UNIVERSITÉ DE LORRAINE

École doctorale IAEM Lorraine

Micro-Data Reinforcement Learning for Adaptive Robots

THÈSE

présentée et soutenue publiquement le 14 December 2018

pour l'obtention du

Doctorat de l'Université de Lorraine

(mention informatique)

par

Konstantinos Chatzilygeroudis

Composition du jury

Rapporteurs :	Pierre-Yves OUDEYER	Directeur de recherche Inria Bordeaux Sud-Ouest, France
	Yiannis DEMIRIS	Professor Imperial College London, UK
Examinateurs :	Aude BILLARD	Professor EPFL, Switzerland
	Alain DUTECH	Chargé de recherche Inria, CNRS, Université de Lorraine, France
	Verena V. HAFNER	Professor Humboldt-Universität zu Berlin, Germany
Directeur :	Jean-Baptiste MOURET	Directeur de recherche Inria, CNRS, Université de Lorraine, France

Laboratoire Lorrain de Recherche en Informatique et ses Applications — UMR 7503

European Research Council
Established by the European Commission

English Abstract

Robots have to face the real world, in which trying something might take seconds, hours, or even days. Unfortunately, the current state-of-the-art reinforcement learning algorithms (*e.g.*, deep reinforcement learning) require big interaction times to find effective policies. In this thesis, we explored approaches that tackle the challenge of learning by trial-and-error in a few minutes on physical robots. We call this challenge "micro-data reinforcement learning".

In our first contribution, we introduced a novel learning algorithm called "Reset-free Trial-and-Error" that allows complex robots to quickly recover from unknown circumstances (*e.g.*, damages or different terrain) while completing their tasks and taking the environment into account; in particular, a physical damaged hexapod robot recovered most of its locomotion abilities in an environment with obstacles, and without any human intervention.

In our second contribution, we introduced a novel model-based reinforcement learning algorithm, called Black-DROPS that: (1) does not impose any constraint on the reward function or the policy (they are treated as black-boxes), (2) is as data-efficient as the state-of-the-art algorithm for data-efficient RL in robotics, and (3) is as fast (or faster) than analytical approaches when several cores are available. We additionally proposed Multi-DEX, a model-based policy search approach, that takes inspiration from novelty-based ideas and effectively solved several sparse reward scenarios.

In our third contribution, we introduced a new model learning procedure in Black-DROPS (we call it GP-MI) that leverages parameterized black-box priors to scale up to high-dimensional systems; for instance, it found high-performing walking policies for a physical damaged hexapod robot (48D state and 18D action space) in less than 1 minute of interaction time.

Finally, in the last part of the thesis, we explored a few ideas on how to incorporate safety constraints, robustness and leverage multiple priors in Bayesian optimization in order to tackle the micro-data reinforcement learning challenge.

Throughout this thesis, our goal was to design algorithms that work on physical robots, and not only in simulation. Consequently, all the proposed approaches have been evaluated on at least one physical robot. Overall, this thesis aimed at providing methods and algorithms that will allow physical robots to be more autonomous and be able to learn in a handful of trials.

French Abstract

Les robots opèrent dans le monde réel, dans lequel essayer quelque chose prend des secondes, des heures ou même des jours. Malheureusement, les algorithmes d'apprentissage par renforcement actuels (par exemple, les algorithmes de "deep reinforcement learning") nécessitent de longues périodes d'interaction pour trouver des politiques efficaces. Dans ce thèse, nous avons exploré des algorithms qui abordent le défi de l'apprentissage par essai et erreur en quelques minutes sur des robots physiques. Nous appelons ce défi "Apprentissage par renforcement micro-data".

Dans notre première contribution, nous avons proposé un nouvel algorithme d'apprentissage appelé "Reset-free Trial-and-Error" qui permet aux robots complexes de s'adapter rapidement dans des circonstances inconnues (par exemple, des dommages ou un terrain différent) tout en accomplissant leurs tâches et en prenant en compte l'environnement; en particulier, un robot hexapode endommagé a retrouvé la plupart de ses capacités de locomotion dans un environnement avec des obstacles, et sans aucune intervention humaine.

Dans notre deuxième contribution, nous avons proposé un nouvel algorithme de recherche de politique "basé modèle", appelé Black-DROPS, qui: (1) n'impose aucune contrainte á la fonction de récompense ou á la politique, (2) est aussi efficace que les algorithmes de l'état de l'art, et (3) est aussi rapide (ou plus rapide) que les approches analytiques lorsque plusieurs processeurs sont disponibles. Nous avons aussi proposé Multi-DEX, une extension qui s'inspire de l'algorithme "Novelty Search" et permet de résoudre plusieurs scénarios où les récompenses sont rares.

Dans notre troisième contribution, nous avons introduit une nouvelle procédure d'apprentissage du modèle dans Black-DROPS qui exploite un simulateur paramétré pour permettre d'apprendre des politiques sur des systèmes avec des espaces d'état de grande taille; par exemple, cet extension de Black-DROPS a trouvé des politiques de marche performantes pour un robot hexapode (espace d'état 48D et d'action 18D) en moins d'une minute de temps d'interaction.

Enfin, dans la dernière partie de la thèse, nous avons exploré quelques idées comment intégrer les contraintes de sécurité, améliorer la robustesse et tirer parti des multiple a priori en optimisation bayésienne.

A travers l'ensemble de cette thèse, notre objectif était de concevoir des algorithmes qui fonctionnent sur des robots physiques, et pas seulement en simulation. Par conséquent, tous les approches proposées ont été évaluées sur au moins un robot physique. Dans l'ensemble, cette thèse propose des

méthodes et des algorithmes qui permettre aux robots physiques d'être plus autonomes et de pouvoir apprendre en poignée d'essais.

Acknowledgements

The last three years and this thesis would not have been the same without the help and support of many people that I would like to warmly thank with these few lines.

First of all, from the bottom of my heart, I would like to thank my dear wife Eleni, for all the moments we shared together through the ups and downs of this three-year adventure. She has been extremely patient during the long evenings, nights and, sometimes, weekends that I spent to finish a paper, a long robotic experiment or simply some code. She has been my highest support during all these 3 years and the manuscript that you are currently reading would not have been the same without her.

Of course, this thesis would not have been of the same quality without the help by my supervisor, Jean-Baptiste Mouret. His supervision was discreet and tight at the same time. He really believed in me and my abilities, and gave me enough freedom to explore my own ideas. At the same time, he always kept a watching eye to prevent me from choosing dangerous roads that could hinder my route. His expectations, patience and useful advices have deeply shaped my scientific researching abilities. He taught me almost everything I know in scientific research and I am really proud of being one of his PhD students.

Within the ResiBots team and the LARSEN lab in general, I had the chance to collaborate and discuss with several excellent researchers and people. The members of the team were always very responsive, caring, friendly and I really enjoyed conversing with them. All the people of the LARSEN lab embraced me and Eleni warmly, and we felt from the first moment as a part of a big caring family. We are deeply grateful to all of them. I would like to thank some people in particular that I had the luck to meet and spend some more time with them (excluding the members of the ResiBots team that have their own dedicated thanks): Serena Ivaldi, Francis Colas, Kazuya Otani, Valerio Modugno, Pauline Maurice, Oriane Dermy, Sebastian Marichal, Olivier Buffet, Adrien Malaisé, Iñaki Fernández Pérez, Adrian Bourgaud, and Luigi Penco.

I would also like to thank all the members of the "ResiBots" team. Several interns, post-docs and collaborators passed, and we always laughed, joked and made a lot of pranks. In particular, I would like to thank Vassilis Vassiliades

for the long, constructive discussions and for the fellow researcher and friend that he has been, and still is, to me. This thesis surely would not have been the same without him. Additionally, I was very lucky to collaborate, discuss and spend some time with the excellent young researcher and student Rémi Pautrat. I gained a lot from this collaboration. I would also like to thank Vaios Papaspyros, Rituraj Kaushik, Federico Allocati, Roberto Rama, Jonathan Spitz, Dorian Goepp, Lucien Renaud, Brice Clement, Vladislav Tempez, Debaleena Misra and Kapil Sawant for all the nice discussions (scientific and not) that we had over the time we spent together and for our excellent collaboration.

I also had the chance to collaborate with some excellent researchers and I am really grateful to Shimon Whiteson, Supratik Paul, Sylvain Calinon and Freek Stulp for the wonderful collaborations that we had.

I was very fortunate to meet a lot of wonderful people in Nancy and to make some lifelong friends. Many thanks go to the three couples that we spent numerous nights drinking, laughing, discussing, sharing personal experiences and eating: Lazaros Vozikis-Chrysa Papantoniou, Nikolas Chrysikos-Lydia Schneider, Alexandros Petrelis-Maria Stathopoulou[1]. I would also like to thank the president of the association "Maison FrancoHellénique Lorraine", Katerina Karagianni, her wonderful husband Thierry Caël and their adorable daughters, Ioanna and Marina, for making us feel like home and always being there for us. I would also like to thank all the members of the board of the association, and in particular, Bernard and Orthodoxia Salomon for always being helpful and available. Finally, I would like to thank Dimitris Chatziathanasiou, Vitalis Ntombrougidis, Vassilis Vassiliades and Marianna Gregoriou, Annabelle Chapron and Nicolas Vicaire, Marianthi Elmaloglou, Dimitris Meimaroglou and Maria Prokopidou (and their wonderful kids, Eleni and Ioannis), Antonis Keremloglou and Marina Kotsani (and their little baby Symeon) for all the nice times that we spent together.

I would also like to thank my whole family for their endless support, love and encouragement: my parents (Ioannis and Maria), my sisters (Marianna, Theodora, Evangelia, Georgia), my brother (Gerasimos), my parents-in-law (Dimitrios and Giannoula) my brothers-in-law (Vassilis, Nikos, Michalis), my sister-in-law (Ourania) and my adorable nephews and nieces (Dimitrios, Ioanna and Anastasia).

I would also like to thank the members of the jury of my PhD defense for accepting to be part of the jury. It is truly an honor to be reviewed by such experienced and respected researchers. Finally, I would like also to thank the European Research Council and the University of Lorraine, who gave me the opportunity to pursue this PhD thesis.

[1]The order was decided by a random number generator because I could not put them in order!

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Aristotle may have been the first to describe how automated mechanical statues could replace slaves and reduce the burden of everyday labor[1]. Since then, numerous advances in physics, mechanical engineering, computer science and mathematics have allowed the wide deployment of automated systems and robots in our everyday life and especially inside factories. For example, FANUC has been operating a "lights out" factory for robots since 2001 (Null and Caulfield, 2003) (a "lights out" factory is one where there are no light, no humans and only robots operate inside it), iRobot has sold more than 8 million robot vacuums, and Amazon currently has more than 10 thousand autonomous mobile robots inside their semi-autonomous warehouses.

Although these robots operate autonomously, they still require humans to specify their tasks and supervise them. Ideally, we would imagine fully autonomous robots operating in various conditions and interacting with humans and the environment. Robots could operate in homes helping with the chores, learning and improving over time. Robots could autonomously design rescue plans and operate in post-disaster sites. Robots could also assist people working in the elderly services by providing physical assistance to elders. Autonomous cars could reduce the number of accidents and minimize the commute time. Overall, we currently have machines that can fulfill specific tasks very well, but none of the currently deployed robots are capable to adapt to truly unforeseen events and improve over time with experience.

