynamics}} \underbrace{\pi(\boldsymbol{u}_t|\boldsymbol{x}_t, t, \boldsymbol{\theta})}_{\text{policy}}. \tag{5}$$

The objective of a *policy search algorithm* is to find the parameters $\boldsymbol{\theta}^*$ that maximize the *expected return* $J(\boldsymbol{\theta})$ when following the policy $\pi_{\boldsymbol{\theta}^*}$:

$$\boldsymbol{\theta}^* = \underset{\boldsymbol{\theta}}{\operatorname{argmax}}\, J(\boldsymbol{\theta}). \tag{6}$$

Most policy search algorithms can be described with a generic algorithm (Algo. 1) and they: (1) start with an initialization strategy (INITSTRATEGY), for instance using random actions, and (2) collect data from the robot (COLLECTSTRATEGY), for instance the states at each discrete time-steps or the reward at the end of the episode; they then (3) enter a loop (for N_{iter} iterations) that alternates between learning one or more models (MODELSTRATEGY) with the data acquired so far, and selecting the next policy $\pi_{\boldsymbol{\theta}_{n+1}}$ to try on the robot (UPDATESTRATEGY). Finally, they return the "optimal" policy using SELECTBESTPOLICYSTRATEGY.

Algorithm 1 Generic policy search algorithm

1: Apply initialization strategy using INITSTRATEGY
2: Collect data, $\boldsymbol{D}_0$, with COLLECTSTRATEGY
3: **for** $n = 1 \rightarrow N_{iter}$ **do**
4: Learn models using MODELSTRATEGY and $\boldsymbol{D}_{n-1}$
5: Calculate $\boldsymbol{\theta}_{n+1}$ using UPDATESTRATEGY
6: Apply policy $\pi_{\boldsymbol{\theta}_{n+1}}$ on the system
7: Collect data, $\boldsymbol{D}_n$, with COLLECTSTRATEGY
8: **end for**
9: **return** $\pi_{\boldsymbol{\theta}^*}$ = SELECTBESTPOLICYSTRATEGY

Algorithm 2 Gradient-free direct policy search algorithm

1: **procedure** INITSTRATEGY
2: Select $\boldsymbol{\theta}_1$ randomly
3: **end procedure**
4: **procedure** COLLECTSTRATEGY
5: Collect samples of the form $(\boldsymbol{\theta}, \frac{\sum_i^N R(\boldsymbol{\tau})_i}{N}) = (\boldsymbol{\theta}, \tilde{J}_{\boldsymbol{\theta}})$ by running policy $\pi_{\boldsymbol{\theta}}$ N times.
6: **end procedure**

This generic outline allows us to describe direct (e.g., policy gradient algorithms [22]), surrogate-based (e.g., Bayesian optimization [20]) and model-based policy search algorithms, where each algorithm implements in a different way each of INITSTRATEGY, COLLECTSTRATEGY, MODELSTRATEGY and UPDATESTRATEGY. We will also see that in this outline we can also fit policy search algorithms that utilize priors; coming from simulators, demonstrations or any other source.

To better understand how policy search is performed, let us use a gradient-free optimizer (UPDATESTRATEGY) and learn directly on the system (i.e., MODELSTRATEGY $= \emptyset$). This type of algorithm falls in the category of *model-free* or *direct* policy search algorithms [1], [23]. INITSTRATEGY can be defined as randomly choosing some policy parameters, $\boldsymbol{\theta}_1$ (Algo. 2), and COLLECTSTRATEGY collects samples of the form $(\boldsymbol{\theta}, \frac{\sum_i^N R(\boldsymbol{\tau})_i}{N})$ by running N times the policy $\pi_{\boldsymbol{\theta}}$. We execute the same policy multiple times because we are interested in approximating the expected return (Eq. (3)). $\tilde{J}_{\boldsymbol{\theta}} = \frac{\sum_i^N R(\boldsymbol{\tau})_i}{N}$ is then used as the value for the sample $\boldsymbol{\theta}$ in a regular optimization loop that tries to maximize it (i.e., the UPDATESTRATEGY is optimizer-dependent).

This straightforward approach to policy search typically requires a large amount of interaction time with the system to find a high-performing solution [1]. Many approaches have been suggested to improve the sample efficiency of model-free approaches (e.g., [4], [22], [24]–[30]). Nevertheless, the objective of the present article is to describe algorithms that require several orders of magnitude less interaction time by leveraging priors and models.

III. USING PRIORS ON THE POLICY PARAMETERS/REPRESENTATION

When designing the policy $\pi(\boldsymbol{u}|\boldsymbol{x}, t, \boldsymbol{\theta})$, the key design choices are what the space of $\boldsymbol{\theta}$ is, and how it maps states to

actions. This design is guided by a trade-off between having a representation that is *expressive*, and one that provides a space that is *efficiently searchable*.

Expressiveness can be defined in terms of the optimal policy π^*_ζ. For a given task ζ, there is theoretically always at least one optimal policy π^*_ζ. Here, we drop $\boldsymbol{\theta}$ to express that we do not mean a specific representation parameterized by $\boldsymbol{\theta}$. Rather π^*_ζ emphasizes that there is some policy (with some representation, perhaps unknown to us) that cannot be outperformed by any other policy (whatever its representation). We use $J_\zeta(\pi^*_\zeta)$ to denote this highest possible expected reward.

A parameterized policy $\pi_{\boldsymbol{\theta}}$ should be expressive enough to *represent* this optimal policy π^*_ζ (or at least come close), i.e.,

$$J_\zeta(\pi^*_\zeta) - \max_{\boldsymbol{\theta}} J_\zeta(\boldsymbol{\theta}) < \delta \tag{7}$$

where δ is some acceptable margin of suboptimality. Note that absolute optimality is rarely required in robotics; in many everyday applications, small tracking errors may be acceptable, and the quadratic command cost does not need to be at the absolute minimum.

On the other hand, the policy representation should be such that it is easy (or at least feasible) to find $\boldsymbol{\theta}^*$, i.e., it should be *efficiently searchable*[4]. In general, smaller values of $\dim(\boldsymbol{\theta})$ lead to more efficiently searchable spaces.

In the following subsections, we describe several common policy representations, which make different trade-offs between expressiveness and being efficiently searchable, and several common strategies to improve the generality and convergence of policy search algorithms.

A. Hand-designed policies

One approach to reducing the policy parameter space is to hand-tailor it to the task ζ to be solved. In [31], for instance, a policy for ball acquisition is designed. The resulting policy only has only four parameters, i.e., $\dim(\boldsymbol{\theta})$ is 4. This low-dimensional policy parameter space is easily searched, and only 672 trials are required to optimize the policy. Thus, prior knowledge is used to find a compact representation, and policy search is used to find the optimal $\boldsymbol{\theta}^*$ for this representation.

One disadvantage of limiting $\dim(\boldsymbol{\theta})$ to a very low dimensionality is that δ may become quite large, and we have no estimate of how much more the reward could have been optimized with a more expressive policy representation. Another disadvantage is that the representation is very specific to the task ζ for which it was designed. Thus, such a policy cannot be reused to learn other tasks. It then greatly limits the transfer learning capabilities of the approaches, since the learned policy can hardly be re-used for any other task.

[4]Analogously, the universal approximation theorem states that a feedforward network with single hidden layer suffices to *represent* any continuous function, but it does not imply that the function is *learnable* from data.

B. Policies as function approximators

Ideally, our policy representation Θ is expressive enough so that we can apply it to many different tasks, i.e.,

$$\operatorname*{argmin}_{\Theta} \sum_{n=1}^{N} J_{\zeta_n}(\pi^*_{\zeta_n}) - \max_{\boldsymbol{\theta}} J_{\zeta_n}(\boldsymbol{\theta}), \text{ with } \boldsymbol{\theta} \in \Theta, \tag{8}$$

i.e., over a set of tasks, we minimize the sum of differences between the theoretically optimal policy π^* for each task, and the optimal policy *given the representation* $\pi_{\boldsymbol{\theta}}$ for each task[5].

A few examples of such generally applicable policy representations are linear policies, radial basis function networks, and neural networks (NN). These more general policies can be used for many tasks [12], [32]. However, prior knowledge is still required to determine the appropriate number of basis functions and their shape. Non-parametric methods partially alleviate the need to such these parameters [33], but the number of basis functions (one for each data point) may become very large and slow down learning. Again, a lower number of basis functions will usually lead to more efficient learning, but less expressive policies and thus potentially higher δ.

One advantage of using a function approximator is that demonstrations can often be used to determine the initial policy parameters. The initial parameters $\boldsymbol{\theta}_1$ can be obtained through supervised learning or other machine learning techniques, by providing the demonstration as training data $(\boldsymbol{x}_i, \boldsymbol{u}_i)_{i=1:N}$. This is discussed in more detail in Section III-G.

The function approximator can be used to generate a single estimate (corresponding to a first order moment in statistics), but it can also be extended to higher order moments. Typically, extending it to second order moments allows the system to get information about the variations that we can exploit to fulfill a task, as well as the synergies between the different policy parameters in the form of covariances. This is typically more expensive to learn—or it requires multiple demonstrations [34]—but the learned representation can typically be more expressive, facilitating adaptation and generalization.

C. Trajectory-based policies

Trajectory-based policy types have been widely used in the robot learning literature [35]–[39], and especially within the policy search problem for robotics [39]–[41]. This type of policy is well-suited for several typical classes of tasks in robotics, such as point-to-point movements or repetitive movements. There exist basically two types of trajectory-based policies: (1) way-point based policies [42], and (2) dynamical system based [35], [41].

One approach to encoding trajectories is to define the policy as a sequence of way-points. In [42], the authors define the problem of motion planning as a policy search problem where the parameters of the policy are the concatenated way-points, $\boldsymbol{w}_i$. They were able to define an algorithm that outperforms several baselines including dynamic programming.

[5]Note that this optimization is never actually performed. It is a mathematical description of what the policy representation designer is implicitly aiming for.

Policies based on dynamical systems have been used more extensively within the robot learning literature as they combine the generality of function approximators with the advantages of dynamical systems, such as robustness towards perturbations and stability guarantees [35], [39]–[41], which are desirable properties of a robotic system.

Perhaps the most widely used trajectory-based policy type within the policy search framework is Dynamical Movement Primitives (DMPs); we can categorize them into discrete DMPs and rhythmic DMPs depending on the type of motion they are describing (point-to-point or repetitive).

Discrete DMPs are summarized in Eq. (9). The canonical system represents the movement *phase* s, which starts at 1, and converges to 0 over time. The transformation systems combines a spring-damper system with a function approximator $f_{\boldsymbol{\theta}}$, which, when integrated, generates accelerations $\ddot{\boldsymbol{\xi}}$. Multi-dimensional DMPs are achieved by coupling multiple transformation systems with one canonical system. The vector $\boldsymbol{\xi}$ typically represents the end-effector pose or the joint angles.

As the spring-damper system converges to $\boldsymbol{\xi}^g$, and s (and thus $s\, f_{\boldsymbol{\theta}}(s)$) converges to 0, the overall system $\boldsymbol{\xi}$ is guaranteed to converge to $\boldsymbol{\xi}^g$. We have:

$$\omega\ddot{\boldsymbol{\xi}} = \underbrace{\alpha(\beta(\boldsymbol{\xi}^g - \boldsymbol{\xi}) - \dot{\boldsymbol{\xi}})}_{\text{Spring-damper system}} + \underbrace{s\, f_{\boldsymbol{\theta}}(s)}_{\text{Forcing term}}, \quad \text{(Transf.)} \quad (9)$$

$$\omega\dot{s} = -\alpha_s s. \quad \text{(Canonical)} \quad (10)$$

This facilitates learning, because, whatever parameterization $\boldsymbol{\theta}$ of the function approximator we choose, a discrete DMP is guaranteed to converge towards a goal $\boldsymbol{\xi}^g$. Similarly, a rhythmic DMP will always generate a repetitive motion, independent of the values in $\boldsymbol{\theta}$. The movement can be made slower or faster by changing the time constant ω.

Another advantage of DMPs is that only one function approximator is learned for each dimension of the DMP, and that the input of each function approximator is the phase variable s, which is always 1D. Thus, whereas the overall DMP closes the loop on the state $\boldsymbol{\xi}$, the part of the DMP that is learned ($f_{\boldsymbol{\theta}}(s)$) is an open-loop system. This greatly facilitates learning, and simple black-box optimization algorithms have been shown to outperform state-of-the-art RL algorithms for such policies [43]. Approaches for learning the goal $\boldsymbol{\xi}^g$ of a discrete movement have also been proposed [44]. Since the goal is constant throughout the movement, few trials are required to learn it.

The optimal parameters $\boldsymbol{\theta}^*$ for a certain DMP are specific to one specific task ζ. Task-parameterized (dynamical) motion primitives aim at generalizing them to variations of a task, which are described with the task parameter vector $\boldsymbol{q}$ (e.g., the 3D pose to place an object on a table [45] or the 3D pose of the end-effector [37]). Similar approaches can be used in contextual policies, see e.g., [46], [47]. Learning a motion primitive that is optimal for all variations of a task (i.e., all $\boldsymbol{q}$ within a range) is much more challenging, because the curse of dimensionality applies to the task parameter vector $\boldsymbol{q}$ just as it does for the state vector $\boldsymbol{x}$ in reinforcement learning. Task-parameterized representations based on the use of multiple coordinate systems have been developed to cope with this curse of dimensionality [48]. These models have only been applied to learning from demonstration applications so far.

DMPs, nevertheless, are time-dependent and thus can produce behaviors that are not desirable; for example, a policy that cannot adapt to perturbations after some time. Stable Estimator of Dynamical Systems (SEDS) [35] explores how to use dynamical systems in order to define autonomous (i.e., time-independent) controllers (or policies) that are asymptotically stable. The main idea of the algorithm is to use a finite mixture of Gaussian functions as the policy, $\dot{\boldsymbol{\xi}} = \pi_{\text{seds}}(\boldsymbol{\xi})$, with specific properties that satisfy some stability guarantees. SEDS, however, requires demonstrated data in order to optimize the policy (i.e., data gathered from experts), although similar ideas have been used within the RL framework [32].

It is important to note that if $\boldsymbol{\xi}$ or $\boldsymbol{w}$ are not defined in joint space (i.e., the control variables), then most of the approaches assume the existence of a low-level controller that can take target accelerations, velocities or positions (in $\boldsymbol{\xi}$ or $\boldsymbol{w}$) and produce the appropriate low-level control commands (e.g., torques) to achieve these targets. Moreover, all the stability and convergence guarantees mentioned in this section apply solely on the behavior or policy dynamics (e.g., stability or convergence of the desired velocity profile in the end-effector space) and not on the robotic system as a whole[6].