Traditionally, robots have been studied under the rigid body dynamics theory and more specifically as chains (either open or closed) of rigid bodies (Murray,

[1]In his *Politics* (322BC, book 1, part 4), he says: *There is only one condition in which we can imagine managers not needing subordinates, and masters not needing slaves. This condition would be that each instrument could do its own work, at the word of command or by intelligent anticipation, like the statues of Daedalus or the tripods made by Hephaestus, of which Homer relates that "Of their own motion they entered the conclave of Gods on Olympus", as if a shuttle should weave of itself, and a plectrum should do its own harp playing.*
Original: εἰ γὰρ ἠδύνατο ἕκαστον τῶν ὀργάνων κελευσθὲν ἢ προαισθανόμενον ἀποτελεῖν τὸ αὑτοῦ ἔργον, ὥσπερ τὰ Δαιδάλου φασὶν ἢ τοὺς τοῦ Ἡφαίστου τρίποδας, οὓς φησιν ὁ ποιητὴς αὐτομάτους θεῖον δύεσθαι ἀγῶνα, οὕτως αἱ κερκίδες ἐκέρκιζον αὐταὶ καὶ τὰ πλῆκτρα ἐκιθάριζεν, οὐδὲν ἂν ἔδει οὔτε τοῖς ἀρχιτέκτοσιν ὑπηρετῶν οὔτε τοῖς δεσπόταις δούλων.

2017; Lynch and Park, 2017). This mathematical framework in conjunction with advances in materials and actuators have allowed the development of reliable and robust control algorithms with impressive results (Rawlings and Mayne, 2009; Krstic et al., 1995; Garcia et al., 1989; Åström, 2012). For instance, modern manipulators can be controlled at 1KHz with millimeter precision (Kyriakopoulos and Saridis, 1988; Bischoff et al., 2011), humanoid robots can now walk reliably on flat terrain (Sellaouti et al., 2006; Mansard et al., 2009) and even robustly perform choreographies under perturbations (Padois et al., 2017; Nori et al., 2015). A few more noteworthy examples are the robots of Boston Dynamics (a US robotics company), like Big Dog (Raibert et al., 2008) and Atlas (Nelson et al., 2012; DARPA, 2013), that are able to showcase locomotion abilities that are very life-like and robust to multiple real-world terrains like snow, grass and rocky roads in forests.

Although these approaches have provided very impressive results and could serve as the basis for more complicated or alternative approaches, the task specifications are usually hard-coded by the programmers or the designers; usually this is described in joint or end-effector acceleration, velocity or position profiles (*i.e.*, joint or end-effector trajectories) (Kyriakopoulos and Saridis, 1988). In addition, these approaches usually assume perfect knowledge of the system dynamics or very high frequency control loops that may be hard to acquire in practice. Overall, current robots are designed to achieve high-performance on specific tasks, but fail to perform when something unexpected happens (Atkeson et al., 2016). In other words, the currently deployed robots are not designed to adapt to unforeseen situations. This realization yields the need for alternative approaches for controlling robots inside a constantly changing world.

Getting inspiration from how animals and humans think and act, robots could also *learn from experience and improve their skills over time*. Learning by trial-and-error is one of the main challenges of Artificial Intelligence (AI) since its beginnings (Russell and Norvig, 2016). Robotics and AI have a long standing, relationship with some notable results. For example, in 1984, Shakey the robot could autonomously navigate an indoor building and interact with voice commands (Nilsson, 1984); Ng et al. (2006) were able to use demonstrations from experts and learn models of the dynamics in order to successfully learn how to perform dynamic and complex maneuvers with a physical helicopter; Calinon et al. (2007) were able to use a few demonstrations (*i.e.*, 4 to 7) from an expert in order to teach a humanoid robot to perform some simple manipulation tasks, like moving a chess pawn to a specified location.

The long-term vision of learning in robotics is to create autonomous and intelligent robots that can adapt in various situations, learn from their mistakes (that is, by trial-and-error), and require no human supervision. This is usually referred to as *General Artificial Intelligence* (GAI) (Legg and Hutter, 2007) and has been one of the main long-term goals of all AI research. In practice, GAI has mostly been studied in the field of computer games with a few noteworthy results. The goal here is to create an algorithm that can learn how to play multiple and different games without any human intervention or reprogramming. For example, PathNet (Fernando et al., 2017) demonstrates transfer learning

capabilities between Atari games by using an evolutionary algorithm in order to select which parts of a neural network should be used for learning new tasks, and Elastic Weight Consolidation (Kirkpatrick et al., 2017) can learn multiple Atari games sequentially without catastrophic forgetting by protecting weights that are important for the previously learned tasks. In robotics, GAI has been mostly studied either theoretically or on simple tasks within the areas of evolutionary and developmental robotics (Oudeyer et al., 2007; Moulin-Frier and Oudeyer, 2013; Schmidhuber, 2006).

Within the field of AI, Machine Learning (ML) (Michalski et al., 2013) has provided the most successful algorithms towards solving the challenge of GAI. ML uses statistical methods to enable computer systems to learn by trial-and-error, that is, to improve with more data without being explicitly programmed. We can split the ML algorithms into three main categories: (1) supervised learning (Russell and Norvig, 2016), (2) reinforcement learning (RL) (Sutton and Barto, 1998), and (3) unsupervised learning. In supervised learning, the system is presented with labeled samples (*i.e.*, input with desired outputs given by an *oracle*) and the task is to learn a mapping (*e.g.*, a function) from the input space to the output space. In reinforcement learning, the agent is given rewards (or punishments) as a feedback to its actions (and current state) in a possibly dynamic environment. In other words, the agent receives reinforcement signals when the actions it takes help towards solving the desired task(s). In unsupervised learning, no labels or reward signals are given to the system and the system has to discover the underlying or hidden structure of the data (*e.g.*, clustering).

There is currently a renewed interest in machine learning and reinforcement learning thanks to recent advances in deep learning (LeCun et al., 2015). For example, deep convolutional neural networks have achieved extraordinary results in detection, segmentation and recognition of objects and regions in images (Vaillant et al., 1994; Lawrence et al., 1997; CireşAn et al., 2012; Turaga et al., 2010), especially in face recognition (Garcia and Delakis, 2004; Taigman et al., 2014), and Deep RL agents can now learn to play many of the Atari 2600 games directly from pixels (Mnih et al., 2015, 2016), that is, without explicit feature engineering, and beat the world's best players at Go and chess with minimal human knowledge (Silver et al., 2017b,a).

Unfortunately, these impressive results are difficult to transfer to robotics because the algorithms behind them are highly data-hungry: 4.4 million labeled faces were required by DeepFace to achieve the reported results (Taigman et al., 2014), 4.8 million games were required to learn to play Go from scratch (Silver et al., 2017b), 38 days of play (real time) for Atari 2600 games (Mnih et al., 2015), and, for example, about 100 hours of simulation time (much more for real time) for a 9-DOF mannequin that learns to walk (Heess et al., 2017). 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.

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.

The above requirement stems from the fact that robots are not only intelligent agents, but they also have *physical bodies that shape the way they act* (Pfeifer and Bongard, 2006). This essentially means that when a robot is trying to complete a task, it is not an abstract agent, but rather a physical agent that interacts with a specific body with the environment. Rodney Brooks describes approaches that are based on this observation as *"Nouvelle AI"* or *"Physically Grounded Methods"* (Brooks, 1990), while others refer to them as *"Embodied Cognition"*. Pfeifer and Bongard (Pfeifer and Bongard, 2006) go even further and make the statement that *"intelligence requires a body"*, which implies that a disembodied agent (*e.g.*, a computer program) cannot be intelligent.

In this thesis, we consider that robots, as embodied systems, are subject to the laws of physics (*e.g.*, gravity, friction, energy supply, etc.) and thus cannot be considered as abstract intelligent agents. Consequently, it is of extreme importance to *minimize the interaction time between the robot and the environment* when using learning algorithms for robot control, as robots are bounded to their physical capabilities and limitations: for example, a mechanical motor has only a limited number of cycles before it dies and thus we cannot use the robot infinitely during the learning procedure.

By analogy with the word "big-data", we refer to the challenge of learning by trial-and-error in a few minutes as "micro-data reinforcement learning" (Mouret, 2016). This concept is close to "data-efficient reinforcement learning" (Deisenroth et al., 2015, 2013), 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"[2]. 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 them. 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[3]? Importantly, a micro-data algorithm might reduce the number of trials by

[2]In some rare cases, a process can be "optimally efficient".

[3]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 manuscript to "a few minutes", a "a few trials" or a "handful of trials". The commonly used word "big-data" has a similar "fuzzy" limit that depends on the exact domain.

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.

The main objective of this thesis is to provide algorithms towards solving this "micro-data reinforcement learning" challenge by *leveraging prior knowledge* or *building surrogate models*. The overall goal of the proposed approaches is to *minimize the interaction time between the robot and the environment required to solve the task at hand* (*i.e.*, minimize the operation time of the robot). We argue that this is the most appropriate metric to compare trial-and-error algorithms and we use it throughout this manuscript. Other metrics like the number of episodes or time-steps are very dependent on the specific task setups and can easily be "overfitted".

Our main motivation is the application of reinforcement learning to robot damage recovery. In our opinion, this is an application that can justify learning in the robotics community, as there is presently no consensus about the best analytic way to recover from damages in robots. Nevertheless, we additionally provide examples of general adaptation (*i.e.*, unforeseen situations that do not necessarily involve damage) and most of the algorithms presented do not have the explicit goal of solving this challenge, but rather the "micro-data reinforcement learning" challenge.

In the next chapter (chapter 2), we provide a review of policy search approaches (Deisenroth et al., 2013; Kober et al., 2013) that have the explicit goal of reducing the interaction time between the robot and the environment to a few seconds or minutes. Policy search methods learn parameters of a controller, called the policy, that maps sensor inputs to motor commands (*e.g.*, velocities or torques). We will see that most published algorithms for *micro-data policy search* implement and sometimes combine two main strategies: *leveraging prior knowledge* and *building surrogate models*.

Prior knowledge can be incorporated either in the policy structure, the policy parameters or the dynamics model. In the first case, this knowledge comes from the traditional robotics literature and the methods use well defined policy structures in order to make the search problem easier. Some examples of well defined policy spaces are: dynamic movement primitives (Ijspeert et al., 2013), finite-state automata (Calandra et al., 2015) or other hand-designed structures. In the second case, learning from demonstrations (Billard et al., 2008; Kober and Peters, 2009) or imitation learning has a long successful history in robotics; in short, demonstrated trajectories provide initialization of the parameters of an expressive policy such that local search around them is enough to find high-performing solutions. Lastly, recent works (Chatzilygeroudis and Mouret, 2018; Chatzilygeroudis et al., 2018b; Cutler and How, 2015; Saveriano et al., 2017; Bongard et al., 2006; Zhu et al., 2018) showcase that taking advantage of dynamics simulators or inaccurate simple dynamics models can help in reducing the amount of interaction time required to obtain an accurate model of the real-world environment.

During the execution of a policy on a robotic system, we can gather data

and create models to help us make better decisions on what to try next. We can find two types of algorithms inside this strategy: (a) algorithms that learn a surrogate model of the expected return from a starting state distribution (Brochu et al., 2010; Shahriari et al., 2016), and (b) algorithms that learn the transition dynamics of the robot/environment (Deisenroth et al., 2013; Chatzilygeroudis et al., 2018b). In the first case, the most promising family of approaches is Bayesian optimization (BO) (Brochu et al., 2010; Shahriari et al., 2016); BO is made of two main components: a surrogate model of the expected return, and an acquisition function, which uses the model to define the utility of each point of the search space. In the second case, using probabilistic models, like Gaussian processes (GPs) (Rasmussen and Williams, 2006), and taking into account the uncertainty of the predictions in the policy search seems to be the most promising direction of research for *model-based policy search* (Deisenroth et al., 2013; Chatzilygeroudis et al., 2018b; Polydoros and Nalpantidis, 2017).

In chapter 3, we consider a robot damage recovery scenario, where a waypoint-controlled robot is damaged in an unknown way and needs to find novel gaits in order to reach the waypoints fixed by its operator. To tackle this challenge, we combine probabilistic modeling and planning with a repertoire of high-level actions produced using a dynamics simulator of the intact robot. In this work, we will show that evolutionary algorithms, and more precisely quality-diversity or illumination algorithms (Mouret and Clune, 2015; Pugh et al., 2016), produce "creative priors" that can be beneficial when searching for a compensating behavior. We propose a new algorithm, called "Reset-free Trial-and-Error" (RTE), and we evaluate it on a simulated wheeled robot, a simulated six-legged robot, and a physical six-legged walking robot that are damaged in several ways (*e.g.*, a missing leg, a faulty motor, etc.) and whose objective is to reach a sequence of targets in an arena. Our experiments show that the robots can recover most of their locomotion abilities in an environment with obstacles, and without any human intervention.

In chapter 4, we confirm the result of the PILCO papers (Deisenroth and Rasmussen, 2011; Deisenroth et al., 2015, 2013) that using probabilistic models and taking into account the uncertainty of the prediction in the policy search is essential for effective model-based policy search (Deisenroth et al., 2015, 2013; Kober et al., 2013; Sutton and Barto, 1998). We, also, go one step further and showcase that combining the policy evaluation step with the optimization procedure can give us a more flexible, faster and modern implementation of model-based policy search algorithms. We propose a new model-based policy search algorithm, called Black-DROPS, and evaluate it on two standard benchmark tasks (*i.e.*, inverted pendulum and cartpole swing-up) and a physical manipulator. Our results showcase that we can keep the low interaction times of analytical approaches like PILCO, but also speed-up the computation due to the parallelization properties of population-based optimizers like CMA-ES (Hansen and Ostermeier, 2001).

In chapter 5, we combine dynamics simulators and Gaussian processes in order to scale up model-based policy search approaches to high-dimensional

robots. In particular, we introduce a new model learning procedure, called GP-MI, that combines model identification, black-box parameterized priors and Gaussian process regression. Overall, we argue that vanilla model-based policy search approaches are only practical in low-dimensional systems, but can be very powerful when combined with the appropriate prior knowledge and model learning procedure. Our results showcase that by combining Black-DROPS with GP-MI, we can learn to control a physical hexapod robot (48D state space, 18D action space) in less than one minute of interaction time.

In chapter 6, we discuss how safety constraints, robustness and multiple priors can be incorporated in a Bayesian optimization procedure towards solving the micro-data reinforcement learning challenge. More precisely, we first introduce sIT&E (Safety-aware Intelligent Trial & Error Algorithm) that extends the Intelligent Trial & Error algorithm to include safety criteria in the learning process; using this approach, a simulated damaged iCub humanoid robot (Nori et al., 2015; Tsagarakis et al., 2007) is able to safely learn crawling behaviors in less than 20 trials. We then propose ALOQ (ALternating Optimization and Quadrature) and TALOQ (Transferable ALOQ) aimed towards learning policies that are robust to rare events while being as sample efficient as possible. Using these approaches we learn robust policies using accurate and inaccurate simulators for a variety of systems (from simple simulated arms to a physical hexapod robot). Lastly, we introduce a new acquisition function for BO, called MLEI (Most Likely Expected Improvement), that effectively combines multiple sources of prior information in order to minimize the interaction time. Using MLEI a hexapod robot is able to find effective gaits in order to climb new types of stairs and adapt to unforeseen damages.

Before the conclusion of this manuscript (chapter 7), we discuss the main limitations of our proposed approaches and the main directions for future work. In this last chapter, we also highlight the interplay between planning, model-predictive control and policy search methods and discuss the main challenges that this new emerging field (micro-data reinforcement learning) faces.

Chapter 2

Background

The text of this chapter has been partially published in the following articles.

Articles:

- **Chatzilygeroudis, K.**, Vassiliades, V., Stulp, F., Calinon, S. and Mouret, J.-B., 2018. *A survey on policy search algorithms for learning robot controllers in a handful of trials.* Under review in *IEEE Transactions on Robotics* (Chatzilygeroudis et al., 2018b).

Other contributors:

- Vassilis Vassiliades (Post-doc)
- Freek Stulp (Head of department at DLR)
- Sylvain Calinon (Senior researcher at Idiap)
- Jean-Baptiste Mouret (Thesis supervisor)

Author contributions:

- **KC** and JBM organized the study. **KC** wrote the majority of the survey with improvements and suggestions by VV and JBM. FS and SC wrote most of section 2.5.

2.1 Introduction

The most successful traditional RL methods typically learn an action-value function that the agent consults to select the best action from each state (i.e., one that maximizes long-term reward) (Sutton and Barto, 1998; Mnih et al., 2015). These methods work well in discrete action spaces (and even better when combined with discrete state spaces)[1], but robots are typically controlled with continuous inputs and outputs (see (Deisenroth et al., 2013; Kober et al., 2013) for detailed discussions on the issues of classic RL methods in robotics).

[1] In Section 2.4 we give a brief overview of approaches based on value or action-value functions for continuous control.

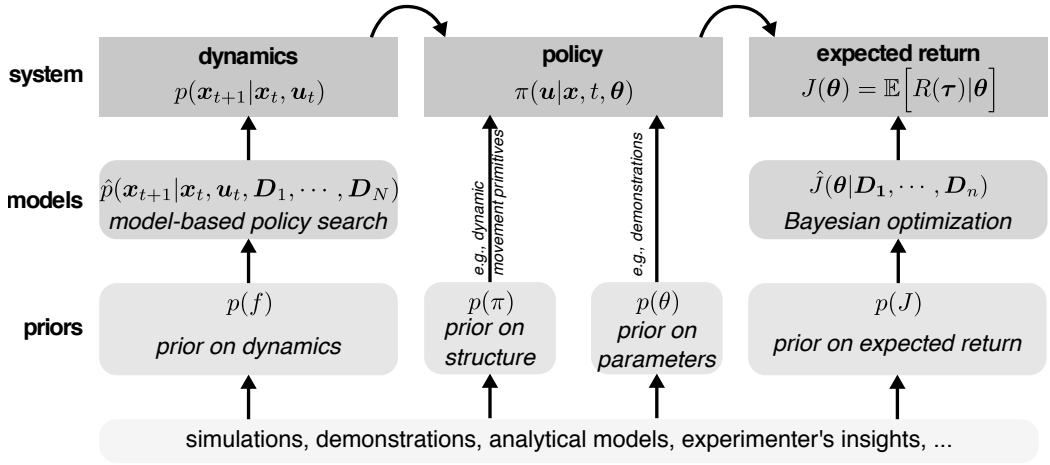


Figure 2.1: Overview of possible strategies for Micro-Data Policy Search (MDPS). The first strategy (bottom) is to leverage prior knowledge on the dynamics, on the policy parameters, on the structure of the policy, or on the expected return. A second strategy is to learn surrogate models of the dynamics or of the expected return; these models can also be initialized or guided by some prior knowledge.

As a result, the most promising approaches to RL for robot control do not rely on value functions; instead, they are *policy search* methods that learn parameters of a controller, called the policy, that maps sensor inputs to joint positions/torque (Deisenroth et al., 2013). These methods make it possible to use policies that are well-suited for robot control such as dynamic movement primitives (Ijspeert et al., 2003) or general-purpose neural networks (Levine and Koltun, 2013). In *direct* policy search, the algorithms view learning as an optimization problem that can be solved with gradient-based or black-box optimization algorithms (Stulp and Sigaud, 2013b). As they are not modeling the robot itself, these algorithms scale well with the dimensionality of the state space. They still encounter difficulties, however, as the number of parameters which define a policy, and thus the dimensionality of the search space, increases (Deisenroth et al., 2013). In *model-based policy search*, the algorithms typically alternate between learning a model of the robot and learning a policy using the learned model (Deisenroth et al., 2015; Chatzilygeroudis et al., 2017). As they optimize policies without interacting with the robot, these algorithms not only scale well with the number of parameters, but can also be very *data efficient*, requiring few trials on the robot itself to develop a policy. They do not scale well with the dimensionality of the state space, however, as the complexity of the dynamics tends to scale exponentially with the number of moving components.

In this chapter, 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. Most published algorithms for *micro-data policy search* implement and sometimes combine two main strategies (Fig. 2.1): *leveraging prior knowledge* and *building surrogate models*.

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 (Ng et al., 2006; Kober and Peters, 2009); some others assume that a damaged robot knows its model in its intact form, but not the damaged model (Cully et al., 2015; Pautrat et al., 2018; Chatzilygeroudis and Mouret, 2018). This knowledge can be introduced at different places, typically in the structure of the policy (*e.g.*, dynamic movement primitives; DMPs) (Ijspeert et al., 2003), in the reward function (*e.g.*, reward shaping), or in the dynamical model (Abbeel et al., 2006; Chatzilygeroudis and Mouret, 2018).

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 (Brochu et al., 2010; Shahriari et al., 2016); 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 (Ng et al., 2006)). The two strategies — priors and surrogates — are often combined; 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.

The rest of the chapter is structured as follows. Section 2.2 presents the problem formulation for reinforcement learning. Section 2.3 provides a brief overview of value-function based approaches and Section 2.4 discusses the goals of policy search and presents briefly the most important direct policy search approaches. Section 2.5 describes the work about priors on the policy structure and parameters. Section 2.6 provides an overview of the work on learning surrogate models of the expected return, with and without prior, while Section 2.7 is focused on learning models of the dynamics and the immediate reward. Section 2.8 lists the few noteworthy approaches for micro-data policy search that do not fit well into the previous sections.

2.2 Problem formulation

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

$$p(\boldsymbol{x}_{t+1}|\boldsymbol{x}_t, \boldsymbol{u}_t) \tag{2.1}$$

with continuous-valued states $\boldsymbol{x} \in \mathbb{R}^E$ and controls $\boldsymbol{u} \in \mathbb{R}^F$.