D. Learning the controller

If the policy generates a reference trajectory, a controller is required to map this trajectory (and the current state) to robot control commands (typically torques or joint angle velocity commands). This can be done for instance with a *proportional-integral-derivative* (PID) controller [49], or a *linear quadratic tracking* (LQT) controller [50]. The parameters of this controller can also be included in $\boldsymbol{\theta}$, so that both the reference trajectory and controller parameters are learned at the same time. By doing so, appropriate gains [49], [51] or forces [52] for the task can be learned together with the movement required to reproduce the task. Typically, such representation provides a way to coordinate motor commands to react to perturbations, by rejecting perturbations only in the directions that would affect task performance.

E. Learning the policy representation

So far we have described how the policy representation is determined with prior knowledge, and the $\boldsymbol{\theta}$ of this policy is then optimized through policy search. Another approach is to learn the policy representation and its parameters at the same time, as in NeuroEvolution of Augmenting Topologies (NEAT) [53]. It is even possible, in simulation, to co-evolve an appropriate body morphology and policy [54], [55]. These approaches, however, require massive amounts of rollouts, and do not focus on learning in a handful of trials.

[6]One would need to analyze the complete system of the policy, low-level controllers, and robot dynamics to see if the whole system behavior is stable.

F. Hierarchical and Symbolic Policy Representations

To further generalize policies to different contexts, several approaches have been proposed. Daniel et al. propose the use of a hierarchical policy composed of a gating network and multiple sub-policies, and introducing an entropy-based constraint ensuring that the agent finds distinct solutions with different sub-policies [56]. These sub-policies are treated as latent variables in an expectation-maximization procedure, allowing the distribution of the update information between the sub-policies. Higher layers of the hierarchy may be replaced with symbolic representations, as in [57]–[59]. A full discussion of the many approaches in this area is beyond the scope of this article.

G. Initialization with demonstrations / imitation learning

An advantage of using expressive policies is that they are able to learn (close to) optimal policies for many different tasks. A downside is that such policies are also able to represent many suboptimal policies for a particular task, i.e., there will be many local minima. To ensure convergence, it is important that the initial policy parameters are close to the global optimum. In robotics, this is possible through imitation [60]–[62], i.e., the initialization of $\boldsymbol{\theta}$ from a demonstrated trajectory. Starting with a $\boldsymbol{\theta}$ that is close $\boldsymbol{\theta}^*$ greatly reduces the number of samples to find $\boldsymbol{\theta}^*$, and the interplay between imitation and policy search is therefore an important component in micro-data learning.

Initialization with demonstrations is possible if we know the general movement a robot should make to solve the task, and if we can demonstrate it, either by recording our movement, by teleoperating the robot, or by physically guiding the robot through kinesthetic teaching. Each of these modalities has some limitations. Observational learning does not take into account differences between user and robot (in terms of embodiment, kinematic and dynamic capabilities). Dynamic or skillful tasks are difficult to demonstrate by teleoperation and kinesthetic teaching. Recording both force and position information is limited with kinesthetic teaching and observational learning.

Message 1: Using policy structures that are inspired or derived by prior knowledge about the task or the robot at hand is an effective way of creating a policy representation that is expressive enough but also efficiently searchable. If it is further combined with learning from demonstrations (or imitation learning), then it can lead to powerful approaches that are able to learn in just a handful of trials.
Recommended readings: [60], [62]

IV. Learning models of the expected return

With the appropriate policy representation (and/or initial policy parameters) chosen, the policy search in Algorithm 1 is then executed. The most important step is determining the next parameter vector $\boldsymbol{\theta}_{n+1}$ to test on the physical robot.

In order to choose the next parameter vector $\boldsymbol{\theta}_{n+1}$ to test on the physical robot, a strategy is to learn a model $\hat{J}(\boldsymbol{\theta})$ of the expected return $J(\boldsymbol{\theta})$ (Eq. (4)) using the values collected during the previous episodes, and then choose the optimal $\boldsymbol{\theta}_{n+1}$ according to this model. Put differently, the main concept is to optimize $J(\boldsymbol{\theta})$ by leveraging $\hat{J}(\boldsymbol{\theta}|R(\boldsymbol{\tau}|\boldsymbol{\theta}_1), \cdots, R(\boldsymbol{\tau}|\boldsymbol{\theta}_N))$.

A. Bayesian optimization: active learning of policy parameters

Algorithm 3 Policy search with Bayesian optimization

```
procedure CollectStrategy
    Collect samples of the form (θ, R(τ))
end procedure
procedure ModelStrategy
    Learn model Ĵ : θ → J(θ)
end procedure
procedure UpdateStrategy
    θ_{n+1} = argmax_θ AcquisitionFunction(θ|Ĵ)
end procedure
```

The most representative class of algorithms that falls in this category is Bayesian optimization (BO) [20]. BO consists of two main components: a model of the *expected return*, and an *acquisition function*, which uses the model to define the utility of each point in the search space.

BO for policy search follows the generic policy search algorithm (Algo. 1) and implements CollectStrategy, ModelStrategy and UpdateStrategy (Algo. 3). More specifically, a surrogate model, $\hat{J}(\boldsymbol{\theta})$, of the expected return is learned from the data, then the next policy to test is selected by optimizing the AcquisitionFunction. The AcquisitionFunction tries to intelligently exploit the model and its uncertainties in order to trade-off exploration and exploitation.

The main axes of variation are: (a) the way InitStrategy is defined (the most usual approaches are random policy parameters or random actions), (b) the type of model used to learn J, (c) which AcquisitionFunction is used, and (d) the optimizer used to optimize the AcquisitionFunction.

Gaussian Processes Gaussian Process (GP) regression [63] is the most popular choice for the model. 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 [63]. More precisely, it is a distribution over functions, completely specified by its mean function, $m(\cdot)$ and covariance function, $k(\cdot,\cdot)$ and it is computed as follows:

$$\hat{J}(\boldsymbol{\theta}) \sim \mathcal{GP}(m(\boldsymbol{\theta}), k(\boldsymbol{\theta}, \boldsymbol{\theta}')). \tag{11}$$

Assuming $D_{1:t} = \{R(\boldsymbol{\tau}|\boldsymbol{\theta}_1), ..., R(\boldsymbol{\tau}|\boldsymbol{\theta}_t)\}$ is a set of observations, we can query the GP at a new input point $\boldsymbol{\theta}_*$ as follows:

$$p(\hat{J}(\boldsymbol{\theta}_*)|D_{1:t}, \boldsymbol{\theta}_*) = \mathcal{N}(\mu(\boldsymbol{\theta}_*), \sigma^2(\boldsymbol{\theta}_*)). \tag{12}$$

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

$$\begin{aligned} \mu(\boldsymbol{\theta}_*) &= \boldsymbol{k}^T K^{-1} D_{1:t}, \\ \sigma^2(\boldsymbol{\theta}_*) &= k(\boldsymbol{\theta}_*, \boldsymbol{\theta}_*) - \boldsymbol{k}^T K^{-1} \boldsymbol{k}. \end{aligned} \tag{13}$$

For the acquisition function, most algorithms use the Expected Improvement, the Upper Confidence Bound or the Probability of Improvement [20], [64].

Probability of Improvement One of the first acquisition functions is the Probability of Improvement [65] (PI). PI defines the probability that a new test point $\hat{J}(\boldsymbol{\theta})$ will be better than the best observation so far $\boldsymbol{\theta}^+$; since we cannot directly get this information from $D_{1:t}$, in practice we query the approximated model $\hat{J}$ on $D_{1:t}$ and get the best parameters. When using GPs as the surrogate model, this can be analytically computed:

$$\begin{aligned} PI(\boldsymbol{\theta}) &= p(\hat{J}(\boldsymbol{\theta}) > \hat{J}(\boldsymbol{\theta}^+)) \\ &= \Phi\Big(\frac{\mu(\boldsymbol{\theta}) - \hat{J}(\boldsymbol{\theta}^+)}{\sigma(\boldsymbol{\theta})}\Big) \end{aligned} \tag{14}$$

where $\Phi(\cdot)$ denotes the CDF of the standard normal distribution. The main drawback of PI is that it basically performs pure exploitation; in practice, a slightly modified version of PI is used where a trade-off parameter ξ is added [20].

Expected Improvement The Expected Improvement [20] (EI) acquisition function is an extension of PI, where the expected improvement (deviation) from the current maximum is calculated. Again, when using GPs as the surrogate model, EI can be analytically computed:

$$\begin{aligned} &I(\boldsymbol{\theta}) = \max\{0, \hat{J}(\boldsymbol{\theta}) - \hat{J}(\boldsymbol{\theta}^+)\} \\ &EI(\boldsymbol{\theta}) = \mathbb{E}(I(\boldsymbol{\theta})) \\ &= \begin{cases} (\mu(\boldsymbol{\theta}) - \hat{J}(\boldsymbol{\theta}^+))\Phi(Z) + \sigma(\boldsymbol{\theta})\phi(Z), & \text{if } \sigma(\boldsymbol{\theta}) > 0. \\ 0, & \text{otherwise.} \end{cases} \\ &Z = \frac{\mu(\boldsymbol{\theta}) - \hat{J}(\boldsymbol{\theta}^+)}{\sigma(\boldsymbol{\theta})} \end{aligned} \tag{15}$$

where $\phi(\cdot)$ and $\Phi(\cdot)$ denote the PDF and CDF of the standard normal distribution respectively.

Upper Confidence Bound The Upper Confidence Bound (UCB) acquisition function is the easiest to grasp and works very well in practice [64]. When using GPs as the surrogate model, it is defined as follows:

$$UCB(\boldsymbol{\theta}) = \mu(\boldsymbol{\theta}) + \alpha\sigma(\boldsymbol{\theta}) \tag{16}$$

where α is a user specified parameter. When using UCB as the acquisition function, it might be difficult to choose α and the initial hyper-parameters of the kernel (that affect σ) as the range of J and $\boldsymbol{\theta}$ plays a huge role on this. The GP-UCB algorithm [20], [66] automatically adjusts α and provides some theoretical guarantees on the regret bounds of the algorithm.

Entropy Search The Entropy Search (ES) [64] acquisition function selects policy parameters in order to maximally reduce the uncertainty about the location of the maximum of $J(\boldsymbol{\theta})$ in each step. It quantifies this uncertainty through the entropy of the distribution over the location of the maximum, $p_{\max}(\boldsymbol{\theta}) = \mathbb{P}(\boldsymbol{\theta} \in \operatorname{argmin}_{\boldsymbol{\theta}} J(\boldsymbol{\theta}))$. ES basically defines a different ACQUISITIONFUNCTION for BO as follows:

$$ES(\boldsymbol{\theta}) = \operatorname*{argmax}_{\boldsymbol{\theta}} \mathbb{E}[\Delta H(\boldsymbol{\theta})] \tag{17}$$

where $\Delta H(\boldsymbol{\theta})$ is the change in entropy of $p_{\max}$ caused by retrieving a new cost value at location $\boldsymbol{\theta}$.

A thorough experimental analysis [64] concluded that EI can perform better than PI and UCB on artificial objective functions, but more recent experiments on gait learning on a physical robot suggested that UCB can outperform EI in real situations [67]. In most cases, ES outperforms all other acquisition functions at a bigger computation cost [64].

Martinez-Cantin et al. [68] were among the first to use BO as a policy search algorithm; in particular, their approach was able to learn a policy composed of way-points in order to control a mobile robot that had to navigate in an uncertain environment. Since BO is not modeling the dynamics of the system/robot, it can be effective for learning policies for robots with complex (e.g., locomotion tasks, because of the non-linearity created by the contacts) or high-dimensional dynamics. For instance, Bayesian optimization was successfully used to learn policies for a quadruped robot [69] (around 100 trials with a well-chosen 15D policy space), a small biped "compass robot" [67] (around 100 trials with a finite state automata policy), and a pocket-sized, vibrating soft tensegrity robot [70] (around 30 trials with directly controlling the motors). In all of these cases, BO was at least an order of magnitude more data-efficient than competing methods.

Unfortunately, BO scales badly with respect to the dimensionality of the policy space because modeling the objective function (i.e., the expected return) becomes exponentially harder when the dimension increases [71]. This is why all the aforementioned studies employed low-dimensional policy spaces and very well chosen policy structures (i.e., they all use a strong prior on the policy structure). Scaling up BO is, however, an active field of research and various promising approaches (e.g., random embeddings [72] and additive models [73]–[75]) could be applied to robotics in the future. Combining stochastic optimization with learned local models of the expected return can be an alternative to BO and could scale much better with respect to the policy dimensions [30].

B. Bayesian optimization with priors: using non-zero mean functions as a starting point for the search process

One of the most interesting features of BO is that it can leverage priors (e.g., from simulation or from previous tasks) to accelerate learning on the actual task. Perhaps the most

representative algorithm in this area is the "Intelligent Trial & Error" (IT&E) algorithm [15]. IT&E first uses MAP-Elites [15], an evolutionary illumination [76], [77] (also known as quality-diversity [78]) algorithm, to create a repertoire of about 15000 high-performing policies and stores them in a low-dimensional map (e.g., 6-dimensional whereas the policy space is 36-dimensional). When the robot needs to adapt, a BO algorithm searches for the best policy in the low-dimensional map and uses the reward stored in the map as the mean function of a GP. This algorithm allowed a 6-legged walking robot to adapt to several damage conditions (e.g., a missing or a shortened leg) in less than 2 minutes (less than a dozen of trials), whereas it used a simulator of the intact robot to generate the prior.

Gaussian processes with priors Assuming $D_{1:t} = \{R(\boldsymbol{\tau}|\boldsymbol{\theta}_1), ..., R(\boldsymbol{\tau}|\boldsymbol{\theta}_t)\}$ is a set of observations and $R_m(\boldsymbol{\theta})$ being the reward in the map, we can query the GP at a new input point $\boldsymbol{\theta}_*$ as follows:

$$p(\hat{J}(\boldsymbol{\theta}_*)|D_{1:t}, \boldsymbol{\theta}_*) = \mathcal{N}(\mu(\boldsymbol{\theta}_*), \sigma^2(\boldsymbol{\theta}_*)). \tag{18}$$

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

$$\begin{aligned} \mu(\boldsymbol{\theta}_*) &= R_m(\boldsymbol{\theta}_*) + \boldsymbol{k}^T K^{-1}(D_{1:t} - R_m(\boldsymbol{\theta}_{1:t})), \\ \sigma^2(\boldsymbol{\theta}_*) &= k(\boldsymbol{\theta}_*, \boldsymbol{\theta}_*) - \boldsymbol{k}^T K^{-1} \boldsymbol{k}. \end{aligned} \tag{19}$$

The formulation above allows us to combine observations from the prior and the real-world smoothly. In areas where real-world data is available, the prior'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 prior function [15], [79], [80].