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

$$\boldsymbol{x}_{t+1} = f(\boldsymbol{x}_t, \boldsymbol{u}_t) + \boldsymbol{w} \tag{2.2}$$

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, where $\boldsymbol{\theta}$ are the policy parameters. Throughout the chapter 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 from an initial state distribution $p(\boldsymbol{x}_0)$, the system's states and actions jointly form *trajectories* $\boldsymbol{\tau} = (\boldsymbol{x}_0, \boldsymbol{u}_0, \boldsymbol{x}_1, \boldsymbol{u}_1, \ldots, \boldsymbol{x}_T)$, 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 $\boldsymbol{\tau}$:

$$R(\boldsymbol{\tau}) = \sum_{t=0}^{T-1} r_{t+1} = \sum_{t=0}^{T-1} r(\boldsymbol{x}_t, \boldsymbol{u}_t, \boldsymbol{x}_{t+1}) \tag{2.3}$$

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* $J(\boldsymbol{\theta})$ when following a policy $\pi_{\boldsymbol{\theta}}$ for t time-steps as:

$$\begin{aligned} J(\boldsymbol{\theta}) &= \mathbb{E}\Big[R(\boldsymbol{\tau})|\boldsymbol{\theta}\Big] \\ &= \int R(\boldsymbol{\tau}) P(\boldsymbol{\tau}|\boldsymbol{\theta}) \end{aligned} \tag{2.4}$$

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:

$$\underbrace{P(\boldsymbol{\tau}|\boldsymbol{\theta})}_{\text{trajectories for } \boldsymbol{\theta}} = \underbrace{p(\boldsymbol{x}_0)}_{\text{initial state}} \prod_t \underbrace{p(\boldsymbol{x}_{t+1}|\boldsymbol{x}_t, \boldsymbol{u}_t)}_{\text{transition dynamics}} \underbrace{\pi(\boldsymbol{u}_t|\boldsymbol{x}_t, t, \boldsymbol{\theta})}_{\text{policy}} \tag{2.5}$$

Value function An important concept in RL is the *value function*. It represents the expected return of a state when the system is controlled with policy $\pi_{\boldsymbol{\theta}}$. It differs from the reward function which is immediate in nature, as value functions account for the effect of future rewards, determining the long-term expected reward of a given state. The *value* of a state $\boldsymbol{x}_0$ under a policy $\pi_{\boldsymbol{\theta}}$ is the expected return of the system starting in state $\boldsymbol{x}_0$ and following policy $\pi_{\boldsymbol{\theta}}$ thereafter:

$$\begin{aligned} V_{\pi_{\boldsymbol{\theta}}}(\boldsymbol{x}_0) &= \mathbb{E}\Big[R(\boldsymbol{\tau})|\boldsymbol{\theta}, \boldsymbol{x}_0\Big] \\ &= \int R(\boldsymbol{\tau}) P(\boldsymbol{\tau}|\boldsymbol{\theta}, \boldsymbol{x}_0) \end{aligned} \tag{2.6}$$

Q-function Similarly, the *action-value function* or *Q-function*, $Q_{\pi_{\boldsymbol{\theta}}}(\boldsymbol{x}, \boldsymbol{u})$, is defined as the expected return of a state and action pair when the system is controlled with policy $\pi_{\boldsymbol{\theta}}$:

$$Q_{\pi_{\boldsymbol{\theta}}}(\boldsymbol{x}_0, \boldsymbol{u}_0) = \mathbb{E}\Big[R(\boldsymbol{\tau})|\boldsymbol{\theta}, \boldsymbol{x}_0, \boldsymbol{u}_0\Big] \tag{2.7}$$

2.3 Value-function approaches

The value-function based approaches attempt to estimate either $V_{\pi_{\boldsymbol{\theta}}}(\boldsymbol{x})$ or $Q_{\pi_{\boldsymbol{\theta}}}(\boldsymbol{x}, \boldsymbol{u})$. In the rest of this section, we will briefly discuss the main concepts and algorithms that fall into this category. Nevertheless, in this dissertation we mainly focus on policy search algorithms with the explicit goal of minimizing the interaction/learning time and thus we cover them in more detail.

Dynamic Programming-Based Methods In this category, the approaches assume that the state and action spaces are discrete[2] and the transition probabilities and immediate reward function fully known. These algorithms basically utilize the *Bellman equation* (Bellman, 1957):

$$V_{\pi_{\boldsymbol{\theta}}}(\boldsymbol{x}_t) = \mathbb{E}\Big[r_t + \gamma V_{\pi_{\boldsymbol{\theta}}}(\boldsymbol{x}_{t+1})\Big] \tag{2.8}$$

$$Q_{\pi_{\boldsymbol{\theta}}}(\boldsymbol{x}_t, \boldsymbol{u}_t) = \mathbb{E}\Big[r_t + \gamma Q_{\pi_{\boldsymbol{\theta}}}(\boldsymbol{x}_{t+1}, \boldsymbol{u}_{t+1})\Big] \tag{2.9}$$

where $\gamma \in [0, 1]$ is a factor that discounts future rewards.

The classical Dynamic Programming (DP) algorithms (Bellman, 1957; Werbos, 1992; Sutton and Barto, 1998) are not used a lot because of their need of a known model of the environment and because of their big computational cost (especially as the state/action space increases). However, they are still very interesting as they define fundamental computational mechanisms that can be used as parts of other more complicated algorithms.

Temporal Difference Methods Temporal Difference (TD) methods can learn directly from interaction with the environment with no prior model, and use bootstrapping to improve the value- or action-value function from the current estimates. TD learning methods can either be *on-policy* or *off-policy*. In on-policy learning, we learn the value of the policy being carried out by the agent including the exploration steps. SARSA (Rummery and Niranjan, 1994; Sutton and Barto, 1998) is on-policy as it updates the Q-values using the Q-value of the next state and the current policy's action; *i.e.*, it estimates the return for state-action pairs assuming that the current policy continues to be followed. Off-policy methods are more general: they learn the value of a different policy (which could be the optimal policy or not) independently of

[2]For continuous problems, one could discretize the spaces and operate in them, but this would, usually, result in a very big number of states and actions.

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 LearnStrategy 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**

the agent's actions. Q-learning (Watkins and Dayan, 1992; Sutton and Barto, 1998) updates the Q-values using the Q-value of the next state and the greedy action; *i.e.*, it estimates the return for state-action pairs assuming a greedy policy would be followed, and is independent of the policy being currently followed. Hence, off-policy methods are able to update the estimated value (or action-value) functions using hypothetical actions, that have not actually been tried, whereas on-policy methods update value (or action-value) functions based strictly on experience.

2.4 Policy Search

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 $\pi_{\boldsymbol{\theta}^*}$:

$$\boldsymbol{\theta}^* = \underset{\boldsymbol{\theta}}{\operatorname{argmax}}\, J(\boldsymbol{\theta}) \tag{2.10}$$

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 (Collect Strategy), 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 (LearnStrategy) with the data acquired so far, and selecting the next policy $\pi_{\boldsymbol{\theta}^*}$ to try on the robot (UpdateStrategy).

This generic outline allows us to describe direct (*e.g.*, policy gradient algorithms (Sutton et al., 2000)), surrogate-based (*e.g.*, Bayesian optimization (Brochu et al., 2010)) and model-based policy search algorithms, where each algorithm implements in a different way each of InitStrategy, CollectStrategy, LearnStrategy and UpdateStrategy. We will also see that 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's use a gradient-free optimizer (UpdateStrategy) and learn directly on the system (*i.e.*, LearnStrategy $= \emptyset$). This type of algorithms falls in the category of *model-free* or *direct* policy search algorithms (Sutton and Barto, 1998; Kohl and

Algorithm 2 Gradient-free direct policy search algorithm

1: **procedure** InitStrategy
2: Select $\boldsymbol{\theta}_0$ 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**

Stone, 2004). InitStrategy can be defined as randomly choosing some policy parameters, $\boldsymbol{\theta}_0$ (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. (2.3)). $\tilde{J}_{\boldsymbol{\theta}} = \frac{\sum_i^N R(\boldsymbol{\tau})_i}{N}$ 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 (Sutton and Barto, 1998). The objective of the present chapter is to describe algorithms that require several orders of magnitude less interaction time by leveraging priors and models.

In the rest of this section, we will briefly discuss the main concepts, formulations and algorithms that fall into the traditional direct policy search category. The approaches in this caterogy assume no prior knowledge of the system or the reward function and try to directly optimize the policy parameters on the real system (Kohl and Stone, 2004). As a result, these algorithms suffer from data-inefficiency, but nevertheless are important as they can be part of other more data-efficient approaches (*e.g.*, as an initialization or an optimizer). In this manuscript we focus mainly on published ideas that explicitly try to drastically reduce the interaction time between the robot and the environment (we refer the reader to (Sigaud and Stulp, 2018) for a recent review of policy search for continuous control).

2.4.1 Policy Gradient Algorithms

Sutton and Barto (1998) describe *"Generalized Policy Iteration"* (GPI) as the process that consists of two interacting parts, one pushing the value function (or the Q-function) to be consistent with the best current policy (*policy evaluation*), and a second one that aims to improve the policy greedily using the current value function (*policy improvement*). In the classical policy iteration formulation (Sutton and Barto, 1998), these two processes strictly alternate; *i.e.*, one happens exactly after the other has finished. Nevertheless, in more modern and asynchronous methods (Mnih et al., 2016), the policy evaluation and improvement steps can be interleaved at a finer grain. As long

as both steps update all states, the result is typically the same-convergence to the optimal value function and an optimal policy (Sutton and Barto, 1998). A lot of reinforcement learning algorithms can be described using the GPI formalization and most of the policy gradient algorithms fall under it. It is also important to note that all approaches that fall under GPI (*e.g.*, actor-critic methods) can also be seen as value-function based approaches. Nevertheless, we include them in the policy search section as the most successful policy gradient approaches do utilize some learned approximation of the value- or action-value function.

Actor-Critic Actor critic methods (Konda and Tsitsiklis, 2000) fall under the GPI formulation and they, naturally, consist of two parts. The *actor* that adjusts the parameters $\boldsymbol{\theta}$ of the policy by utilizing some policy gradient. And the *critic* that estimates the action-value function $\hat{Q}^{\pi}(\mathbf{x}_t, \mathbf{u}_t) = Q^{\pi}(\mathbf{x}_t, \mathbf{u}_t)$ with an appropriate policy evaluation method, *e.g.*, temporal-difference learning.

Stochastic Policy Gradients *Policy gradient algorithms* are the most popular class of continuous action reinforcement learning algorithms (Degris et al., 2012; Silver et al., 2014; Ciosek and Whiteson, 2018a; Schulman et al., 2015; Lillicrap et al., 2016; Zimmer et al., 2016). The overall idea is to adjust the policy parameters $\boldsymbol{\theta}$ in the direction of the performance gradient $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$. Computing this gradient analytically is not possible, as the state distributions under every policy parameters and the real action-value function $Q(\mathbf{x}, \mathbf{u})$ should be known. To make this computation efficient and practical, most algorithms utilize the results of the *stochastic policy gradient theorem* (SPG) (Sutton et al., 2000):

$$\begin{aligned} \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) &= \int P(\boldsymbol{\tau}|\boldsymbol{\theta}) \int \nabla_{\boldsymbol{\theta}} \pi(\mathbf{u}|\mathbf{x}, \boldsymbol{\theta}) Q^{\pi}(\mathbf{x}, \mathbf{u}) d\mathbf{x} d\mathbf{u} \\ &= \mathbb{E}\Big[\sum_{t=0}^{T-1} \nabla_{\boldsymbol{\theta}} \log \pi(\mathbf{u}_t|\mathbf{x}_t, \boldsymbol{\theta}) Q^{\pi}(\mathbf{x}_t, \mathbf{u}_t) \Big] \end{aligned} \tag{2.11}$$

The most interesting part of the policy gradient theorem (apart from its simplicity) is that despite the fact that the distribution over trajectories $P(\boldsymbol{\tau}|\boldsymbol{\theta})$ depends on the policy parameters $\boldsymbol{\theta}$, the policy gradient does not depend on the trajectory distribution. This result has very important practical value, as the computation of the policy gradient is reduced to a simple expectation. However, the stochastic policy gradient theorem requires that the policies are stochastic.