Following a similar line of thought but implemented differently, a few recent works [81], [82] use a simulator to learn the kernel function of a GP, instead of utilizing it to create a mean function like in IT&E [15]. In particular, Antonova et al. [81] used domain knowledge for bipedal robots (i.e., *Determinants of Gait* (DoG) [83]) to produce a kernel that encodes the differences in walking gaits rather than the Euclidean distance of the policy parameters. In short, for each controller parameter $\boldsymbol{\theta}$ a score $\text{sc}(\boldsymbol{\theta})$ is computed by summing the 5 DoG and the kernel $k(\cdot, \cdot)$ is defined as $k(\boldsymbol{\theta}_i, \boldsymbol{\theta}_j) = k(\text{sc}(\boldsymbol{\theta}_i), \text{sc}(\boldsymbol{\theta}_j))$. This approach outperformed both traditional BO and state-of-the-art black-box optimizers (Covariance Matrix Adaptation Evolution Strategies; CMA-ES [84]). Moreover, in a follow-up work [82], the authors use NNs to model this kernel instead of hand-specifying it. Their evaluation shows that the learned kernels perform almost as good as hand-tuned ones and outperform traditional BO. Lastly, in this work they were able to make a physical humanoid robot (ATRIAS) walk in a handful of trials.

A similar but more general idea (i.e., no real assumption about the underlying system) was introduced by [85]. The authors propose a Behavior-Based Kernel (BBK) that utilizes trajectory data to compare policies, instead of using the distance in parameters (as is usually done). More specifically, they define the behavior of a policy to be the associated trajectory density $P(\boldsymbol{\tau}|\boldsymbol{\theta})$ and the kernel $k(\cdot, \cdot)$ is defined as $k(\boldsymbol{\theta}_i, \boldsymbol{\theta}_j) = \exp(-\alpha \cdot D(\boldsymbol{\theta}_i, \boldsymbol{\theta}_j))$, where $D(\boldsymbol{\theta}_i, \boldsymbol{\theta}_j)$ is defined as a sum of KL-divergences between the trajectory densities of different policies. Their approach was able to efficiently learn on several benchmarks; e.g., it required on average less than 20 episodes on the mountain car, acrobot and cartpole swing-up tasks. One could argue that this approach does not utilize any prior information, but rather creates it on the fly; nevertheless, the evaluation was only performed with low-dimensional and well-chosen policy spaces.

Wilson et al. [85] proposed to learn models of the dynamics and the immediate reward to compute an approximate mean function of the GP, which is then used in a traditional BO procedure. They also combine this idea with the BBK kernel and follow a regular BO procedure where at each iteration they re-compute the mean function of the GP with the newly learned models. Although, their approach successfully learned several tasks in less than 10 episodes (e.g., mountain car, cartpole swing-up), there is an issue that might not be visible at first sight: the authors combine model learning, which scales badly with the state/action space dimensionality (see Section V), with Bayesian optimization, which scales badly with the dimensionality of the policy space. As such, their approach can only work with relatively small state/action spaces and small policy spaces. Using priors on the dynamics (see Section V-B) and recent improvements on BO (see Section IV-A) could make their approach more practical.

Lober et al. [86] use a BO procedure that selects parameterizations of a QP-based whole body controller [38], [87] in order to control a humanoid robot. In particular, they formulate a policy that includes the QP-based controller (that contains a model of the system and an optimizer) and is parameterized by way-points (and/or switching times). Their approach was able to allow an iCub robot to move a heavy object while maintaining body balance and avoid collisions [86], [88].

Multiple information sources Instead of using the simulator to precompute priors, Alonso et al. [89] propose an approach that has the ability to automatically decide whether it will gain crucial information from a real sample or it can use the simulator that is cheaper. More specifically, they present a BO algorithm for multiple information sources. Their approach relies on entropy search (see Eq. (17)) and they use entropy to measure the information content of simulations and real experiments. Since this is an appropriate unit of measure for the utility of both sources, the algorithm is able to compare physically meaningful quantities in the same units, and trade off accuracy for cost. As a result, the algorithm can automatically decide whether to evaluate cheap, but inaccurate simulations or perform expensive and precise real experiments. They applied their method, called *Multifidelity Entropy Search* (MF-ES), to fine-tune the policy of a cart-pole system and showed that their approach can speed up the optimization process significantly compared to standard BO.

Pautrat et al. [16] also recently proposed to combine BO

with multiple information sources (or *priors*). They define a new ACQUISITIONFUNCTION function for BO, which they call *Most Likely Expected Improvement* (MLEI). MLEI attempts to have the right balance between the likelihood of the priors and the potential for high-performing solutions. In other words, a good expected improvement according to an unlikely model should be ignored; conversely, a likely model with a low expected improvement might be too pessimistic ("nothing works") and not helpful. A model that is "likely enough" and lets us expect some good improvement might be the most helpful to find the maximum of the objective function. The MLEI acquisition function is defined as follows:

$$EIP(\boldsymbol{\theta}, \mathcal{P}) = EI(\boldsymbol{\theta}) \times p(\hat{J}(\boldsymbol{\theta}_{1..t}) \mid \boldsymbol{\theta}_{1..t}, \boldsymbol{\mathcal{P}}(\boldsymbol{\theta}_{1..t}))$$
$$MLEI(\boldsymbol{\theta}, \mathcal{P}_1, \cdots, \mathcal{P}_m) = \max_{\boldsymbol{p} \in \mathcal{P}_1, \cdots, \mathcal{P}_m} EIP(\boldsymbol{\theta}, \boldsymbol{p}) \quad (20)$$

where $\mathcal{P}_i, i = 1 \dots m$ is the set of available priors (where each $\mathcal{P}_i$ is defined similarly to R_m in Eq.(19)). They evaluated their approach in a transfer learning scenario with a simulated arm and in a damage recovery one with both a simulated and a physical hexapod robot. Their approach demonstrates improved performance relative to random trials or a hand-chosen prior (when that prior does not correspond to the new task). Interestingly, this method also is able to outperform the real prior in some circumstances.

Safety-Aware Approaches Another interesting direction of research is using variants of BO for safety-aware learning; that is learning that actively tries to avoid regions that might cause harm to the robot. In [90] the authors proposed an extension of IT&E that safely trades-off between exploration and exploitation in a damage recovery scenario. To achieve this, (1) they generate, through MAP-Elites, a diverse archive of estimations concerning performance and safety criteria and (2) they use this as prior knowledge in a constrained BO [91] procedure that guides the search towards a compensatory behavior and with respect to the safety beliefs. Their algorithm, sIT&E, allowed a simulated damaged iCub to crawl again safely.

Similarly, in [92] Berkenkamp et al. introduced SafeOpt, a BO procedure to automatically tune controller parameters by trading-off between exploration and exploitation only within a safe zone of the search space. Their approach requires minimal knowledge, such as an initial, not optimal, safe controller to bootstrap the search. This allowed a quadrotor vehicle to safely improve its performance over the initial policy.

Message 2: Bayesian optimization is an active learning framework for micro-data reinforcement learning that is effective when using uncertainty-based models and when there exists some prior on the structure of the policy or on the expected return. However, BO is limited to low-dimensional policy spaces.
Recommended readings: [15], [69]

V. LEARNING MODELS OF THE DYNAMICS

Instead of learning a model of the expected long-term reward (Section IV-A), one can also learn a model of the dynamics of the robot. By repeatedly querying this surrogate model, it is then possible to make a prediction of the expected return. This idea leads to *model-based policy search algorithms* [10], [93], in which the trajectory data are used to learn the dynamics model, then policy search is performed on the model [94], [95].

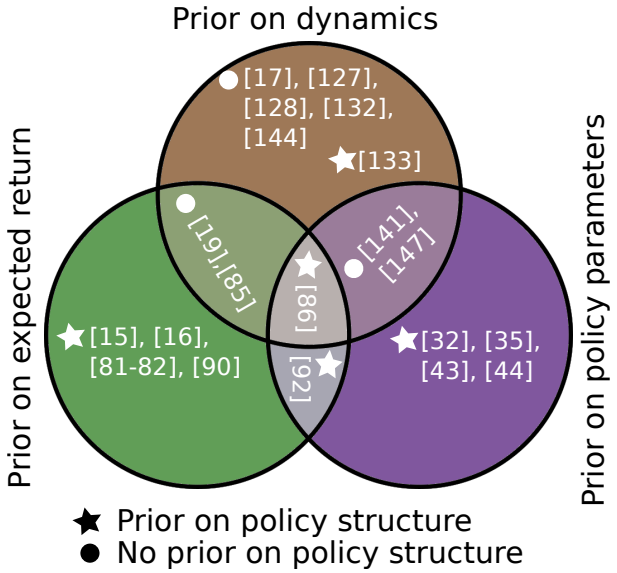

Fig. 2: Main references per prior combination.

Put differently, the algorithms leverage the trajectories $\boldsymbol{\tau}_1, \cdots, \boldsymbol{\tau}_N$ observed so far to learn a function $\hat{f}(\boldsymbol{x}, \boldsymbol{u})$ such that:

$$\hat{\boldsymbol{x}}_{t+1} = \hat{f}(\boldsymbol{x}_t, \boldsymbol{u}_t). \quad (21)$$

This function, $\hat{f}(\boldsymbol{x}_t, \boldsymbol{u}_t)$, is then used to compute an estimation of the expected return, $\hat{J}(\boldsymbol{\theta}|\boldsymbol{\tau}_1, \cdots, \boldsymbol{\tau}_N)$.

A. Model-based Policy Search: alternating between updating the model and learning a policy in the model

Let us consider that the actual dynamics f (and consequently the transition probabilities) are approximated by a model $\hat{f}$ and the immediate reward function r is approximated by a model $\hat{r}$. As such, in model-based policy search we are alternating between learning the models ($\hat{f}$ and $\hat{r}$) and maximizing the expected long-term reward on the model:

$$\hat{J}(\boldsymbol{\theta}) = \mathbb{E}[\hat{R}(\boldsymbol{\tau})|\boldsymbol{\theta}] = \int \hat{R}(\boldsymbol{\tau})\hat{P}(\boldsymbol{\tau}|\boldsymbol{\theta}) \quad (22)$$

where

$$\hat{P}(\boldsymbol{\tau}|\boldsymbol{\theta}) = p(\boldsymbol{x}_0) \prod_t^T \hat{p}(\boldsymbol{x}_{t+1}|\boldsymbol{x}_t, \boldsymbol{u}_t)\pi_{\boldsymbol{\theta}}(\boldsymbol{u}_t|\boldsymbol{x}_t, t). \quad (23)$$

$$\hat{R}(\boldsymbol{\tau}) = \sum_{t=0}^{T-1} \hat{r}_{t+1} = \sum_{t=0}^{T-1} \hat{r}(\boldsymbol{x}_t, \boldsymbol{u}_t, \boldsymbol{x}_{t+1}) \quad (24)$$

This iterative scheme can be seen as follows:

$$\boldsymbol{\tau}_n \sim P(\boldsymbol{\tau}|\boldsymbol{\theta}_n) \quad (25)$$
$$\boldsymbol{D}_n = \boldsymbol{D}_{n-1} \cup \{\boldsymbol{\tau}_n, R(\boldsymbol{\tau}_n)\} \quad (26)$$
$$\boldsymbol{\theta}_{n+1} = \underset{\boldsymbol{\theta}}{\operatorname{argmax}}\, \hat{J}(\boldsymbol{\theta}|\boldsymbol{D}_n) \quad (27)$$

where $\boldsymbol{\theta}_0$ is randomly determined or initialized to some value, $\boldsymbol{D}_0 = \emptyset$ and $\hat{J}(\boldsymbol{\theta}|\boldsymbol{D})$ means calculating $\hat{J}(\boldsymbol{\theta})$ once the models $\hat{f}$ and $\hat{r}$ are learned using the dataset of trajectories and rewards $\boldsymbol{D}$.

Algorithm 4 Model-based policy search

1: **procedure** COLLECTSTRATEGY
2: Collect samples of the form $(\boldsymbol{x}_t, \boldsymbol{u}_t, r_{t+1})$
3: **end procedure**
4: **procedure** MODELSTRATEGY
5: Learn model $\hat{f} : (\boldsymbol{x}_t, \boldsymbol{u}_t) \rightarrow \boldsymbol{x}_{t+1}$
6: Learn model $\hat{r} : (\boldsymbol{x}_t, \boldsymbol{u}_t, \boldsymbol{x}_{t+1}) \rightarrow r_{t+1}$
7: **end procedure**
8: **procedure** UPDATESTRATEGY
9: $\boldsymbol{\theta}_{n+1} = \text{argmax}_{\boldsymbol{\theta}} \hat{J}(\boldsymbol{\theta}|\boldsymbol{D}_n)$
10: **end procedure**

Model-based policy search follows the generic policy search algorithm (Algo. 1) and implements COLLECTSTRATEGY, MODELSTRATEGY and UPDATESTRATEGY (Algo. 4). The main axes of variation are: (a) the way INITSTRATEGY is defined (the most usual approaches are random policy parameters or random actions), (b) the type of models used to learn $\hat{f}$ and $\hat{r}$, (c) the optimizer used to optimize $\hat{J}(\boldsymbol{\theta}|\boldsymbol{D}_n)$, and (d) how are the long-term predictions, given the models, performed (i.e., how Eq. (22) is calculated or approximated).

Model-based policy search algorithms are usually more data-efficient than both direct and surrogate-based policy search methods as they do not depend much on the dimensionality of the policy space. On the other hand, since they are modeling the transition dynamics, practical algorithms are available only for relative small state-action spaces [10], [93].

1) Model learning: There exist many approaches to learn the models $\hat{f}$ and $\hat{r}$ (for model-based policy search) in the literature [9], [96], [97]. Most algorithms assume a known reward function; otherwise they usually use the same technique to learn both models. We can categorize the learned models in deterministic (e.g., NNs or linear regression) and probabilistic ones (e.g., GPs).

Probabilistic models usually rely on Bayesian methods and are typically non-parametric, whereas deterministic models are typically parametric. Probabilistic models are usually more effective than deterministic models in model-based policy search [10], [98] because they provide uncertainty information that can be incorporated into the long-term predictions, thus giving the capability to the optimizer to find more robust controllers (and not over-exploit the model biases). Black-DROPS [99] and PILCO [100] both utilize GPs to greatly reduce the interaction time to solve several tasks, although Black-DROPS is not tied to them and any deterministic or probabilistic model can be used.

The model-based Policy Gradients with Parameter-based Exploration algorithm [96] suggested to directly estimate the transition probabilities $p(\boldsymbol{x}_{t+1}|\boldsymbol{x}_t, \boldsymbol{u}_t)$ using least-squares conditional density estimation [101], instead of learning the model $\hat{f}$. This formulation allowed to bypass some drawbacks of GPs such as computation speed and smoothness assumption (although choosing appropriate kernels in the GPs can produce non-smooth predictions).

Another way of learning models of the dynamics is to use local linear models [97], [102], [103]; i.e., models that are trained on and are only correct in the regions where one controller/policy can drive the system. Guided policy search with unknown dynamics utilizes this scheme and is able to learn efficiently even in high-dimensional states and discontinuous dynamics, like 2D walking and peg-in-the-hole tasks [97], [102] and even dexterous manipulation tasks [103].

There has, also, recently been some work on using Bayesian NNs (BNNs) [104] to improve the scaling of model-based policy search algorithms [105], [106]. Compared to GPs, BNNs scale much better with the number of samples. Nevertheless, BNNs require more tedious hyper-parameter optimization and there is no established, intuitive way to include prior knowledge (apart from the structure). A combination of ensembles and probabilistic NNs has been recently proposed [107] for learning probabilistic dynamics models of higher dimensional systems; for example, state-of-the-art performance was obtained in controlling the half-cheetah benchmark [108] by combining these models with model-predictive control. Recent works showcase that using BNNs with stochastic inputs (and the appropriate policy search procedure) is beneficial when learning in scenarios with multi-modality and heteroskedasticity [109]; traditional model learning approaches (e.g., GPs) fail to properly model these scenarios. Moreover, decomposing aleatoric (i.e., inherent uncertainty of the underlying system) and epistemic (i.e., uncertainty due to limited data) uncertainties in BNNs (with latent input variables) can provide useful information on which points to sample next [110].

Lastly, when performing model-based policy search under partial observability, different model learning techniques should be used. One interesting idea is to optimize the model with the explicit goal of explaining the already observed trajectories instead of focusing on the step-by-step predictions. Doerr et al. [111] recently proposed a principled approach to incorporate these ideas into GP modeling and were able to outperform other robust models in long-term predictions and showcase improved performance for model-based policy search on a real robot with noise and latencies.

2) Long-term predictions: Traditionally, we would categorize the model-based policy search algorithms in those that perform *stochastic long-term predictions* by means of samplings and those that perform *deterministic long-term predictions* by deterministic inference techniques [10]. Recently, an alternative way of computing the expected long-term reward was introduced by [99] (*Policy Evaluation as a Noisy Observation*), where the trajectory generation is combined with the optimization process in order to achieve high-quality predictions with fewer Monte-Carlo rollouts.

a) Stochastic long-term predictions: The actual dynamics of the system are approximated by the model $\hat{f}$, and the immediate reward function by the model $\hat{r}$. The model $\hat{f}$ provides the transition probabilities $\hat{p}(\boldsymbol{x}_{t+1}|\boldsymbol{x}_t, \boldsymbol{u}_t)$. Similarly, the model $\hat{r}$ provides the immediate reward $\hat{r}_{t+1} = \hat{r}(\boldsymbol{x}_t, \boldsymbol{u}_t, \boldsymbol{x}_{t+1})$. When applying a policy (with some parame-

ters $\boldsymbol{\theta}$) on the model, we get a *rollout* or *trajectory*:

$$\boldsymbol{\tau} = (\boldsymbol{x}_0, \boldsymbol{u}_0, \boldsymbol{x}_1, \boldsymbol{u}_1, \dots, \boldsymbol{x}_T) \tag{28}$$

$$\boldsymbol{r} = (\hat{r}_1, \hat{r}_2, \dots, \hat{r}_T) \tag{29}$$

where

$$\boldsymbol{x}_0 \sim p(\boldsymbol{x}_0) \tag{30}$$

$$\hat{r}_{t+1} = \hat{r}(\boldsymbol{x}_t, \boldsymbol{u}_t, \boldsymbol{x}_{t+1}) \tag{31}$$

$$\boldsymbol{u}_t \sim \pi_{\boldsymbol{\theta}}(\boldsymbol{u}_t | \boldsymbol{x}_t, t) \tag{32}$$

$$\boldsymbol{x}_{t+1} \sim \hat{p}(\boldsymbol{x}_{t+1} | \boldsymbol{x}_t, \boldsymbol{u}_t). \tag{33}$$

This is basically sampling the distribution over trajectories, $\hat{P}(\boldsymbol{\tau}|\boldsymbol{\theta})$, which is feasible since the sampling is performed with the models. When applying the same policy (i.e., a policy with the same parameters $\boldsymbol{\theta}$), the trajectories $\boldsymbol{\tau}$ (and consequently $\boldsymbol{r}$) can be different (i.e., stochastic) because (of at least one of the following):

- The policy is stochastic. If the policy is deterministic, then $\boldsymbol{u}_t = \pi_{\boldsymbol{\theta}}(\boldsymbol{x}_t, t)$;
- The models ($\hat{f}$ and/or $\hat{r}$) are probabilistic;
- Of the initial state distribution, $p(\boldsymbol{x}_0)$.

Monte-Carlo & PEGASUS policy evaluation: Once we know how to generate trajectories given some policy parameters, we need to define the way to evaluate the performance of these policy parameters. Perhaps the most straightforward way of computing the expected log-term reward of some policy parameters is to generate m trajectories with the same policy along with their long-term costs and then compute the average (i.e., perform Monte-Carlo sampling):

$$\tilde{\hat{J}}(\boldsymbol{\theta}) = \frac{1}{m} \sum_{i=1}^{m} \hat{R}_i(\boldsymbol{\tau}^i). \tag{34}$$

A more efficient way of computing the expected long-term reward with stochastic trajectories is with the PEGASUS sampling procedure [112]. In the PEGASUS sampling procedure the random seeds for each time step are fixed. As a result, repeating the same experiment (i.e., the same sequence of control inputs and the same initial state) would result into exactly the same trajectories. This significantly reduces the sampling variance compared to pure Monte-Carlo sampling and can be shown that optimizing this *semi-stochastic* version of the model is equivalent to optimizing the actual model.

The advantages of the sampling-based policy evaluations schemes are that each *rollout* can be performed in parallel and that they require much less implementation effort than the deterministic long-term predictions (see Section V-A2b). Nevertheless, these sampling-based procedures can experience big variances in the predictions that may negatively affect the optimization process. In [46] the authors showed that when using enough sample trajectories, better approximations of the expected return can be obtained than the ones of deterministic long-term predictions (see Section V-A2b); moreover, computation time can be greatly reduced by exploiting the parallelization capabilities of modern GPUs. Another recent work [107] also strongly justifies the use of sampling-based policy evaluations over deterministic inference methods (especially in higher dimensional systems).

Probabilistic Inference for Particle-based Policy Search (PIPPS): Recently, Parmas et al. [98] proposed the PIPPS algorithm which effectively combines the Reparameterization gradients (RP) and the Likelihood ratio gradients (LR); they call them Total Propagation (TP). Their paper showcases that LR gradients (and their combined TP gradients) do not suffer from the curse of chaos (or exploding gradients), whereas RP gradients require a very large number of rollouts to accurately estimate the gradients, even for simple problems.

b) Deterministic long-term predictions: Instead of sampling trajectories $\boldsymbol{\tau}$, the probability distribution $\hat{P}(\boldsymbol{\tau}|\boldsymbol{\theta})$ can be computed with deterministic approximations, such as linearization [113], sigma-point methods [114] or moment matching [9]. All these inference methods attempt to approximate the original distribution with a Gaussian.

Assuming a joint probability distribution $\hat{p}(\boldsymbol{x}_t, \boldsymbol{u}_t) = \mathcal{N}(\boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t)$, the distribution $\hat{P}(\boldsymbol{\tau}|\boldsymbol{\theta})$ can be computed by successively computing the distribution of $\hat{p}(\boldsymbol{x}_{t+1})$ given $\hat{p}(\boldsymbol{x}_t, \boldsymbol{u}_t)$. Computing $\hat{p}(\boldsymbol{x}_{t+1})$ corresponds to solving the integral:

$$\hat{p}(\boldsymbol{x}_{t+1}) = \iiint \hat{p}(\boldsymbol{x}_{t+1}|\boldsymbol{x}_t, \boldsymbol{u}_t)\hat{p}(\boldsymbol{x}_t, \boldsymbol{u}_t) d\boldsymbol{x}_t d\boldsymbol{u}_t d\boldsymbol{w}. \tag{35}$$

This integral can be computed analytically only if the transition dynamics $\hat{f}$ are linear (in that case $\hat{p}(\boldsymbol{x}_{t+1})$ is Gaussian). This is rarely the case and as such, approximate inference techniques are used. Usually, we approximate $\hat{p}(\boldsymbol{x}_{t+1})$ as a Gaussian; this can be done either by linearization [113], sigma-point methods [114] or moment matching [9]. The PILCO algorithm [100] uses moment matching, which is the best unimodal approximation of the predictive distribution in the sense that it minimizes the KL-divergence between the true predictive distribution and the unimodal approximation [10].

One big advantage of using deterministic inference techniques for long-term predictions is the low-variance they exhibit in the predictions. In addition, using these inference techniques allows for analytic gradient computation and as such we can exploit efficient gradient-based optimization. However, each of these inference techniques has its own disadvantages; for example, exact moments (for moment matching) can be computed only in special cases since the required integrals might be intractable, which limits the overall approach (e.g., PILCO requires that the reward function is known and differentiable).

The PILCO algorithm [9] uses this type of long-term predictions and it was the first algorithm that showed remarkable data-efficiency on several benchmark tasks (e.g., less than 20 seconds of interaction time to solve the cart-pole swing-up task) [100]. It was also able to learn on a physical low-cost manipulator [115] and simulated walking tasks [116] among the many successful applications of the algorithm [9].

c) Policy evaluation as a noisy observation: This approach [99] exploits the *implicit averaging* property [117]–[119] of population, rank-based optimizers, like CMA-ES [120], in order to perform sampling-based evaluation of the trajectories efficiently (i.e., reducing the computation time of the policy search on the model). The key idea is that when using this type of optimizers, the problem can be

transformed into a noisy optimization one, thus, there is no need to (fully) compute the expected long-term reward, as this expectation can be implicitly computed by the optimizer. Similar ideas have been previously explored for model-free policy search [121].

In more detail, instead of performing deterministic long-term predictions, like PILCO, or Monte-Carlo evaluation, like PEGASUS, Black-DROPS stochastically generates trajectories, but considers that each of these trajectories (or rollouts) is a measurement of a function $G(\boldsymbol{\theta})$ that is the actual function $\hat{J}(\boldsymbol{\theta})$ perturbed by a noise $N(\boldsymbol{\theta})$:

$$G(\boldsymbol{\theta}) = \hat{J}(\boldsymbol{\theta}) + N(\boldsymbol{\theta}). \tag{36}$$

It is easy to verify that maximizing $\mathbb{E}\Big[G(\boldsymbol{\theta})\Big]$ is equivalent to maximizing $\hat{J}(\boldsymbol{\theta})$, when $\mathbb{E}\Big[N(\boldsymbol{\theta})\Big] = \text{constant}$.

Implicit averaging and noisy functions: Seeing the maximization of $\hat{J}(\boldsymbol{\theta})$ as the optimization of a noisy function allows to maximize it without computing or estimating it explicitly. The Black-DROPS algorithm ulitizes a recent variant of CMA-ES (i.e., one of the most successful algorithms for optimizing noisy and black-box functions [117], [122], [123]) that combines random perturbations with re-evaluation for uncertainty handling [122] along with restart strategies for better exploration [124].

While Black-DROPS has the same data-efficiency as PILCO, it has the added benefit of being able to exploit multi-core architectures, thus, greatly reducing the computation time [99]. Similar to most Monte-Carlo methods (like GP-REPS [46]), Black-DROPS is a purely black-box model-based policy search algorithm; i.e., one can swap the model types, reward functions and/or initialization procedure with minimal effort. This is an important feature as it allows us to more easily exploit good sources of prior information [17]. Black-DROPS was able to learn in less than 20 seconds of interaction time to solve the cartpole swing-up task as well as to control a physical 4-DOF physical manipulator in less than 5-6 episodes.

B. Using priors on the dynamics

Reducing the interaction time in model-based policy search can be achieved by using priors on the models [17], [79], [125]–[129]; i.e., starting with an initial guess of the dynamics (and/or the reward function) and then learning the residual model. This type of algorithm follow the general model-based policy search framework (Algo. 4) and usually implement different types of INITSTRATEGY. Notably, the most successful approaches rely on GPs to model the dynamics, as priors can be very elegantly incorporated.

Gaussian processes with priors for dynamical models Assuming $D_{1:t} = \{f(\tilde{\boldsymbol{x}}_1), ..., f(\tilde{\boldsymbol{x}}_t)\}$ is a set of observations, $\tilde{\boldsymbol{x}}_t = (\boldsymbol{x}_t, \boldsymbol{u}_t) \in \mathbb{R}^{E+F}$ and $M(\tilde{\boldsymbol{x}})$ being the simulator function (i.e., the initial guess of the dynamics), we can query the GP at a new input point $\tilde{\boldsymbol{x}}_*$ similar to Eq. (18)-(19) (we provide only the mean prediction for notation):

$$\mu(\tilde{\boldsymbol{x}}_*) = M(\tilde{\boldsymbol{x}}_*) + \boldsymbol{k}^T K^{-1}(D_{1:t} - M(\tilde{\boldsymbol{x}}_{1:t})) \tag{37}$$

Of course, we have E independent GPs; one for each output dimension [99], [100].

A few approaches [125], [130] use simple analytic and fast simulators to create a GP prior of the dynamics (and assume the reward function to be known). PILCO with priors [127] uses simulated data (from running PILCO in the simulator) to create a GP prior for the dynamics and then performs policy search with PILCO. It was able to increase the data-efficiency of PILCO in a real inverted pendulum using a very simple model as a prior. A similar approach, PI-REM [128], utilizes analytic equations for the dynamics prior and tries to actively bring the real trials as close as possible to the simulated ones (i.e., reference trajectory) using a slightly modified PILCO policy search procedure. PI-REM was also able to increase the data-efficiency of PILCO in a real inverted pendulum (with variable stiffness actuators) using a simple model as a prior.

Black-DROPS with priors [17] proposes a new GP learning scheme that combines model identification and non-parametric model learning (called GP-MI) and then performs policy search with Black-DROPS. The main idea of GP-MI is to use simulators with tunable parameters, i.e., mean functions of the form $M(\tilde{\boldsymbol{x}}, \boldsymbol{\phi}_M)$ where each vector $\boldsymbol{\phi}_M \in \mathbb{R}^{n_M}$ corresponds to a different prior model of the 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 authors formulate it in a way so that they can exploit the GP framework 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.