Deterministic Policy Gradients Silver et al. (2014) introduced the *deterministic policy gradient theorem* (DPG) that adapts the SPG theorem for deterministic policies:

$$\begin{aligned}\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) &= \int P(\boldsymbol{\tau}|\boldsymbol{\theta}) \nabla_{\mathbf{u}} Q^{\pi}(\mathbf{x}, \mathbf{u} = \pi(\mathbf{x}|\boldsymbol{\theta})) d\mathbf{x} \\ &= \mathbb{E}\Big[\sum_{t=0}^{T-1} \nabla_{\boldsymbol{\theta}} \log \pi(\mathbf{x}_t|\boldsymbol{\theta}) \nabla_{\mathbf{u}_t} Q^{\pi}(\mathbf{x}_t, \mathbf{u}_t = \pi(\mathbf{x}_t|\boldsymbol{\theta}))\Big] \end{aligned} \tag{2.12}$$

They also showed that the DPG theorem is a limiting case of the SPG theorem. This is important because it shows that the familiar machinery of policy gradients, for example compatible function approximation (Sutton et al., 2000), natural gradients(Kakade, 2002), actor-critic (Konda and Tsitsiklis, 2000), or episodic/batch methods, is also applicable to deterministic policy gradients.

Expected Policy Gradients Recently, Ciosek and Whiteson (2018a,b), taking inspiration from expected SARSA (Sutton and Barto, 1998; Van Seijen et al., 2009), introduced *expected policy gradients* (EPG) that unifies the SPG and DPG theorems and shows improved performance on several benchmarks. Their main contribution is to restate the Eq. (2.11) as follows:

$$\begin{aligned}\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) &= \int P(\boldsymbol{\tau}|\boldsymbol{\theta}) \int \nabla_{\boldsymbol{\theta}} \pi(\mathbf{u}|\mathbf{x}, \boldsymbol{\theta}) Q^{\pi}(\mathbf{x}, \mathbf{u}) d\mathbf{x} d\mathbf{u} \\ &= \int P(\boldsymbol{\tau}|\boldsymbol{\theta}) I_{\pi}^{Q}(\mathbf{x}) d\mathbf{x} \\ &= \mathbb{E}\Big[\sum_{t=0}^{T-1} I_{\pi}^{Q}(\mathbf{x}_t)\Big] \end{aligned} \tag{2.13}$$

This formulation makes explicit that one step in estimating the gradient is to evaluate an integral. The key insight of EPG is that given a state $\mathbf{x}_t$, $I_{\pi}^{Q}(\mathbf{x}_t)$ can be fully expressed with known quantities. Consequently, $I_{\pi}^{Q}(\mathbf{x}_t)$ can be analytically computed or approximated by Monte Carlo quadrature in cases where the integral is not possible to compute.

In their paper, Ciosek and Whiteson (2018b) formulate the *General Policy Gradient Theorem* that unifies both SPG and DPG theorems. In particular, they show that the choice between a deterministic or a stochastic policy is fundamentally a choice of the quadrature method for approximating $I_{\pi}^{Q}(\mathbf{x}_t)$. One important conclusion of their work is that the success of DPG over SPG should not be attributed to a fundamental issue of stochastic policies, but to superior (easier) quadrature method. Thanks to EPG, a deterministic policy is no longer required to obtain a method with low variance.

2.5 Using priors on the policy parameters or 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*[3].

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.*,

$$J_\zeta(\pi_\zeta^*) - \max_{\boldsymbol{\theta}} J_\zeta(\boldsymbol{\theta}) < \delta, \tag{2.14}$$

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 needs not be 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.

2.5.1 Hand-designed policies

One approach to reducing the policy parameter space is to hand-tailor it to the task ζ to be solved. In (Fidelman and Stone, 2004), 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.

[3] Freek Stulp and Sylvain Calinon greatly contributed in this section (Chatzilygeroudis et al., 2018b).

[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.

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.

2.5.2 Policies as function approximators

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

$$\underset{\Theta}{\operatorname{argmin}} \sum_{n=1}^{N} J_{\zeta_n}(\pi^*_{\zeta_n}) - \max_{\boldsymbol{\theta}} J_{\zeta_n}(\boldsymbol{\theta}), \text{ with } \boldsymbol{\theta} \in \Theta, \tag{2.15}$$

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].

Two examples of such generally applicable policy representations are linear policies (2.16), radial basis function networks (2.17), or neural networks, namely

$$\pi_{\boldsymbol{\theta}}(\boldsymbol{x}) = \boldsymbol{\theta}^\intercal \boldsymbol{\psi}(\boldsymbol{x}) \tag{2.16}$$

$$\pi_{\boldsymbol{\theta}}(\boldsymbol{x}) = \boldsymbol{w}^\intercal \boldsymbol{\psi}_{\boldsymbol{\theta}}(\boldsymbol{x}). \tag{2.17}$$

These more general policies can be used for many tasks (Guenter et al., 2007; Kober et al., 2013). However, prior knowledge is still required to determine the appropriate number of basis functions and their shape. 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 programming by demonstration can often be used to determine an initial policy. The initial parameters $\boldsymbol{\theta}$ are obtained through supervised learning, by providing the demonstration as training data $(\boldsymbol{x}_i, \boldsymbol{u}_i)_{i=1:N}$. This is discussed in more detail in Section 2.5.6

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 (Matsubara et al., 2011)—but the learned representation can typically be more expressive, facilitating adaptation and generalization.

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

2.5.3 Dynamical Movement Primitives

Dynamical Movement Primitives (DMPs) combine the generality of function approximators with the advantages of dynamical systems, such as robustness towards perturbations and convergence guarantees (Ijspeert et al., 2013, 2002). DMPs limit the expressiveness of the policy to typical classes of tasks in robotics, such as point-to-point movements ('discrete DMPs') or repetitive movements ('rythmic DMPs').

Discrete DMPs are summarized in Eq. 2.18. 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 $\ddot{y}$. Multi-dimensional DMPs are achieved by coupling multiple transformation systems with one canonical system. The vector $\boldsymbol{y}$ typically represents the end-effector pose or the joint angles.

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

$$\omega \ddot{y} = \underbrace{\alpha(\beta(y^g - y) - \dot{y})}_{\text{Spring-damper system}} + \underbrace{s\, f_{\boldsymbol{\theta}}(s)}_{\text{Forcing term}} . \qquad \text{(Transf.)} \qquad (2.18)$$

$$\omega \dot{s} = -\alpha_s s. \qquad \text{(Canonical)} \qquad (2.19)$$

This facilitates learning, because, whatever parameterization $\boldsymbol{\theta}$ of the function approximator we choose, a discrete DMP is guaranteed to converge towards a goal y^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 y, 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 (Stulp and Sigaud, 2013a). Approaches for learning the goal y^g of a discrete movement have also been proposed (Stulp et al., 2012). 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 (Stulp et al., 2013)). 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 (Calinon, 2016), but these models have only been applied to learning from demonstration applications so far.

Another approach to avoid the curse of dimensionality is to consider a hierarchical organization of the policy. In (Daniel et al., 2016), 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. These sub-policies are treated as latent variables in an expectation-maximization procedure, allowing the distribution of the update information between the sub-policies. In Queisser and Steil (Queisser and Steil, 2018), an upper-level policy is used to interpolate between policy parameterizations for different task variations. This substantially speeds up learning when many variations of the same task must be learned.

2.5.4 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 (Buchli et al., 2011), or a *linear quadratic tracking* (LQT) controller (Calinon et al., 2014). 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 (Buchli et al., 2011; Calinon et al., 2013) or forces (Kalakrishnan et al., 2011) 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.

2.5.5 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) (Stanley and Miikkulainen, 2002). It is even possible, in simulation, to co-evolve an appropriate body morphology and policy (Sims, 1994; Bongard and Pfeifer, 2003). As in natural evolution itself, these approaches require massive amounts of rollouts, and do not focus on learning in a handful of trials.

2.5.6 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, *i.e.* the initialization of $\boldsymbol{\theta}$ from a demonstrated trajectory. This is possible if we know the general movement a robot should make to solve the task, and are able to demonstrate it by recording our movement, or physically guiding the robot through kinesthetic teaching. 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.

2.6 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. (2.4)) 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 $\hat{J}(\boldsymbol{\theta}|R(\boldsymbol{\tau}|\boldsymbol{\theta}_1), \cdots, R(\boldsymbol{\tau}|\boldsymbol{\theta}_N))$.

2.6.1 Bayesian optimization

Algorithm 3 Policy search with Bayesian optimization

1: **procedure** CollectStrategy
2: Collect samples of the form $(\boldsymbol{\theta}, R(\boldsymbol{\tau}))$
3: **end procedure**
4: **procedure** LearnStrategy
5: Learn model $\hat{J} : \boldsymbol{\theta} \rightarrow J(\boldsymbol{\theta})$
6: **end procedure**
7: **procedure** UpdateStrategy
8: $\boldsymbol{\theta}_{n+1} = \text{argmax}_{\boldsymbol{\theta}}$ AcquisitionFunction$(\boldsymbol{\theta})$
9: **end procedure**

The most representative class of algorithms that falls in this category is Bayesian optimization (BO) (Brochu et al., 2010). Bayesian optimization 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.

Bayesian optimization, for policy search, follows the generic policy search algorithm (Algo. 1) and implements CollectStrategy, LearnStrategy 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 (Rasmussen and Williams, 2006) 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 (Rasmussen and Williams, 2006). More precisely, it is a distribution over functions, completely specified by its mean function, $\mu(\cdot)$ and covariance function, $k(\cdot,\cdot)$ and it is computed as follows:

$$\hat{J}(\boldsymbol{\theta}) \sim \mathcal{GP}(\mu(\boldsymbol{\theta}), k(\boldsymbol{\theta}, \boldsymbol{\theta}')) \tag{2.20}$$

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

$$p(\hat{J}(\boldsymbol{\theta}_*)|\boldsymbol{D}_{1:n}, \boldsymbol{\theta}_*) = \mathcal{N}(\mu(\boldsymbol{\theta}_*), \sigma^2(\boldsymbol{\theta}_*)) \tag{2.21}$$

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

$$\begin{aligned} \mu(\boldsymbol{\theta}_*) &= \boldsymbol{k}^T K^{-1} \boldsymbol{D}_{1:n} \\ \sigma^2(\boldsymbol{\theta}_*) &= k(\boldsymbol{\theta}_*, \boldsymbol{\theta}_*) - \boldsymbol{k}^T K^{-1} \boldsymbol{k} \end{aligned} \tag{2.22}$$

Exponential Kernel A kernel (also called a covariance function, kernel function, or covariance kernel), is a positive-definite function of two inputs $\boldsymbol{\theta}_i$ and $\boldsymbol{\theta}_j$. There exist many kernel functions that can be used with GPs. We can categorize them in two categories: (1) stationary, meaning that their value only depends on the difference $\boldsymbol{\theta}_i - \boldsymbol{\theta}_j$, and (2) non-stationary, meaning that their value depends on the actual inputs and not just their difference. The most widely used kernel for Gaussian process regression is the stationary Squared Exponential Kernel[6] (SE), defined as follows:

$$k_{\text{se}}(\boldsymbol{\theta}_i, \boldsymbol{\theta}_j) = \sigma_f^2 \exp(-\frac{1}{2}(\boldsymbol{\theta}_i - \boldsymbol{\theta}_j)^T \boldsymbol{\Lambda}^{-1} (\boldsymbol{\theta}_i - \boldsymbol{\theta}_j)) + \sigma_n^2 \delta_{ij} \tag{2.23}$$

where $\delta_{ij} = 1$ when $i = j$ and $\boldsymbol{\phi}_K = [\boldsymbol{\Lambda}, \sigma_f^2, \sigma_n^2]$ are the hyper-parameters of the kernel.

Maximum Likelihood Estimation In order to find the optimal parameters of the kernel, that is, to select the most likely model, we use the Maximum

[6]Maybe a better name is Squared Quadratic Kernel

Likelihood Estimation (MLE) and maximize the marginal likelihood of the GP model (or the log-likelihood):

$$\log p(\boldsymbol{D}_{1:n}|\boldsymbol{\theta}_{1:n}, \boldsymbol{\phi}_K) = -\frac{1}{2}\boldsymbol{D}_{1:n}^T K \boldsymbol{D}_{1:n} - \log|K| - \log(2\pi) \tag{2.24}$$

For the acquisition function, most algorithms use the Expected Improvement, the Upper Confidence Bound or the Probability of Improvement (Brochu et al., 2010; Hennig and Schuler, 2012).

Probability of Improvement One of the first acquisition functions is the Probability of Improvement (Kushner, 1964) (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 $\boldsymbol{D}_{1:n}$, in practice we query the approximated model $\hat{J}$ on $\boldsymbol{D}_{1:n}$ and get the best parameters. When using GPs as the surrogate model, this can be analytically computed:

$$\begin{aligned} PI(\boldsymbol{\theta}) &= p(\hat{J}(\boldsymbol{\theta}) > \hat{J}(\boldsymbol{\theta}^+)) \\ &= \Phi\Big(\frac{\mu(\boldsymbol{\theta}) - \hat{J}(\boldsymbol{\theta}^+)}{\sigma(\boldsymbol{\theta})}\Big) \end{aligned} \tag{2.25}$$

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 (Brochu et al., 2010).

Expected Improvement The Expected Improvement (Brochu et al., 2010) (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:

$$\begin{aligned} &I(\boldsymbol{\theta}) = \max\{0, \hat{J}(\boldsymbol{\theta}) - \hat{J}(\boldsymbol{\theta}^+)\} \\ &EI(\boldsymbol{\theta}) = \mathbb{E}(I(\boldsymbol{\theta})) \\ &= \begin{cases} (\mu(\boldsymbol{\theta}) - \hat{J}(\boldsymbol{\theta}^+))\Phi(Z) + \sigma(\boldsymbol{\theta})\phi(Z), & \text{if } \sigma(\boldsymbol{\theta}) > 0. \\ 0, & \text{otherwise.} \end{cases} \\ &Z = \frac{\mu(\boldsymbol{\theta}) - \hat{J}(\boldsymbol{\theta}^+)}{\sigma(\boldsymbol{\theta})} \end{aligned} \tag{2.26}$$

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 (Hennig and Schuler, 2012). When using GPs as the surrogate model, it is defined as follows:

$$UCB(\boldsymbol{\theta}) = \mu(\boldsymbol{\theta}) + \alpha\sigma(\boldsymbol{\theta}) \tag{2.27}$$

where α is a user specified parameter.

Hennig and Schuler (2012) performed a thorough experimental analysis and concluded that EI can perform better on artificial objective functions than PI and UCB, but more recent experiments on gait learning on a physical robot suggested that UCB can outperform EI in real situations (Calandra et al., 2015).

Martinez-Cantin et al. (2007) 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 does not depend on the dimensionality of the state space, 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 (Lizotte et al., 2007) (around 100 trials with a well-chosen 15D policy space), a small biped "compass robot" (Calandra et al., 2015) (around 100 trials with a finite state automata policy), and a pocket-sized, vibrating soft tensegrity robot (Rieffel and Mouret, 2018) (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 more difficult when the dimension increases (Bellman, 1957). 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 in optimization and some ideas might be applied to robotics in the future; according to the optimization literature, random embeddings (Wang et al., 2016) and additive models (Kandasamy et al., 2015; Rolland et al., 2018) are among the most promising ideas.

2.6.2 Bayesian optimization with priors

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 (Cully et al., 2015). IT&E first uses MAP-Elites (Cully et al., 2015), an evolutionary illumination (Mouret and Clune, 2015; Vassiliades et al., 2017) (also known as quality-diversity (Pugh et al., 2016)) 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 $\boldsymbol{D}_{1:n} = \{R(\boldsymbol{\tau}|\boldsymbol{\theta}_1), ..., R(\boldsymbol{\tau}|\boldsymbol{\theta}_n)\}$ 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:

$$p(\hat{J}(\boldsymbol{\theta}_*)|\boldsymbol{D}_{1:n}, \boldsymbol{\theta}_*) = \mathcal{N}(\mu(\boldsymbol{\theta}_*), \sigma^2(\boldsymbol{\theta}_*)) \tag{2.28}$$

The mean and variance predictions of this GP are computed using a kernel vector $\boldsymbol{k} = k(\boldsymbol{D}_{1:n}, \boldsymbol{\theta}_*)$, 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:

$$\begin{aligned} \mu(\boldsymbol{\theta}_*) &= R_m(\boldsymbol{\theta}_*) + \boldsymbol{k}^T K^{-1}(\boldsymbol{D}_{1:n} - R_m(\boldsymbol{\theta}_{1:n})) \\ \sigma^2(\boldsymbol{\theta}_*) &= k(\boldsymbol{\theta}_*, \boldsymbol{\theta}_*) - \boldsymbol{k}^T K^{-1} \boldsymbol{k} \end{aligned} \tag{2.29}$$

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 (Cully et al., 2015; Lee et al., 2017; Chatzilygeroudis et al., 2018a).

Following a similar line of thought but implemented differently, a few recent works (Antonova et al., 2016, 2017) use a simulator to learn the kernel function of a GP, instead of utilizing it to create a mean function like in IT&E (Cully et al., 2015). In particular, Antonova et al. (2016) used domain knowledge for bipedal robots (*i.e.*, *Determinants of Gait* (DoG) (Inman et al., 1953)) 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 proved to be beneficial and their approach outperformed both traditional BO and state-of-the-art black-box optimizers (CMA-ES). Moreover, in their follow-up work (Antonova et al., 2017), the same authors use neural networks 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) to walk in a handful of trials.

A similar but more general idea (*i.e.*, no real assumption about the underlying system) was introduced by (Wilson et al., 2014). 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 $k(\boldsymbol{\theta}_i, \boldsymbol{\theta}_j) = \alpha \exp D(\boldsymbol{\theta}_i, \boldsymbol{\theta}_j)$, 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. (2014) 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 2.7), 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 in the dynamics (see Section 2.7.2) and recent improvements on BO (see Section 2.6.1) could make their approach more practical.

Instead of using the simulator to precompute priors, Marco et al. (2017) 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 (ES) (Hennig and Schuler, 2012), which selects 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:

$$ES(\boldsymbol{\theta}) = \operatorname*{argmax}_{\boldsymbol{\theta}} \mathbb{E}[\Delta H(\boldsymbol{\theta})] \tag{2.30}$$

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

Lober et al. (2016) use a BO procedure that selects parameterizations of a QP-based whole body controller (Salini et al., 2011; Spitz et al., 2017) in

order to control a humanoid robot. In particular, they formulate a policy that includes the QP-based controller (that contains an approximate 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 (Lober et al., 2016, 2017).

Safety-Aware Approaches

Berkenkamp et al. (2016) 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. Using this approach a quadrotor vehicle was able to safely improve its performance over the initial sub-optimal policy.

2.7 Learning models of the dynamics

Instead of learning a model of the expected long-term reward (section 2.6.1), 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* (Deisenroth et al., 2013; Polydoros and Nalpantidis, 2017), in which the trajectory data are used to learn the dynamics model, then policy search is performed on the model (Sutton, 1991; Kaelbling et al., 1996).

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

$$\hat{\boldsymbol{x}}_{t+1} = \hat{f}(\boldsymbol{x}_t, \boldsymbol{u}_t) \tag{2.31}$$

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

2.7.1 Model-based Policy Search

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:

$$\hat{J}(\boldsymbol{\theta}) = \mathbb{E}[\hat{R}(\boldsymbol{\tau})|\boldsymbol{\theta}] = \int \hat{R}(\boldsymbol{\tau})\hat{P}(\boldsymbol{\tau}|\boldsymbol{\theta}) \tag{2.32}$$

where

$$\hat{P}(\boldsymbol{\tau}|\boldsymbol{\theta}) = p(\boldsymbol{x}_0)\prod_t \hat{p}(\boldsymbol{x}_{t+1}|\boldsymbol{x}_t, \boldsymbol{u}_t)\pi_{\boldsymbol{\theta}}(\boldsymbol{u}_t|\boldsymbol{x}_t, t) \tag{2.33}$$

This iterative scheme can be seen as follows:

$$\boldsymbol{\tau}_n \sim P(\boldsymbol{\tau}|\boldsymbol{\theta}_n) \tag{2.34}$$
$$\boldsymbol{D}_n = \boldsymbol{D}_{n-1} \cup \{\boldsymbol{\tau}_n, R(\boldsymbol{\tau}_n)\} \tag{2.35}$$
$$\boldsymbol{\theta}_{n+1} = \operatorname*{argmax}_{\boldsymbol{\theta}} \hat{J}(\boldsymbol{\theta}|\boldsymbol{D}_n) \tag{2.36}$$

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 $\boldsymbol{D}$. After some stopping criteria is met (usually number of episodes or convergence of policy parameters), the policy that approximately maximizes Eq. (2.10) can be straightforwardly retrieved:

$$\pi_{\boldsymbol{\theta}^*} = \boldsymbol{\theta}_{\text{last}} \tag{2.37}$$

If the system (and the policy) is not stochastic, one could retrieve the best policy differently:

$$\pi_{\boldsymbol{\theta}^*} = \operatorname*{argmax}_{\boldsymbol{\theta}} R(\boldsymbol{\tau}_n) \tag{2.38}$$

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** LearnStrategy
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, LearnStrategy 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. (2.32) 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 modelling the transition dynamics, practical algorithms are available only for relative small state-action spaces (Chatzilygeroudis et al., 2017; Deisenroth et al., 2015, 2013; Polydoros and Nalpantidis, 2017).

2.7.1.1 Model Learning

There exist many approaches to learn the models $\hat{f}$ and $\hat{r}$ (for model-based policy search) in the literature (Tangkaratt et al., 2014; Levine and Abbeel, 2014; Deisenroth et al., 2015). We can categorize the learned models in deterministic (*e.g.*, neural networks or linear regression) and probabilistic ones (*e.g.*, Gaussian Processes).

Probabilistic models usually rely on Bayesian methods and are typically non-parametric (and thus exhibit potentially infinite capacity), whereas deterministic models are typically parametric (and thus do not have infinite capacity). Probabilistic models are usually more effective than deterministic models in model-based policy search (Deisenroth et al., 2013) 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). PILCO (Deisenroth and Rasmussen, 2011) utilizes Gaussian processes (GPs) to greatly reduce the interaction time to solve several tasks, like the cart-pole swing-up task.