Black-DROPS with GP-MI was able to robustly learn controllers for a pendubot swing-up task [131] even when the priors were misleading. More precisely, it was able to outperform Black-DROPS, PILCO, PILCO with priors, Black-DROPS with fixed priors (i.e., this should be similar to PI-REM) and IT&E. Moreover, Black-DROPS with GP-MI was able to find high-performing walking policies for a physical damaged hexapod robot (48D state and 18D action space) in less than 1 minute of interaction time and outperformed IT&E that excels in this setting [15], [17].

Following a similar rationale, VGMI [132], uses a Bayesian optimization procedure to find the simulator's mechanical parameters so as to match the real-world trajectories (i.e., it performs model identification) and then performs policy search on the updated simulator. In particular, VGMI was able to learn policies for a physical dual-arm collaborative task and out-performed PILCO.

Finally, an approach that splits the self-modeling process from the policy search is presented in [133]. The authors were among the first ones to combine a self-modeling procedure (close to model identification [134]) with policy search. The self-modeling part of their approach consists of 3 steps: (a) action executing and data-collection, (b) synthesization of 15 candidate self-models that explain the sensory data and (c) active selection of the action that will elicit the most information from the robot. After a few cycles of these steps (i.e., around 15), the most accurate model is selected and

policy search is performed to produce a desired behavior. Their approach was able to control in less than 20 episodes a four-legged robot and it was also able to adapt to damages in a few trials (by re-running the self-modeling procedure).

Message 3: Model-based policy search algorithms are the most data-efficient algorithms, especially when they take into account the uncertainty of the model. While they typically suffer from the curse of dimensionality (state/action space), endowing them with prior knowledge on the dynamics can reduce their interaction time requirements even when learning with high-dimensional or complicated systems. The main challenge in this direction is to overcome the computational complexity of the approaches.
Recommended readings: [9], [17], [99]

VI. Other approaches

A. Guided policy search

Guided policy search (GPS) with unknown dynamics [97], [102] is a somewhat hybrid approach that combines local trajectory optimization (that happens directly on the real system), learning local models of the dynamics (see Section V-A1) and indirect policy search where it attempts to approximate the local controllers with one big NN policy (using supervised learning). In more detail, GPS consists of two loops: an outer loop that executes the local linear-Gaussian policies on the real system, records data and fits the dynamics models and an inner loop where it alternates between optimizing the local linear-Gaussian policies (using trajectory optimization and the fitted dynamics models) and optimizing the global policy to match all the local policies (via supervised learning and without utilizing the learned models) [102].

The results of GPS show that it is less data-efficient than model-based policy search approaches, but more data-efficient than traditional direct policy search. Moreover, GPS is able to handle bigger state-action spaces (i.e., it has also been used with image observations [102]) than traditional model-based policy search approaches as it reduces the final policy optimization step in a supervised one that can be efficiently tackled with all the recent deep learning methods [2]. GPS was able to learn in less than 100 episodes even in high-dimensional states and discontinuous dynamics like 2D walking, peg-in-the-hole task and controlling an octopus robot [97], [102] among the many successful applications of the algorithm [135], [136].

B. Transferability approaches

The main hypothesis of the transferability approach [137], [138] is that physics simulators are accurate for some policies, e.g., static gaits, and inaccurate for some others, e.g., highly dynamic gaits. As a consequence, it is possible to learn in simulation if the search is constrained to policies that are simulated accurately. As no simulator currently comes with an estimate of its accuracy, the key idea of the transferability approach is to learn a model of a *transferability function*, which predicts the accuracy of a simulator given policy parameters or a trajectory in simulation. This function is often easier to learn than the expected return because this is essentially a classification problem (instead of regression). In addition, small errors in the model have often little consequences, because the search is mainly driven by the expected return in simulation (and not by the transferability optimization).

The resulting learning process requires only a handful trials on the physical robot (in most of the experiments, less than 25); however, the main drawback is that it can only find policies that perform similarly in simulation and in reality (e.g., static gaits versus highly dynamic gaits). These type of algorithms were able to efficiently learn policies for mobile robots that have to navigate in mazes [137] (15 trials on the robot), for a walking quadruped robot [137], [139] (about 10 trials) and for a 6-legged robot that had to learn how to walk in spite of a damaged leg without updating the simulator [138] (25 trials). Similar ideas were recently developed for humanoid robots with QP-based controllers [38].

C. Simulation-to-reality & meta-learning approaches

The main idea behind meta-learning and *SimToReal* approaches is to find a policy that is robust to a distribution of tasks (or environments). *SimToReal* approaches exploit parameterized simulators in order to learn a policy that can effectively transfer on the real system. SimToReal algorithms can be categorized into ones that find policies that are robust: (1) to visual differences [140]–[143] (*domain randomization*), and (2) to different dynamics properties [144]–[146] (*dynamics randomization*).

James et al. [141] use a rather simple controller, sample different goal targets and visual conditions (e.g., lighting, textures) and collect 1 million state-action trajectories of completing different goals. Once this dataset is collected, a convolutional NN, that will later serve as the policy, is trained in a supervised manner to find a mapping between image observations and the appropriate actions to take. Finally, they deploy this policy in the real world. Astonishingly, they were able to get 100% success rate in the real world scenarios despite the fact that their task involved contacts and anticipating dynamic effects (i.e., picking and placing objects in a basket). Peng et al. [146] use the Hindsight Experience Replay [147] algorithm in order to maximize the expected return across a distribution of dynamics models. The dynamics parameters include masses and lengths of the links, damping and friction coefficients among others. Using their algorithm a 7-DOF manipulator learned how to push a puck on a desired location and directly transfered from simulation to reality.

However, these approaches do not provide any online adaptation capabilities; this basically means that if for some reason the policy does not generalize to the real world instance, the robot cannot improve its performance. SimOpt [144] tries to close the loop by using real experience in order to find the distribution of the dynamics models to optimize on, but this type of approaches is very similar to model-based policy search with priors on the dynamics models (see Sec. V-B). We can draw a parallel here and argue that model-based policy search with probabilistic models is performing something similar to

dynamics randomization. More concretely, performing policy search under an uncertain model is equivalent to finding a "robust" policy that can perform well under various dynamics models: the ones defined by the mean predictions and the uncertainty of the model.

Similarly, meta-learning approaches [148]–[151] do not only try to find a robust policy but also a learning rule that can allow for fast adaptation (i.e., good performance with few gradient steps). Model-Agnostic Meta-Learning (MAML) [149] learns a good set of initial policy parameters, $\boldsymbol{\theta}_0$, such that every task can be solved within few gradient steps. A few applications of meta-learning target fast robot adaptation with promising results [150], [151]. For example, Sæmundsson et al. [151] model the distribution over systems using a latent embedding and model the dynamics using a global function (with GPs) conditioned on the latent embedding. They were able to learn control policies for the cartpole swing-up and the double pendulum tasks in less than $30\,s$ of interaction time including the meta-training time. Clavera et al. [150] use MAML to train a dynamics model prior such that, when combined with recent data, this prior can be rapidly adapted to the local context. They were able to combine their dynamics model with MPC in order to control a six-legged miniature physical robot in unknown/new situations (e.g., payload or different terrains), but still required 30 minutes of interaction time for the meta-training process.

Message 4: Simulation-to-reality or meta-learning approaches can produce robust and adaptive policies that offer fast adaptation at test time. While they typically require expensive interaction time before the mission (e.g., in simulation), this should not be feared, as they can possibly produce the right prior for the task at hand. If they are combined with some online adaptation or model-learning [152], they can learn effectively.
Recommended readings: [144] [150] [151]

VII. Challenges and Frontiers

A. Scalability

Most of the works we described so far have been demonstrated with simple robots and simple tasks, such as the cartpole swing-up task (4D state space, 1D action space) [100] or simple manipulators (4D state space, 4D action space) [99]. By contrast, humanoid robots have orders of magnitude larger state-action spaces; for example, the 53-DOF iCub robot [153] has a state space of more than 100 dimensions (not counting tactile and visual sensors [154]). Most of the current micro-data approaches are unable to learn with such complex robots.

On the one hand, model-based policy search algorithms (Section V-A) generalize well to new tasks (since the model does not depend on the task) and learn high-dimensional policies with little interaction time (since the policy search happens within the model and not in interaction with the robot); but they do not scale well with the size of the state space: in the general case, the quantity of data to learn a good approximation of the forward model scales exponentially with the dimensionality of the state-space (this is the curse of dimensionality, see [71]). A factored state representation may provide the means to tackle such complexity, for example, by using dynamic Bayesian networks [155] to represent the forward model [156], but we are not aware of any recent work in this direction.

On the other hand, direct policy search algorithms (Sections III-G and IV) can be effective in learning control policies for high-dimensional robots, because the complexity of the learning problem mostly depends on the number of parameters of the policy, and not on the dimensionality of the state-space; however, they do not generalize well to new tasks (when there is a model, it is specific to the reward) and they require a low-dimensional policy. Such a low-dimensional policy is an important, task-specific prior that constrains what can be learnt. For example, central pattern generators can be used for rhythmic tasks such as locomotion [157], but they are unlikely to work well for a manipulation task; similarly, quadratic programming-based controllers (and in general model-based controllers) can facilitate learning whole body controllers for humanoid robots [38], [158], but they impose the control strategy and the model.

In summary, model-based policy search algorithms scale well with the dimensionality of the policy, but they do not scale with the dimensionality of the state space; and direct policy search algorithms scale well with the dimensionality of the state-space, but not with the dimensionality of the policy. None of these two approaches will perform well on every task: future work should focus on either scaling model-based policy search algorithms so that they can learn in high-dimensional state spaces, or scaling direct policy search algorithms so that they can use higher-dimensional policies.

The dimensionality of the sensory observations is also an important challenge for micro-data learning: to our knowledge, no approach that performs "end-to-end learning", that is, learning with a raw data stream like a camera, has the efficiency of micro-data learning. Deep RL has recently made possible to learn policies from raw pixel input [3], largely because of the prior (i.e., an architectural inductive bias) provided by convolutional networks. However, deep RL algorithms typically require a very large interaction time with the environment (e.g., 38 days of play for Atari 2600 games [3]), which is not compatible with most robotics experiments and applications. To address this challenge, a potential starting point is to use unsupervised learning to learn low-dimensional features, which can then be used as inputs for policies. Interestingly, it is possible to leverage priors to learn such state representations from raw observations in a reasonable interaction time [159], [160]. It is also possible to create forward models in image space, that is, predicting the next image knowing the current one and the actions, which would allow to design model-based policy search algorithms that work with an image stream [161]–[164].

B. Priors

Evolution has endowed animals and humans with substantial prior knowledge. For instance, hatchling turtles are prewired

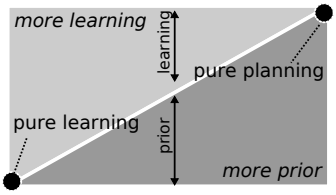

Fig. 3: The trade-off between prior knowledge and learning: for any task, there is an infinity of combinations between the amount of prior knowledge and the amount of learning required (image based on a slide by Oliver Brock, 2017).

to run towards the sea [165]; or marine iguanas are able to run and jump within moments of their birth in order to avoid being eaten by snakes[7]. These species cannot rely on online learning mechanisms for mastering these behaviors: without such priors they would simply cease to exist.

Similarly to priors obtained from nature, artificial agents or robots can learn very quickly when provided with the right priors, as we presented in Sections III, IV-B, and V-B. In other words, priors play a catalytic role in reducing the interaction time of policy search methods. Thus, the following questions naturally arise (Fig. 3): what should be innate and what should be learned? and how should the innate part be designed?

Most of the existing methodologies use task-specific priors (e.g., demonstrations). Such priors can greatly accelerate policy search, but have the disadvantage of requiring an expert to provide them for all the different tasks the robots might face. More generic or task-agnostic priors (e.g., properties of the physical world) could relax these assumptions while still providing a learning speedup. Some steps have been made into identifying such task-agnostic priors for robotics, and using them for state representation [159], [166]. We believe this is an important direction that requires more investigation. Meta-learning [148]–[151] is a related line of work that can provide a principled and potentially automatic way of designing priors.

Physical simulations can also be used to automatically generate priors while being a very generic tool [15], [16], [81], [82]. By essence, physical simulations can run in parallel and take advantage of faster computing hardware (from clusters of CPUs to GPUs): learning priors in simulation could be an analog of the billions of years of evolution that shaped the learning systems of all the current lifeforms.

While priors can bootstrap policy search, they can also be misleading when a new task is encountered. Thus, an important research avenue is to design policy search algorithms that can not only incorporate well-chosen priors, but also ignore those that are irrelevant for the current task [17]. Following this line of thought, a promising idea is to design algorithms that actively select among a variety of priors [16].

C. Generalization and robustness

The majority of aforementioned articles are not much concerned with the generalization abilities and the robustness of the learned policy: they are designed to solve a single task, in a single context, with often little evaluation of the abilities to reject perturbations. For example, IT&E [15] focused on a repertoire of forward walking gaits for a hexapod robot on flat ground, rather than on various surfaces (e.g., incline surface or stairs) or various directions [80]; PILCO was applied for stacking a tower of foam blocks with a robotic manipulator [9], but the task remained fixed over the course of learning (e.g., the size of the cubes did not vary) and there were no external perturbations (e.g., a wind gust). Put differently, in most of the reported experiments, the algorithms are very likely to have "overfitted" the robot and the task.

This situation could appear surprising because generalization and robustness are two of the most important questions in machine learning and control theory [134]. Its source is, however, straightforward: assessing the robustness or the generalization abilities of a policy typically requires a significant additional interaction time. For example, a typical approach to measure the robustness of a control policy is to evaluate it with many different starting conditions and perturbations; a similar technique is often used to test the generalization abilities. Nevertheless, such an approach multiplies the interaction time by the number of tested conditions, which is likely to make the algorithm very quickly intractable on a real robot. In addition, this problem is amplified when the dimension of the state space increases, since there exist many more ways of perturbing a high-dimensional system than a low-dimensional one.

A potential remedy is to use policies that are intrinsically robust to some perturbations, that is, designing the policy space such that a change in the parameter space keeps the policy robust. For instance, the learning algorithm could search for a trajectory and a controller could be designed to follow it in a robust way: this corresponds to traditional trajectory optimization (or planning) in robotics [134]. This is one of the ideas behind dynamic movement primitives (see Section III), which act like "attractors" towards a trajectory of a fixed point. Similarly, it is possible to learn waypoints [86] or "repulsors" [38] to mix learning with advanced, closed-loop "whole-body" controllers. It is, also, possible to incorporate optimization layers (e.g., a QP program [167]) in a NN in order to take advantage of the structure they provide. Lastly, one can learn distinct *soft* policies for simpler tasks and then compose them in order to achieve a more complicated task [168].