Recently, the model-based Policy Gradients with Parameter-based Exploration (M-PGPE) algorithm (Tangkaratt et al., 2014) suggested instead of learning the model $\hat{f}$, to directly try to estimate the transition probabilities $p(\boldsymbol{x}_{t+1}|\boldsymbol{x}_t, \boldsymbol{u}_t)$ using least-squares conditional density estimation (Sugiyama et al., 2010). Using this formulation they were able 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 (Levine et al., 2016; Levine and Abbeel, 2014; Kumar et al., 2016); *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 (Levine et al., 2016; Levine and Abbeel, 2014) and even dexterous manipulation tasks (Kumar et al., 2016).

There has, also, recently been some work on using Bayesian Neural Networks (BNNs) (Gal and Ghahramani, 2016) to improve the scaling of model-based policy search algorithms (Gal et al., 2016; Higuera et al., 2018). 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 neural networks has been recently proposed (Chua et al., 2018) for learning probabilistic dynamics models of higher dimensional systems; for example, state-of-the-art performance was obtained in controlling the half-cheetah benchmark (Wawrzynski, 2007) 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 (Depeweg et al., 2017); traditional model learning approaches

(*e.g.*, Gaussian processes) 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 (Depeweg et al., 2018).

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. (2017) 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.

2.7.1.2 Long-term predictions

We can 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 (Deisenroth et al., 2013).

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 distribution $\hat{p}(\hat{r}_{t+1}|\boldsymbol{x}_t, \boldsymbol{u}_t, \boldsymbol{x}_{t+1})$. When applying a policy (with some parameters $\boldsymbol{\theta}$) on the model, we get a *rollout* or *trajectory*:

$$\boldsymbol{\tau} = (\boldsymbol{x}_0, \boldsymbol{u}_0, \boldsymbol{x}_1, \boldsymbol{u}_1, \ldots, \boldsymbol{x}_T) \tag{2.39}$$

$$\boldsymbol{r} = (\hat{r}_1, \hat{r}_2, \ldots, \hat{r}_T) \tag{2.40}$$

$$\hat{R}(\boldsymbol{\tau}) = \sum_{t=0}^{T-1} \hat{r}_{t+1} \tag{2.41}$$

where

$$\boldsymbol{x}_0 \sim p(\boldsymbol{x}_0) \tag{2.42}$$

$$\hat{r}_{t+1} \sim \hat{p}(\hat{r}_{t+1}|\boldsymbol{x}_t, \boldsymbol{u}_t, \boldsymbol{x}_{t+1}) \tag{2.43}$$

$$\boldsymbol{u}_t \sim \pi_{\boldsymbol{\theta}}(\boldsymbol{u}_t|\boldsymbol{x}_t, t) \tag{2.44}$$

$$\boldsymbol{x}_{t+1} \sim \hat{p}(\boldsymbol{x}_{t+1}|\boldsymbol{x}_t, \boldsymbol{u}_t) \tag{2.45}$$

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):

$$\tilde{\hat{J}}(\boldsymbol{\theta}) = \frac{1}{m} \sum_{i=1}^{m} \hat{R}_i(\boldsymbol{\tau}^i) \tag{2.46}$$

One more efficient way of computing the expected long-term reward with stochastic trajectories is the PEGASUS sampling procedure (Ng and Jordan, 2000). 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 2.7.1.2). Nevertheless, these sampling-based procedures can experience big variances in the predictions that can negatively affect the optimization process.

Model-based contextual REPS (Kupcsik et al., 2017) heavily uses sampling-based policy evaluations and showed that when using enough sample trajectories, you can get better approximations than deterministic long-term predictions (see Section 2.7.1.2); another recent work also strongly justifies the usage of sampling-based policy/action evaluations over deterministic inference methods (Chua et al., 2018) (especially in higher dimensional systems). They were also able to greatly reduce the computation time by exploiting the parallelization capabilities of modern GPUs. In their paper, model-based contextual REPS is able to learn policies for controlling robots that play table tennis and hockey, where different goal positions are handled as different contexts.

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 (Anderson and Moore, 1979), sigma-point methods (Julier and Uhlmann, 2004) or moment matching (Deisenroth

et al., 2015). 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:

$$\hat{p}(\boldsymbol{x}_{t+1}) = \iiint \hat{p}(\boldsymbol{x}_{t+1}|\boldsymbol{x}_t, \boldsymbol{u}_t)\hat{p}(\boldsymbol{x}_t, \boldsymbol{u}_t)d\boldsymbol{x}_t d\boldsymbol{u}_t d\boldsymbol{w} \tag{2.47}$$

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 (Anderson and Moore, 1979), sigma-point methods (Julier and Uhlmann, 2004) or moment matching (Deisenroth et al., 2015). The PILCO algorithm (Deisenroth and Rasmussen, 2011) uses moment matching, which is the best unimodal approximation of the predictive distribution in the sense that it minimizes the Kullback-Leibler divergence between the true predictive distribution and the unimodal approximation (Deisenroth et al., 2013).

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 (Deisenroth et al., 2015) 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) (Deisenroth and Rasmussen, 2011). It was also able to learn on a physical low-cost manipulator (Deisenroth et al., 2011) and simulated walking tasks (Deisenroth et al., 2012) among the many successful applications of the algorithm (Deisenroth et al., 2015).

2.7.2 Using priors on the dynamics

Reducing the interaction time in model-based policy search can be achieved by using priors on the models (Bischoff et al., 2014; Deisenroth et al., 2014; Chatzilygeroudis and Mouret, 2018; Cutler and How, 2015; Lee et al., 2017; Saveriano et al., 2017; Wu and Movellan, 2012); *i.e.*, starting with an initial guess of the dynamics (and/or the reward function) and then learning the residual model. This type of algorithms follow the general model-based policy search framework (Algo. 4) and usually implement different types of InitStrategy. Notably, most of the approaches (and the most successful ones) rely on Gaussian processes to model the dynamics, as priors can be very elegantly incorporated

(the formulation is identical to Eq. (2.28), but re-stated with the proper the notation for clarity):

Gaussian Processes dynamics with priors Assuming $\boldsymbol{D}_{1:t} = \{f(\tilde{\boldsymbol{x}}_1), ..., f(\tilde{\boldsymbol{x}}_t)\}$ 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 $\tilde{\boldsymbol{x}}_*$ as follows (of course we have E independent GPs; one for each output dimension (Chatzilygeroudis et al., 2017; Deisenroth and Rasmussen, 2011)):

$$p(\hat{f}(\tilde{\boldsymbol{x}}_*)|\boldsymbol{D}_{1:t}, \tilde{\boldsymbol{x}}_*) = \mathcal{N}(\mu(\tilde{\boldsymbol{x}}_*), \sigma^2(\tilde{\boldsymbol{x}}_*)) \tag{2.48}$$

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

$$\begin{aligned} \mu(\tilde{\boldsymbol{x}}_*) &= M(\tilde{\boldsymbol{x}}_*) + \boldsymbol{k}^T K^{-1}(\boldsymbol{D}_{1:t} - M(\tilde{\boldsymbol{x}}_{1:t})) \\ \sigma^2(\tilde{\boldsymbol{x}}_*) &= k(\tilde{\boldsymbol{x}}_*, \tilde{\boldsymbol{x}}_*) - \boldsymbol{k}^T K^{-1} \boldsymbol{k} \end{aligned} \tag{2.49}$$

A few approaches (Ko et al., 2007; Bischoff et al., 2014) use simple analytic and fast simulators prior to create a Gaussian process prior of the dynamics (and assume the reward function to be known). PILCO with priors (Cutler and How, 2015) uses simulated data (from running PILCO in the simulator) to create a GP prior for the dynamics and then performs policy search with PILCO. PILCO with priors 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 (Saveriano et al., 2017), 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.

An approach that splits the self-modeling process from the policy search is presented in (Bongard et al., 2006). The authors were among the first ones to combine a self-modelling procedure (close to model identification (Siciliano and Khatib, 2016)) with policy search. The self-modelling 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/system. 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 efficiently (*i.e.*, 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).

VGMI (Zhu et al., 2018) 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.

2.8 Other approaches

2.8.1 Guided policy search

Guided policy search (GPS) with unknown dynamics (Levine et al., 2016; Levine and Abbeel, 2014) 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 2.7.1.1) and indirect policy search where it attempts to approximate the local controllers with one big neural network 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) (Levine et al., 2016).

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 (Levine et al., 2016)) 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 (LeCun et al., 2015). 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 (Levine et al., 2016; Levine and Abbeel, 2014) among the many successful applications of the algorithm (Montgomery et al., 2017; Levine and Koltun, 2013).

2.8.2 Transferability approaches

The main hypothesis of the transferability approach (Koos et al., 2013b,a) is that physics simulators are accurate for some policies, *e.g.*, static gaits, and innacurate 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 (Section 2.6.1) because it is essentially a classification problem (instead of a regression problem). 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 (Koos et al., 2013b) (15 trials on the robot), for a walking quadruped robot (Koos et al., 2013b; Koos and Mouret, 2012) (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 (Koos et al., 2013a) (25 trials). Similar ideas were recently developed for humanoid robots with QP-based controllers (Spitz et al., 2017).

2.9 Conclusion

In this chapter, we presented two main strategies for tackling the "micro-data reinforcement learning" challenge that emerged from recently published works: *i.e.*, *leveraging prior knowledge* and *building surrogate models*. We, also, showcased that most published algorithms usually combine these two strategies (even when they do not explicitly discuss or acknowledge it) in order to further reduce the interaction time needed to learn a task.

Prior knowledge can be introduced at different places: in the structure/type of the policy (*e.g.*, dynamic movement primitives), in the policy parameters (*e.g.*, from demonstrations), in the reward function (*e.g.*, reward shaping) and in the dynamics model (*e.g.*, simulators). We can categorize the surrogate-based methods into (a) algorithms that learn a surrogate model of the expected return (*i.e.*, long-term reward) from a starting state; and (b) algorithms that learn models of the transition dynamics and/or the immediate reward function.

Overall, it is possible to learn with real robots in a handful of trials by leveraging these two strategies. Nevertheless, it is still not obvious how to (a) generate generic but explicit priors, (b) perform policy search effectively and withing reasonable computation time in a model-based setting, (c) scale up micro-data approaches to high dimensional robots, and (d) exploit several priors to speed-up the learning.

In the next chapter (chapter 3), we will see how evolutionary algorithms (more precisely quality-diversity or illumination algorithms (Mouret and Clune, 2015; Pugh et al., 2016)) can be used with a simulated robot to generate "creative" priors that can be beneficial both in performance and computation time when searching for a behavior on the real robot. In chapters 4, we will showcase that combining the policy evaluation step with the optimization procedure can give us a more flexible, faster and modern implementation of model-based policy search algorithms (Deisenroth et al., 2013). In chapter 5, we will introduce a dynamics model learning method that combines model identification and Gaussian processes and it is able to scale up to high dimensional robots. Lastly,

in chapter 6 we will see how we can intelligently select the most promising prior from multiple ones in a Bayesian optimization (BO) procedure in order to reduce the interaction time and leverage multiple sources of information.

Chapter 3

Reset-free Trial and Error for Robot Damage Recovery

The results and text of this chapter have been partially published in the following articles.

Articles:

- **Chatzilygeroudis, K.**, Vassiliades, V. and Mouret, J.-B., 2018. *Reset-free trial-and-error learning for robot damage recovery.* **Robotics and Autonomous Systems**, 100, pp.236-250 (Chatzilygeroudis et al., 2018a).
- **Chatzilygeroudis, K.**, Cully, A. and Mouret, J.B., 2016. *Towards semi-episodic learning for robot damage recovery.* **Workshop on AI for Long-Term Autonomy**, ICRA (Chatzilygeroudis et al., 2016).