It is also conceivable to learn models of the generalization abilities [169], although it has, to our knowledge, never been tested with real robots. In that case, a model is trained to distinguish between behaviors (or trajectories) that are likely to overfit from those that are likely to be robust. This model can then be used in a policy search algorithm (e.g., in a constrained BO scheme).

Ultimately, we would like to have robots that can learn to execute various tasks quickly under varying conditions. This means that they need to be able to generalize from their previous experience without requiring much interaction time when the task changes. Having a policy that generalizes well offers the benefit of very fast execution, as opposed to algorithms that perform planning [80] or model identification [17]. This challenge of micro-data multitask learning can be decomposed into two challenges. The first is about learning quickly to

[7]As portrayed in the recent documentary "Planet Earth 2" from BBC.

achieve different goals (i.e., only the reward function changes between tasks, for example, a robot that needs to throw a dart at different specified targets), while the second challenge is about adapting quickly to changes in the dynamics (i.e., the reward function does not change, for example, a robot that needs to cover as much distance forward as possible while walking on grass and transitioning on slippery ground).

Learning to achieve multiple goals has been tackled by a variety of methods, from using goal-conditioned policies, both in model-free (e.g., [47], [147], [149], [170]–[179]) and model-based settings (e.g., [46], [180]), to creating behavioral repertoires (e.g., [15], [77], [80]). Fast adaptation to changing dynamics could be addressed through BO (e.g., [16], [181]), meta-learning (e.g., [150], [151], [182]–[184]), model identification (e.g., [17], [185]), or generally policies that are robust to changes in the dynamics (e.g., [146], [186]).

D. Interplay between planning, model-predictive control and policy search

The data-efficiency of policy search algorithms like PILCO or Black-DROPS rises from the fact that they learn and use dynamical models (Section V-A). However, if we assume that the dynamical model is known or can be learnt, there is a large literature on control methods that can be used. So, is policy search the right approach in such a case?

A fundamental controller from control theory is the linear-quadratic regulator (LQR) [187], which is optimal when the the dynamics are linear and the cost function is quadratic. Systems with nonlinear dynamics can be tackled with LQR by linearizing them around the current state and action, however, other approaches can be used such as differential dynamic programming [188], [189] and its simpler variant, the iterative linear-quadratic Gaussian algorithm [190] (iLQG). Generally, these methods can be used for optimal control with a large horizon lookahead, however, doing so can be computationally costly. For this reason, they are mostly employed to calculate trajectories offline; for example, GPS uses iLQG as the trajectory optimization procedure.

A way to permit online trajectory optimization is by reducing the horizon lookahead, thus, gaining in computational efficiency. This is known as model-predictive control (MPC) [11]. Using shorter horizons, MPC is no longer optimal with respect to the overall, high-level task. This means that MPC can be used for short-term tasks, such as tracking a trajectory, which can be produced offline. The advantage of MPC is that it can get feedback from the real system and replan at every step. Such a control scheme can be very effective and has, for example, recently allowed real-time whole-body control of humanoid robots [191].

Although MPC can replan at every step, it still has the disadvantage of relying on models. Models can be inaccurate or wrong (especially in the first episodes of learning), therefore, there needs to be a mechanism that corrects the mismatch. A potential solution could be to combine iterative learning control [192], [193] with MPC (e.g., see [163], [194]–[196]). MPC additionally has the disadvantage of requiring full knowledge of the system state. This problem can be mitigated by combining MPC with policy search. For example, in [197], the authors used MPC with full state information during training, to learn NN policies that do not require full state information (only raw observations) when deployed, and even run faster than MPC online.

Should we then learn a big NN policy for complex high-level tasks, such as a humanoid robot helping with the house chores? Firstly, we need to consider that such complex tasks require long planning horizons. Secondly, as the task becomes more complex, so could potentially the policy space. Even if we do not consider memory requirements, learning such tasks from scratch would be intractable, even in simulation. One way of addressing such complexity is by decomposing the high-level task into a hierarchy of subtasks. Sampling-based planners [198], [199] could operate at the high to mid levels of the hierarchy, whereas MPC could operate at the mid to low levels. Furthermore, policy search (or other algorithms for optimal control) can be used to discover primitives which themselves are used as components of a higher-level policy (e.g., see [200]) or a planning algorithm (e.g., see [80], [201]).

E. Computation time

Micro-data learning focuses on the desirable property of reducing the interaction time. However, most articles purposefully neglect computation time because they assume that it will be tackled automatically with faster hardware in the future. Although this is possible, it is worth investigating how different algorithms can potentially be sped up for near real-time execution with today's hardware.

For illustration, PILCO (see Section V-A) is a very successful and data-efficient algorithm, but can be very computationally expensive when the state-action or policy space dimensionality increases [85], [99] (e.g., Wilson et al. [85] report that PILCO required 3 weeks of computation time for 20 episodes on a 3-link planar arm task) and cannot take advantage of multi-core architectures. Black-DROPS and Black-DROPS with GP-MI (see Section V-B) can greatly reduce the interaction time and take advantage of multi-core architectures, but they still require a considerable amount of computation time (e.g., Black-DROPS with GP-MI required 24 hours on a modern 16-core computer for 26 episodes of the pendubot task [17]). Both approaches use GP models which have a complexity that is quadratic to the number of points when queried; this is clearly inefficient when millions of such GP queries (e.g., Black-DROPS performs around 64M [99]) are performed in each episode.

On the other hand, IT&E [15] and "robust policies" (e.g., see [146], [185], [181], [186]) can practically run in real-time because the prior is pre-computed offline. This "recipe" is shared by recent meta-learning methodologies, such as [149], that aim to learn an expressive policy that can be optimized online using a single gradient update.

This does not mean that the offline precomputation time should not be optimized. Algorithms such as IT&E or the work in [146] use a form of directed exploration to create such a prior. If, for example, random search were used, it would probably need orders of magnitude more computation time to create a prior of the same quality.

VIII. Conclusions

Thanks to recent advances in priors, policy representations, reward modeling, and dynamical models, it is now possible to learn policies on robots in a few minutes of interaction time. These micro-data learning algorithms considerably expand the usefulness of learning on robots: with these algorithms, we can envision robots that adapt "in front of our eyes". These algorithms, nonetheless, face critical challenges, most notably to scale-up simultaneously to high-dimensional state spaces and high-dimensional policy spaces.

As guidelines for future work in the field, we propose 5 precepts that summarize the "generic rules" that govern most of the work published so far about micro-data learning:

1) Leveraging prior knowledge is key for micro-data learning: it should not be feared. However, the prior knowledge should be as explicit and as generic as possible.
2) Use as much data as possible from each trial (e.g., trajectory data, not only reward value): when data is scarce, every bit matters.
3) Take the time to choose what to test next (active learning): computers are likely to become faster in the future, but physics will not accelerate; it is therefore a sensible strategy to trade data resources for computational resources. It is still desirable, but less critical on the long term, to design algorithms that are fast enough to run on embedded systems.
4) Every estimate (or model) should come with a measure of its uncertainty: when very little data is available, models will never have enough data to be "right" for the whole search space; algorithms must take this fact into account and reason with this uncertainty.
5) If needed, use expensive algorithms before the mission: since we mostly care about online adaptation, we can have access to time and resources before the mission (access to computing clusters, GPUs, etc.)

Finally, we would like to give a few recommendations for practical usage of micro-data algorithms:

- **Low-DOF robots:** For robots with less than 10 DOFs, model-based policy search algorithms should be the choice of the researcher. Algorithms like Black-DROPS [99] and PILCO [100] will operate within reasonable computation time and will learn in very few trials.
- **High-DOF robots:** For robots with higher dimensional state/action space but with low dimensional policy spaces, Bayesian optimization approaches will provide the best trade-off between computation time and learning convergence. If a prior model or simulator is available, algorithms like IT&E [15] and MF-ES [89] should be on the front line of learning in just a few trials.
- **Complex robots:** For robots with higher dimensional state/action space and high dimensional policy spaces, model-based policy search with priors on the dynamics will provide the most data-efficient results at the expense of increased computation cost. Algorithms like Black-DROPS with GP-MI [17] and VGMI [132] effectively exploit parameterized simulators and should be able to learn in a handful of trials even for complex robots.
- **Raw observations:** When the observation (or state) space is very high dimensional (e.g., visual input), *SimTo-Real* methods combined with online adaptation (e.g., SimOpt [144]) should provide the best results.

In all cases, a good policy space and initialization of the policy parameters (e.g., from demonstrations [62]) will accelerate learning.

IX. Acknowledgements

This project received funding from: the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (GA no. 637972, project "ResiBots"); the Helmholtz Association through the project "Reduced Complexity Models"; the European Commission through the projects H2020 AnDy (GA no. 731540) and MEMMO (GA no. 780684); the CHIST-ERA project "HEAP"; the European Union's Horizon 2020 research and innovation programme under grant agreement No 739578 complemented by the Government of the Republic of Cyprus through the Directorate General for European Programmes, Coordination and Development.

References

[1] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press Cambridge, 1998.
[2] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.
[3] V. Mnih *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, p. 529, 2015.
[4] ——, "Asynchronous methods for deep reinforcement learning," in *ICML*, 2016.
[5] D. Silver *et al.*, "Mastering the game of go without human knowledge," *Nature*, vol. 550, no. 7676, p. 354, 2017.
[6] N. Heess *et al.*, "Emergence of locomotion behaviours in rich environments," *arXiv preprint arXiv:1707.02286*, 2017.
[7] S. Levine, P. Pastor, A. Krizhevsky, J. Ibarz, and D. Quillen, "Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection," *IJRR*, vol. 37, no. 4-5, pp. 421–436, 2018.
[8] J.-B. Mouret, "Micro-data learning: The other end of the spectrum," *ERCIM News*, no. 107, p. 2, 2016.
[9] 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.
[10] 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.
[11] C. E. Garcia, D. M. Prett, and M. Morari, "Model predictive control: theory and practice-a survey," *Automatica*, vol. 25, pp. 335–348, 1989.
[12] J. Kober, J. A. Bagnell, and J. Peters, "Reinforcement learning in robotics: A survey," *IJRR*, vol. 32, no. 11, pp. 1238–1274, 2013.
[13] A. Y. Ng *et al.*, "Autonomous inverted helicopter flight via reinforcement learning," in *Experimental Robotics IX*, 2006, pp. 363–372.
[14] J. Kober and J. Peters, "Learning motor primitives for robotics," in *ICRA*, 2009.
[15] 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.
[16] R. Pautrat, K. Chatzilygeroudis, and J.-B. Mouret, "Bayesian Optimization with Automatic Prior Selection for Data-Efficient Direct Policy Search," in *ICRA*, 2018.
[17] K. Chatzilygeroudis and J.-B. Mouret, "Using Parameterized Black-Box Priors to Scale Up Model-Based Policy Search for Robotics," in *ICRA*, 2018.
[18] A. J. Ijspeert, J. Nakanishi, and S. Schaal, "Learning attractor landscapes for learning motor primitives," in *NIPS*, 2003.
[19] P. Abbeel, M. Quigley, and A. Y. Ng, "Using inaccurate models in reinforcement learning," in *ICML*, 2006.

[20] E. Brochu, V. M. Cora, and N. De Freitas, "A tutorial on Bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning," *arXiv preprint arXiv:1012.2599*, 2010.

[21] 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," *Proceedings of the IEEE*, vol. 104, no. 1, pp. 148–175, 2016.

[22] R. S. Sutton, D. A. McAllester, S. P. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation," in *NIPS*, 2000.

[23] N. Kohl and P. Stone, "Policy gradient reinforcement learning for fast quadrupedal locomotion," in *ICRA*, 2004.

[24] D. Silver *et al.*, "Deterministic policy gradient algorithms," in *ICML*, 2014.

[25] T. Degris, M. White, and R. S. Sutton, "Linear off-policy actor-critic," in *ICML*, 2012.

[26] K. Ciosek and S. Whiteson, "Expected Policy Gradients for Reinforcement Learning," *arXiv preprint arXiv:1801.03326*, 2018.

[27] H. Van Seijen, H. Van Hasselt, S. Whiteson, and M. Wiering, "A theoretical and empirical analysis of Expected Sarsa," in *ADPRL*, 2009.

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

[29] 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," in *ICLR*, 2016.

[30] A. Abdolmaleki, R. Lioutikov, J. R. Peters, N. Lau, L. P. Reis, and G. Neumann, "Model-based relative entropy stochastic search," in *NIPS*, 2015.

[31] P. Fidelman and P. Stone, "Learning ball acquisition on a physical robot," in *ISRA*, 2004.

[32] F. Guenter, M. Hersch, S. Calinon, and A. Billard, "Reinforcement learning for imitating constrained reaching movements," *Advanced Robotics*, vol. 21, pp. 1521–1544, 2007.

[33] H. van Hoof, T. Hermans, G. Neumann, and J. Peters, "Learning robot in-hand manipulation with tactile features," in *Humanoids*, 2015, pp. 121–127.

[34] T. Matsubara, S. Hyon, and J. Morimoto, "Learning parametric dynamic movement primitives from multiple demonstrations," *Neural Networks*, vol. 24, no. 5, pp. 493–500, 2011.

[35] S. M. Khansari-Zadeh and A. Billard, "Learning stable nonlinear dynamical systems with gaussian mixture models," *IEEE Transactions on Robotics*, vol. 27, no. 5, pp. 943–957, 2011.

[36] A. Ude, B. Nemec, J. Morimoto, *et al.*, "Trajectory representation by nonlinear scaling of dynamic movement primitives," in *IROS*, 2016.

[37] A. Ude, A. Gams, T. Asfour, and J. Morimoto, "Task-specific generalization of discrete and periodic dynamic movement primitives," *IEEE Transactions on Robotics*, vol. 26, no. 5, pp. 800–815, 2010.

[38] J. Spitz, K. Bouyarmane, S. Ivaldi, and J.-B. Mouret, "Trial-and-Error Learning of Repulsors for Humanoid QP-based Whole-Body Control," in *Humanoids*, 2017.

[39] 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.

[40] A. Ijspeert, J. Nakanishi, P. Pastor, H. Hoffmann, and S. Schaal, "Dynamical Movement Primitives: Learning attractor models for motor behaviors," *Neural Computation*, vol. 25, no. 2, pp. 328–373, 2013.

[41] A. J. Ijspeert, J. Nakanishi, and S. Schaal, "Movement imitation with nonlinear dynamical systems in humanoid robots," in *ICRA*, 2002.

[42] N. Roy and S. Thrun, "Motion planning through policy search," in *IROS*, 2002.

[43] F. Stulp and O. Sigaud, "Policy improvement: Between black-box optimization and episodic reinforcement learning," in *Journées Francophones Planification, Décision, et Apprentissage pour la conduite de systèmes*, 2013.

[44] F. Stulp, E. Theodorou, and S. Schaal, "Reinforcement learning with sequences of motion primitives for robust manipulation," *IEEE Transactions on Robotics*, vol. 28, no. 6, pp. 1360–1370, 2012.

[45] F. Stulp, G. Raiola, *et al.*, "Learning Compact Parameterized Skills with a Single Regression," in *Humanoids*, 2013.

[46] 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," *Artif. Intel.*, vol. 247, pp. 415–439, 2017.

[47] A. Abdolmaleki, B. Price, N. Lau, L. P. Reis, and G. Neumann, "Contextual covariance matrix adaptation evolutionary strategies," in *IJCAI*, 2017.

[48] S. Calinon, "A tutorial on task-parameterized movement learning and retrieval," *Intelligent Service Robotics*, vol. 9, no. 1, pp. 1–29, 2016.

[49] J. Buchli, F. Stulp, E. Theodorou, and S. Schaal, "Learning Variable Impedance Control," *IJRR*, vol. 30, no. 7, pp. 820–833, 2011.

[50] S. Calinon, D. Bruno, and D. G. Caldwell, "A task-parameterized probabilistic model with minimal intervention control," in *ICRA*, 2014.

[51] S. Calinon, P. Kormushev, and D. G. Caldwell, "Compliant skills acquisition and multi-optima policy search with EM-based reinforcement learning," *Robot. Auton. Syst.*, vol. 61, pp. 369–379, 2013.

[52] M. Kalakrishnan, L. Righetti, P. Pastor, and S. Schaal, "Learning force control policies for compliant manipulation," in *IROS*, 2011.

[53] K. Stanley and R. Miikkulainen, "Evolving Neural Networks Through Augmenting Topologies," *Evol. Comput.*, vol. 10, pp. 99–127, 2002.

[54] K. Sims, "Evolving Virtual Creatures," in *SIGGRAPH*, 1994.

[55] J. C. Bongard and R. Pfeifer, "Evolving Complete Agents using Artificial Ontogeny," in *Proc. of Morpho-functional Machines: The New Species*, 2003.

[56] C. Daniel, G. Neumann, O. Kroemer, and J. Peters, "Hierarchical relative entropy policy search," *JMLR*, pp. 1–50, 2016.

[57] M. R. K. Ryan and M. D. Pendrith, "Rl-tops: An architecture for modularity and re-use in reinforcement learning." in *ICML*, 1998, pp. 481–487.

[58] T. Lang, M. Toussaint, and K. Kersting, "Exploration in relational domains for model-based reinforcement learning," *J. Mach. Learn. Res.*, pp. 3725–3768, 2012.

[59] F. Yang, D. Lyu, B. Liu, and S. Gustafson, "Peorl: Integrating symbolic planning and hierarchical reinforcement learning for robust decision-making," in *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18*. International Joint Conferences on Artificial Intelligence Organization, 7 2018, pp. 4860–4866. [Online]. Available: https://doi.org/10.24963/ijcai.2018/675

[60] T. Osa, J. Pajarinen, G. Neumann, J. A. Bagnell, P. Abbeel, J. Peters, *et al.*, "An algorithmic perspective on imitation learning," *Foundations and Trends® in Robotics*, vol. 7, no. 1-2, pp. 1–179, 2018.

[61] B. D. Argall, S. Chernova, M. Veloso, and B. Browning, "A survey of robot learning from demonstration," *Robot. Auton. Syst.*, vol. 57, pp. 469–483, 2009.

[62] A. Billard, S. Calinon, R. Dillmann, and S. Schaal, "Robot programming by demonstration," in *Springer handbook of robotics*. Springer, 2008, pp. 1371–1394.

[63] C. E. Rasmussen and C. K. Williams, *Gaussian processes for machine learning*. MIT press Cambridge, 2006, vol. 1.

[64] P. Hennig and C. J. Schuler, "Entropy search for information-efficient global optimization," *JMLR*, vol. 13, pp. 1809–1837, 2012.

[65] H. J. Kushner, "A new method of locating the maximum point of an arbitrary multipeak curve in the presence of noise," *J. Basic. Eng.*, vol. 86, pp. 97–106, 1964.

[66] N. Srinivas, A. Krause, S. M. Kakade, and M. Seeger, "Gaussian process optimization in the bandit setting: No regret and experimental design," *arXiv preprint arXiv:0912.3995*, 2009.

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

[68] R. Martinez-Cantin, N. de Freitas, A. Doucet, and J. A. Castellanos, "Active Policy Learning for Robot Planning and Exploration under Uncertainty," in *RSS*, 2007.

[69] D. J. Lizotte, T. Wang, M. H. Bowling, and D. Schuurmans, "Automatic gait optimization with gaussian process regression," in *IJCAI*, 2007.

[70] J. Rieffel and J.-B. Mouret, "Adaptive and resilient soft tensegrity robots," *Soft Robotics*, vol. 5, pp. 318–329, 2018.

[71] R. E. Bellman, *Dynamic Programming*. Princeton, NJ: Princeton University Press, 1957.

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

[73] K. Kandasamy, J. Schneider, and B. Póczos, "High dimensional Bayesian optimisation and bandits via additive models," in *ICML*, 2015.

[74] P. Rolland, J. Scarlett, I. Bogunovic, and V. Cevher, "High-Dimensional Bayesian Optimization via Additive Models with Overlapping Groups," *arXiv preprint arXiv:1802.07028*, 2018.

[75] R. Akrour, D. Sorokin, J. Peters, and G. Neumann, "Local bayesian optimization of motor skills," in *ICML*, 2017.

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

[77] 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. Evol. Comput.*, 2017.

[78] J. K. Pugh, L. B. Soros, and K. O. Stanley, "Quality diversity: A new frontier for evolutionary computation," *Frontiers in Robotics and AI*, vol. 3, p. 40, 2016.

[79] 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.

[80] K. Chatzilygeroudis, V. Vassiliades, and J.-B. Mouret, "Reset-free Trial-and-Error Learning for Robot Damage Recovery," *Robot. Auton. Syst.*, vol. 100, pp. 236–250, 2018.

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

[82] ——, "Deep Kernels for Optimizing Locomotion Controllers," in *CoRL*, 2017.

[83] V. T. Inman, H. D. Eberhart, *et al.*, "The major determinants in normal and pathological gait," *JBJS*, vol. 35, no. 3, pp. 543–558, 1953.

[84] N. Hansen and A. Ostermeier, "Completely derandomized self-adaptation in evolution strategies," *Evol. Comput.*, vol. 9, pp. 159–195, 2001.

[85] A. Wilson, A. Fern, and P. Tadepalli, "Using trajectory data to improve bayesian optimization for reinforcement learning," *JMLR*, vol. 15, no. 1, pp. 253–282, 2014.

[86] R. Lober, V. Padois, and O. Sigaud, "Efficient reinforcement learning for humanoid whole-body control," in *Humanoids*, 2016.

[87] J. Salini, V. Padois, and P. Bidaud, "Synthesis of complex humanoid whole-body behavior: a focus on sequencing and tasks transitions," in *ICRA*, 2011.

[88] R. Lober, J. Eljaik, G. Nava, S. Dafarra, F. Romano, D. Pucci, S. Traversaro, F. Nori, O. Sigaud, and V. Padois, "Optimizing task feasibility using model-free policy search and model-based whole-body control," in *ICRA*, 2017.

[89] 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 *ICRA*, 2017.

[90] V. Papaspyros, K. Chatzilygeroudis, V. Vassiliades, and J.-B. Mouret, "Safety-Aware Robot Damage Recovery Using Constrained Bayesian Optimization and Simulated Priors," in *Proc. of the International Workshop "Bayesian Optimization: Black-box Optimization and Beyond" at NIPS*, 2016.

[91] J. R. Gardner *et al.*, "Bayesian Optimization with Inequality Constraints," in *ICML*, 2014.

[92] F. Berkenkamp, A. P. Schoellig, and A. Krause, "Safe Controller Optimization for Quadrotors with Gaussian Processes," in *ICRA*, 2016.

[93] A. S. Polydoros and L. Nalpantidis, "Survey of Model-Based Reinforcement Learning: Applications on Robotics," *Journal of Intelligent & Robotic Systems*, pp. 1–21, 2017.

[94] R. S. Sutton, "Dyna, an integrated architecture for learning, planning, and reacting," *ACM SIGART Bulletin*, vol. 2, no. 4, pp. 160–163, 1991.

[95] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement learning: A survey," *JAIR*, vol. 4, pp. 237–285, 1996.

[96] 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.

[97] S. Levine and P. Abbeel, "Learning neural network policies with guided policy search under unknown dynamics," in *NIPS*, 2014.

[98] P. Parmas, C. E. Rasmussen, J. Peters, and K. Doya, "PIPPS: Flexible Model-Based Policy Search Robust to the Curse of Chaos," in *ICML*, 2018.

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

[100] M. P. Deisenroth and C. E. Rasmussen, "PILCO: A model-based and data-efficient approach to policy search," in *ICML*, 2011.

[101] M. Sugiyama, I. Takeuchi, T. Suzuki, T. Kanamori, H. Hachiya, and D. Okanohara, "Least-squares conditional density estimation," *IEICE Trans. on Information and Systems*, vol. 93, no. 3, pp. 583–594, 2010.

[102] S. Levine, C. Finn, T. Darrell, and P. Abbeel, "End-to-end training of deep visuomotor policies," *JMLR*, vol. 17, no. 39, pp. 1–40, 2016.

[103] V. Kumar, E. Todorov, and S. Levine, "Optimal control with learned local models: Application to dexterous manipulation," in *ICRA*, 2016.

[104] Y. Gal and Z. Ghahramani, "Dropout as a Bayesian approximation: Representing model uncertainty in deep learning," in *ICML*, 2016.

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

[106] J. C. G. Higuera, D. Meger, and G. Dudek, "Synthesizing neural network controllers with probabilistic model based reinforcement learning," *arXiv preprint arXiv:1803.02291*, 2018.

[107] K. Chua, R. Calandra, R. McAllister, and S. Levine, "Deep reinforcement learning in a handful of trials using probabilistic dynamics models," in *NIPS*, 2018.

[108] P. Wawrzynski, "Learning to control a 6-degree-of-freedom walking robot," in *EUROCON*, 2007.

[109] S. Depeweg, J. M. Hernández-Lobato, F. Doshi-Velez, and S. Udluft, "Learning and policy search in stochastic dynamical systems with bayesian neural networks," in *ICLR*, 2017.

[110] ——, "Decomposition of uncertainty in bayesian deep learning for efficient and risk-sensitive learning," in *ICML*, 2018.

[111] A. Doerr *et al.*, "Optimizing long-term predictions for model-based policy search," in *CoRL*, 2017.

[112] A. Y. Ng and M. Jordan, "PEGASUS: a policy search method for large MDPs and POMDPs," in *UAI*, 2000.

[113] B. D. Anderson and J. B. Moore, "Optimal filtering," *Englewood Cliffs*, vol. 21, pp. 22–95, 1979.

[114] S. J. Julier and J. K. Uhlmann, "Unscented filtering and nonlinear estimation," *Proceedings of the IEEE*, vol. 92, pp. 401–422, 2004.

[115] M. P. Deisenroth, C. E. Rasmussen, and D. Fox, "Learning to Control a Low-Cost Manipulator using Data-Efficient Reinforcement Learning," in *RSS*, 2011.

[116] M. P. Deisenroth, R. Calandra, A. Seyfarth, and J. Peters, "Toward fast policy search for learning legged locomotion," in *IROS*, 2012.

[117] Y. Jin and J. Branke, "Evolutionary optimization in uncertain environments - a survey," *IEEE Trans. Evol. Comput.*, vol. 9, pp. 303–317, 2005.

[118] B. L. Miller and D. E. Goldberg, "Genetic algorithms, selection schemes, and the varying effects of noise," *Evol. Comput.*, vol. 4, pp. 113–131, 1996.

[119] S. Tsutsui and A. Ghosh, "Genetic algorithms with a robust solution searching scheme," *IEEE Trans. Evol. Comput.*, vol. 1, pp. 201–208, 1997.

[120] N. Hansen, *The CMA Evolution Strategy: A Comparing Review*. Springer, 2006.

[121] V. Heidrich-Meisner and C. Igel, "Hoeffding and bernstein races for selecting policies in evolutionary direct policy search," in *ICML*, 2009.

[122] 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. Evol. Comput.*, vol. 13, pp. 180–197, 2009.

[123] N. Hansen, "Benchmarking a BI-population CMA-ES on the BBOB-2009 noisy testbed," in *GECCO*, 2009.

[124] A. Auger and N. Hansen, "A restart cma evolution strategy with increasing population size," in *CEC*, 2005, pp. 1769–1776.

[125] 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 *ICRA*, 2014.

[126] M. P. Deisenroth, P. Englert, J. Peters, and D. Fox, "Multi-task policy search for robotics," in *ICRA*, 2014.

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

[128] M. Saveriano, Y. Yin, P. Falco, and D. Lee, "Data-efficient control policy search using residual dynamics learning," in *IROS*, 2017.

[129] T. Wu and J. Movellan, "Semi-parametric Gaussian process for robot system identification," in *IROS*, 2012.

[130] J. Ko, D. J. Klein, D. Fox, and D. Haehnel, "Gaussian processes and reinforcement learning for identification and control of an autonomous blimp," in *ICRA*, 2007.

[131] M. Spong and D. Block, "The pendubot: A mechatronic system for control research and education," in *Proc IEEE Conf Decis Control*, 1995.

[132] S. Zhu, A. Kimmel, K. E. Bekris, and A. Boularias, "Fast Model Identification via Physics Engines for Data-Efficient Policy Search," in *IJCAI*, 2018.

[133] J. Bongard, V. Zykov, and H. Lipson, "Resilient machines through continuous self-modeling," *Science*, vol. 314, pp. 1118–1121, 2006.

[134] B. Siciliano and O. Khatib, *Springer handbook of robotics*, 2nd ed. Springer, 2016.

[135] W. Montgomery, A. Ajay, C. Finn, P. Abbeel, and S. Levine, "Reset-free guided policy search: efficient deep reinforcement learning with stochastic initial states," in *ICRA*, 2017.

[136] S. Levine and V. Koltun, "Guided policy search," in *ICML*, 2013.

[137] S. Koos, J.-B. Mouret, and S. Doncieux, "The transferability approach: Crossing the reality gap in evolutionary robotics," *IEEE Trans. Evol. Comput.*, vol. 17, pp. 122–145, 2013.

[138] S. Koos, A. Cully, and J.-B. Mouret, "Fast damage recovery in robotics with the t-resilience algorithm," *IJRR*, vol. 32, pp. 1700–1723, 2013.

[139] S. Koos and J.-B. Mouret, "Online discovery of locomotion modes for wheel-legged hybrid robots: A transferability-based approach," in *CLAWAR*, 2012.

[140] F. Sadeghi and S. Levine, "CAD2RL: Real single-image flight without a single real image," in *RSS*, 2017.

[141] S. James, A. J. Davison, and E. Johns, "Transferring end-to-end visuomotor control from simulation to real world for a multi-stage task," in *CoRL*, 2017.

[142] S. James, M. Bloesch, and A. J. Davison, "Task-Embedded Control Networks for Few-Shot Imitation Learning," in *CoRL*, 2018.

[143] S. James, P. Wohlhart, M. Kalakrishnan, D. Kalashnikov, A. Irpan, J. Ibarz, S. Levine, R. Hadsell, and K. Bousmalis, "Sim-to-Real via Sim-to-Sim: Data-efficient Robotic Grasping via Randomized-to-Canonical Adaptation Networks," in *CVPR*, 2019.

[144] Y. Chebotar, A. Handa, V. Makoviychuk, M. Macklin, J. Issac, N. Ratliff, and D. Fox, "Closing the Sim-to-Real Loop: Adapting Simulation Randomization with Real World Experience," in *ICRA*, 2018.

[145] J. Tan, T. Zhang, E. Coumans, A. Iscen, Y. Bai, D. Hafner, S. Bohez, and V. Vanhoucke, "Sim-to-Real: Learning Agile Locomotion For Quadruped Robots," in *RSS*, 2018.

[146] X. B. Peng, M. Andrychowicz, W. Zaremba, and P. Abbeel, "Sim-to-real transfer of robotic control with dynamics randomization," in *ICRA*, 2018.

[147] M. Andrychowicz, F. Wolski, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, O. P. Abbeel, and W. Zaremba, "Hindsight experience replay," in *NIPS*, 2017.

[148] M. Feurer, J. T. Springenberg, and F. Hutter, "Initializing bayesian hyperparameter optimization via meta-learning," in *AAAI*, 2015.

[149] C. Finn, P. Abbeel, and S. Levine, "Model-agnostic meta-learning for fast adaptation of deep networks," in *ICML*, 2017.

[150] I. Clavera, A. Nagabandi, R. S. Fearing, P. Abbeel, S. Levine, and C. Finn, "Learning to Adapt in Dynamic, Real-World Environments through Meta-Reinforcement Learning," in *ICLR*, 2019.

[151] S. Sæmundsson, K. Hofmann, and M. P. Deisenroth, "Meta Reinforcement Learning with Latent Variable Gaussian Processes," in *UAI*, 2018.

[152] J. Hwangbo, J. Lee, A. Dosovitskiy, D. Bellicoso, V. Tsounis, V. Koltun, and M. Hutter, "Learning agile and dynamic motor skills for legged robots," *Science Robotics*, vol. 4, no. 26, 2019.

[153] N. G. Tsagarakis, G. Metta, *et al.*, "icub: the design and realization of an open humanoid platform for cognitive and neuroscience research," *Advanced Robotics*, vol. 21, no. 10, pp. 1151–1175, 2007.

[154] P. Maiolino, M. Maggiali, G. Cannata, G. Metta, and L. Natale, "A flexible and robust large scale capacitive tactile system for robots," *IEEE Sensors Journal*, vol. 13, no. 10, pp. 3910–3917, 2013.

[155] T. Dean and K. Kanazawa, "A model for reasoning about persistence and causation," *Comput. Intell.*, vol. 5, pp. 142–150, 1989.

[156] C. Boutilier, R. Dearden, and M. Goldszmidt, "Stochastic dynamic programming with factored representations," *Artif. Intel.*, vol. 121, pp. 49–107, 2000.

[157] A. J. Ijspeert, "Central pattern generators for locomotion control in animals and robots: a review," *Neural Netw.*, vol. 21, pp. 642–653, 2008.

[158] V. C. Kumar, S. Ha, and K. Yamane, "Improving Model-Based Balance Controllers using Reinforcement Learning and Adaptive Sampling," in *ICRA*, 2018.

[159] R. Jonschkowski and O. Brock, "Learning state representations with robotic priors," *Autonomous Robots*, vol. 39, no. 3, pp. 407–428, 2015.

[160] T. Lesort, N. Díaz-Rodríguez, J.-F. Goudou, and D. Filliat, "State representation learning for control: An overview," *Neural Netw.*, vol. 108, pp. 379–392, 2018.

[161] J. Oh, X. Guo, H. Lee, R. L. Lewis, and S. Singh, "Action-conditional video prediction using deep networks in atari games," in *NIPS*, 2015.

[162] D. Ha and J. Schmidhuber, "World models," *arXiv preprint arXiv:1803.10122*, 2018.

[163] J.-A. M. Assael, N. Wahlström, T. B. Schön, and M. P. Deisenroth, "Data-efficient learning of feedback policies from image pixels using deep dynamical models," *NIPS Deep RL Workshop*, 2015.

[164] S. A. Eslami, D. J. Rezende, F. Besse, F. Viola, A. S. Morcos, M. Garnelo, A. Ruderman, A. A. Rusu, I. Danihelka, K. Gregor, *et al.*, "Neural scene representation and rendering," *Science*, vol. 360, no. 6394, pp. 1204–1210, 2018.

[165] J. A. Musick and C. J. Limpus, "Habitat utilization and migration in juvenile sea turtles," *The biology of sea turtles*, vol. 1, pp. 137–163, 1997.

[166] T. Lesort, M. Seurin, X. Li, N. D. Rodríguez, and D. Filliat, "Unsupervised state representation learning with robotic priors: a robustness benchmark," *arXiv preprint arXiv:1709.05185*, 2017.

[167] T.-H. Pham, G. De Magistris, and R. Tachibana, "OptLayer-Practical Constrained Optimization for Deep Reinforcement Learning in the Real World," in *ICRA*, 2018.

[168] T. Haarnoja, V. Pong, A. Zhou, M. Dalal, P. Abbeel, and S. Levine, "Composable Deep Reinforcement Learning for Robotic Manipulation," in *ICRA*, 2018.

[169] T. Pinville, S. Koos, J.-B. Mouret, and S. Doncieux, "How to promote generalisation in evolutionary robotics: the progab approach," in *GECCO*, 2011.

[170] B. C. Da Silva, G. Konidaris, and A. G. Barto, "Learning parameterized skills," in *ICML*, 2012.

[171] J. Kober, A. Wilhelm, E. Oztop, and J. Peters, "Reinforcement learning to adjust parametrized motor primitives to new situations," *Autonomous Robots*, vol. 33, no. 4, pp. 361–379, 2012.

[172] A. Fabisch and J. H. Metzen, "Active contextual policy search," *JMLR*, vol. 15, no. 1, pp. 3371–3399, 2014.

[173] T. Schaul, D. Horgan, K. Gregor, and D. Silver, "Universal value function approximators," in *ICML*, 2015.

[174] P. Karkus, A. Kupcsik, D. Hsu, and W. S. Lee, "Factored Contextual Policy Search with Bayesian Optimization," in *BayesOpt'16: Proceedings of the International Workshop "Bayesian Optimization: Black-box Optimization and Beyond" at NIPS*, 2016.

[175] S. Ha and C. K. Liu, "Evolutionary optimization for parameterized whole-body dynamic motor skills," in *ICRA*, 2016.

[176] Y. Zhu, R. Mottaghi, E. Kolve, J. J. Lim, A. Gupta, L. Fei-Fei, and A. Farhadi, "Target-driven visual navigation in indoor scenes using deep reinforcement learning," in *ICRA*, 2017.

[177] P. Rauber, A. Ummadisingu, F. Mutz, and J. Schmidhuber, "Hindsight policy gradients," *arXiv preprint arXiv:1711.06006*, 2017.

[178] D. Ghosh, A. Singh, A. Rajeswaran, V. Kumar, and S. Levine, "Divide-and-conquer reinforcement learning," in *ICLR*, 2018.

[179] D. J. Mankowitz, A. Žídek, A. Barreto, D. Horgan, M. Hessel, J. Quan, J. Oh, H. van Hasselt, D. Silver, and T. Schaul, "Unicorn: Continual learning with a universal, off-policy agent," *arXiv preprint arXiv:1802.08294*, 2018.

[180] M. P. Deisenroth, P. Englert, J. Peters, and D. Fox, "Multi-task policy search for robotics," in *ICRA*, 2014.

[181] S. Paul, K. Chatzilygeroudis, K. Ciosek, J.-B. Mouret, M. A. Osborne, and S. Whiteson, "Alternating Optimisation and Quadrature for Robust Control," in *AAAI*, 2018.

[182] V. Vassiliades and C. Christodoulou, "Toward nonlinear local reinforcement learning rules through neuroevolution," *Neural Computation*, vol. 25, no. 11, pp. 3020–3043, 2013.

[183] J. X. Wang, Z. Kurth-Nelson, D. Tirumala, H. Soyer, J. Z. Leibo, R. Munos, C. Blundell, D. Kumaran, and M. Botvinick, "Learning to reinforcement learn," *arXiv preprint arXiv:1611.05763*, 2016.

[184] J. Harrison, A. Sharma, R. Calandra, and M. Pavone, "Control Adaptation via Meta-Learning Dynamics," in *Workshop on Meta-Learning at NeurIPS 2018*, 2018.

[185] W. Yu, C. K. Liu, and G. Turk, "Preparing for the unknown: Learning a universal policy with online system identification," in *RSS*, 2017.

[186] A. Rajeswaran, S. Ghotra, B. Ravindran, and S. Levine, "Epopt: Learning robust neural network policies using model ensembles," *arXiv preprint arXiv:1610.01283*, 2016.

[187] R. E. Kalman, "A new approach to linear filtering and prediction problems," *J. Basic. Eng.*, vol. 82, pp. 35–45, 1960.

[188] D. Mayne, "A second-order gradient method for determining optimal trajectories of non-linear discrete-time systems," *International Journal of Control*, vol. 3, no. 1, pp. 85–95, 1966.

[189] D. H. Jacobson and D. Q. Mayne, *Differential dynamic programming*. Elsevier, 1970.

[190] E. Todorov and W. Li, "A generalized iterative lqg method for locally-optimal feedback control of constrained nonlinear stochastic systems," in *American Control Conference*, 2005.

[191] J. Koenemann, A. Del Prete, Y. Tassa, E. Todorov, O. Stasse, M. Bennewitz, and N. Mansard, "Whole-body model-predictive control applied to the HRP-2 humanoid," in *IROS*, 2015.

[192] K. L. Moore, M. Dahleh, and S. Bhattacharyya, "Iterative learning control: A survey and new results," *Journal of Field Robotics*, vol. 9, no. 5, pp. 563–594, 1992.

[193] D. A. Bristow, M. Tharayil, and A. G. Alleyne, "A survey of iterative learning control," *IEEE Control Systems*, vol. 26, pp. 96–114, 2006.
[194] K. S. Lee, I.-S. Chin, H. J. Lee, and J. H. Lee, "Model predictive control technique combined with iterative learning for batch processes," *AIChE Journal*, vol. 45, no. 10, pp. 2175–2187, 1999.
[195] J. H. Lee, K. S. Lee, and W. C. Kim, "Model-based iterative learning control with a quadratic criterion for time-varying linear systems," *Automatica*, vol. 36, no. 5, pp. 641–657, 2000.
[196] Y. Wang, D. Zhou, and F. Gao, "Iterative learning model predictive control for multi-phase batch processes," *Journal of Process Control*, vol. 18, no. 6, pp. 543–557, 2008.
[197] T. Zhang, G. Kahn, S. Levine, and P. Abbeel, "Learning deep control policies for autonomous aerial vehicles with mpc-guided policy search," in *ICRA*, 2016.
[198] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *IJRR*, vol. 30, no. 7, pp. 846–894, 2011.
[199] C. B. Browne, E. Powley, D. Whitehouse, S. M. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton, "A survey of monte carlo tree search methods," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 4, pp. 1–43, 2012.
[200] M. Duarte, J. Gomes, S. M. Oliveira, and A. L. Christensen, "Evolution of repertoire-based control for robots with complex locomotor systems," *IEEE Trans. Evol. Comput.*, 2017.
[201] D. Clever, M. Harant, K. Mombaur, M. Naveau, O. Stasse, and D. Endres, "Cocomopl: A novel approach for humanoid walking generation combining optimal control, movement primitives and learning and its transfer to the real robot hrp-2," *IEEE Robotics and Automation Letters*, vol. 2, no. 2, pp. 977–984, 2017.

Konstantinos Chatzilygeroudis is currently a post-doctoral fellow at the LASA team at EPFL. He obtained a B.Sc. and M.Sc. in Computer Science and Engineering from the University of Patras in 2014, and a Ph.D. in Robotics and Machine Learning from Inria Nancy-Grand Est (France) and the University of Lorraine. His research interests lie in the area of artificial intelligence and focus on reinforcement learning and fast robot adaptation.
Website: http://costashatz.github.io

Vassilis Vassiliades received the Ph.D. degree from the University of Cyprus (2015). He is currently a team leader at the Research Centre on Interactive Media, Smart Systems and Emerging Technologies (RISE) in Cyprus. He held post-doctoral and research engineer positions at Inria Nancy, France (2015-2018), and research associate positions at the University of Cyprus (2015-2019) and RISE (2019). His research focuses on reinforcement learning, neural networks and evolutionary computation.

Freek Stulp received his doctorate degree in Computer Science from the Technische Universität München in 2007. He is currently the head of the department of Cognitive Robotics at the Institute of Robotics and Mechatronics at the German Aerospace Center (DLR). Previously, he was an assistant professor at the École Nationale Supérieure de Techniques Avancées (ENSTA-ParisTech). He currently serves as an Associate Editor in IEEE Transactions on Robotics.

Sylvain Calinon received the Ph.D. degree from the Ecole Polytechnique Fédérale de Lausanne (EPFL) in 2007. He is a Senior Researcher at the Idiap Research Institute, and a Lecturer at the EPFL. From 2009 to 2014, he was a Team Leader at the Department of Advanced Robotics, Italian Institute of Technology. From 2007 to 2009, he was a Postdoc at EPFL. He currently serves as an Associate Editor in IEEE Transactions on Robotics and IEEE Robotics and Automation Letters. Website: http://calinon.ch

Jean-Baptiste Mouret received the Ph.D. degree in 2008 from the Pierre and Marie Curie University (Paris, France). He is currently a senior researcher ("Directeur de recherche") at Inria, the French research institute dedicated to computer science and mathematics; from 2009 to 2015, he was an assistant professor ("maître de conférences") at the Pierre and Marie Curie University. His work was recently featured on the cover of Nature (Cully et al., 2015) and it received several national and international scientific awards, including the "Prix La Recherche 2016" and the "Distinguished Young Investigator in Artificial Life 2017".
Website: http://members.loria.fr/jbmouret