Other contributors:

- Vassilis Vassiliades (Post-doc)
- Antoine Cully (Lecturer at Imperial College London)
- Jean-Baptiste Mouret (Thesis supervisor)

Author contributions:

- **KC** and JBM organized the studies. **KC** wrote the code and performed the experiments. **KC**, VV, AC and JBM analyzed the results and wrote the papers.

3.1 Introduction

Following the discussion in the introduction, in this chapter we will focus on a robot damage recovery scenario inspired by search-and-rescue missions (Guizzo, 2011; Atkeson et al., 2015; DeDonato et al., 2017). The robots that operate in these missions are inherently complex machines that have to cope with a

possibly dynamic and sometimes adversarial environment. This realization means that these robots must be able to perform their tasks as robustly as possible and to adapt to unforeseen situations. Moreover, in these scenarios hardware failures and damages is almost the norm; for instance, C. Atkeson et al. report that the Atlas robot they used in the DARPA Robotics challenge had a "mean time between failures of hours or, at most, days" (Atkeson et al., 2015; DeDonato et al., 2017). Therefore, these robots would greatly benefit from algorithms that allow them to autonomously learn how to cope with damages. In particular, damage recovery possess three interesting properties that make micro-data reinforcement learning an appealing approach for tackling it:

1. Hardware failures will always be a possibility, especially with highly complex robots in complex environments (Carlson and Murphy, 2005).

2. The traditional diagnosis process (Isermann, 2006; Verma et al., 2004; Lengagne et al., 2013) does not provide a generic solution because accurate diagnosis becomes increasingly challenging as the robots and environments become more complex: the probability of failing grows exponentially with the complexity of the robot and the environment.

3. Prior knowledge can be naturally justified as the designers can and should know a lot — if not everything — about the intact robot and its intended use.

Additionally, a limitation of most of the current RL methods used in robotics — that is usually not discussed — is that after each trial, the robot needs to be reset to the same state (Kober et al., 2013; Deisenroth et al., 2011, 2013; Polydoros and Nalpantidis, 2017). While this reset is often not a problem for a manipulator, it prevents mobile robots (*e.g.*, a stranded mobile manipulator or a legged robot) from exploiting this kind of algorithms to recover from damage in real-world situations. Moreover, the robot cannot ignore its environment while learning, which is usually the case, as it may be further damaged if it makes a wrong decision. For example, if the robot is in front of a wall and needs to try a new way to move, it should not try to go forward, but it should select actions that would make it more likely to move backwards in order to avoid hitting the wall.

Therefore, an ideal damage recovery algorithm should (1) not need any reset between episodes, (2) scale well enough with respect to the dimensionality of the state/action space of the robot, so that it can be used for "complex" robots (*e.g.*, legged robots) with the computing resources that are typically embedded in modern robots, and (3) explicitly take into account the environment. *The objective of the present chapter is to introduce a reinforcement learning algorithm that fits these three requirements by exploiting specific features of the damage recovery problem.*

More precisely, we investigate a simplified scenario that captures these challenges: a waypoint-controlled robot is damaged in a way that is unknown to its operator (*e.g.*, a leg is partially cut or a motor working at half speed); to

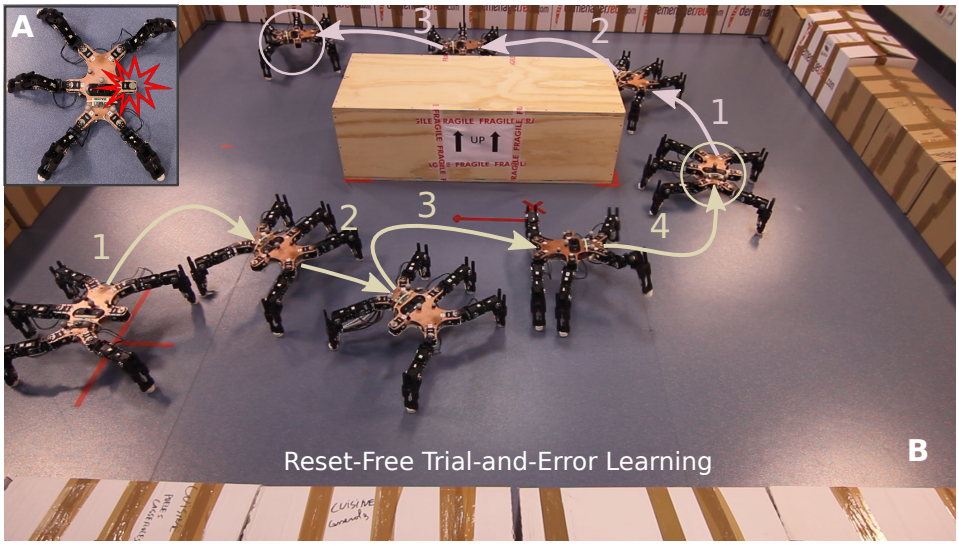


Figure 3.1: A typical experiment with the Reset-free Trial-and-Error (RTE) algorithm. **A.** A 6-legged (hexapod) robot is damaged; *i.e.*, missing a leg. **B.** The robot uses RTE to learn how to compensate while completing its task and taking into account the environment. As the robot moves, it improves its performance, *i.e.*, it needs fewer episodes to reach the next target.

get out of the building, the robot must recover its locomotion abilities so that it can reach the waypoints fixed by its operator. Our objective is to have the robot recover its locomotive abilities to the maximum extent possible in the shortest amount of time (Fig. 3.1). We assume that no diagnosis is available or that the diagnosis failed, either because the robot lacks the right sensor or because the damage is so out of the ordinary that it cannot be properly diagnosed. For simplicity, we also assume that the environment is known to the robot; we will discuss possible extensions of our approach when the environment is unknown in the discussion section.

Our first source of inspiration is the recently introduced Intelligent Trial and Error (IT&E) algorithm (Cully et al., 2015). This algorithm is an episodic policy search algorithm that is specifically designed for damage recovery. It addresses the scaling challenge by assuming that some high-performing policies for the intact robot still work on the damaged robot. While this assumption does not always hold, empirical experiments show that it often holds with highly redundant robots (*e.g.*, legged robots or humanoids) (Cully et al., 2015; Koos et al., 2013a) because (1) there are often many ways to perform a task, and (2) the outcomes of behaviors that do not use the damaged parts are similar between the intact and the damaged robot. Using this assumption, IT&E searches for a diverse set of high-performing policies *before the mission* (offline), then performs the *online* search, that is, the adaptation to damage, by searching solely in this lower-dimensional set of pre-selected policies (using Bayesian optimization) (Cully et al., 2015). As a result, most of the trials required for the policy search are transferred from the real damaged robot, which can perform only a few trials, to simulations with the intact robot, which

can perform many more trials, especially on modern computing clusters. For instance, IT&E allows an 18-DOF hexapod robot to learn to walk after several injuries within a dozen episodes (Cully et al., 2015) and only two minutes of combined interaction and computation time.

A second source of inspiration is the recent AlphaGo algorithm that succeeded in beating the European and World champions in the game of Go (Silver et al., 2016). Essentially, the authors use deep learning to pre-compute default policies and initial values for a Monte Carlo Tree Search (MCTS) (Chaslot et al., 2008; Browne et al., 2012) algorithm that plans (approximately) the best next action to take. We can draw an analogy in robotics and pre-compute actions or policies, learn the model of the robot on-line (the physical robot is damaged) and use MCTS to select the most promising action. Interestingly, MCTS can also take into account the uncertainty of the prediction of the model of the environment (*e.g.*, when using Gaussian processes for models (Nguyen-Tuong and Peters, 2011)). Unfortunately, it seems unrealistic to learn a probabilistic model of the full dynamics of a walking robot (like in (Hester and Stone, 2013)) within a few seconds (or minutes) of interaction time and the on-board computational power of a typical robot; more importantly, a probabilistic planner that would plan in the full controller space is even more computationally demanding.

Our main idea is to adapt the pre-computing part of IT&E, so that it can be used by a MCTS-based planner to select the next trial, in place of the Bayesian optimization used in IT&E. In addition, we utilize a probabilistic model to learn how to correct the outcome of each action on the damaged robot and use the MCTS-based planner in a similar way as in AlphaGo (Silver et al., 2016) and the TEXPLORE algorithm (Hester and Stone, 2013), but also incorporating the uncertainty of the model prediction in the search. This allows us to propose a trial-and-error learning algorithm for damage recovery that can work on a real hexapod robot, within reasonable computation time (less than 1 minute between each episode), that does not need any reset between each episode and takes into account the environment when learning. We call this new algorithm "Reset-free Trial-and-Error" (RTE). In the following, we will show that RTE performs significantly better than a modified (improved) version of TEXPLORE in both a simple differential drive mobile robot and a hexapod robot locomotion task (in the latter task, we empirically evaluate that TEXPLORE is not applicable due to the dimensionality of the action space).

The main contributions are as follows:

- a novel formulation of robot damage recovery as a model-based RL problem;
- a novel combination of learning techniques that resembles that of AlphaGo and exploits simulations of the intact robot to accelerate learning on the physical, damaged robot;
- extensive experiments in simulation with a damaged simple differential drive mobile robot and a damaged hexapod (6-legged) robot, which

validate the performance of the proposed approach and show that RTE performs and scales significantly better than TEXPLORE;

- experimental validation on a physical, damaged hexapod robot that recovers most of its locomotion abilities and is able to complete its task(s), *without any human intervention.*

3.2 Problem Formulation

Here, we adapt the generic problem formulation of Section 2.2 to our specific case. Our problem can be cast in the general framework of Markov Decision Processes (MDP) (Sutton and Barto, 1998). An MDP is a tuple (X, U, P, r), where X is the state space (continuous or discrete), U is the action space (continuous or discrete), $P(\boldsymbol{x}_t, \boldsymbol{u}_t, \boldsymbol{x}_{t+1})$ is the state transition function specifying the probability of transitioning to state $\boldsymbol{x}_{t+1} \in X$ when the agent takes action $\boldsymbol{u}_t \in U$ in state $\boldsymbol{x}_t \in X$, and $r : S \to \mathbb{R}$ is the immediate reward function (which defines the task of the agent), with $r(\boldsymbol{x}_{t+1})$ being the immediate reward of state $\boldsymbol{x}_{t+1}$ and $\boldsymbol{x}_{t+1}$ may contain both internal variables (such as body position) and external variables (such as obstacles). The objective of the agent (i.e., the robot) is to find a deterministic policy π, i.e., a mapping from states to actions, $\boldsymbol{u}_t = \pi(\boldsymbol{u}_t)$, that maximizes its expected discounted return:

$$J^{\pi} = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r(\boldsymbol{x}_{t+1}) \Big| \pi\right] \tag{3.1}$$

where $\gamma \in [0, 1)$ is a factor that discounts future rewards. P and r describe the environmental dynamics and they are collectively known as the model of the environment. If the agent has access to this model, it can use a planning algorithm to find the optimal policy. In this work, the transition function P is learned and we assume that the reward function r is known to the robot.

In our setting, the robot needs to execute a sequence of related tasks $G_1, G_2, \ldots, G_n$, each of which is a shortest path problem:

$$r(\boldsymbol{x}_t) = \begin{cases} R_{goal} & \text{if } \boldsymbol{x}_t = goal(G_i) \\ -R_{term} & \text{if } \boldsymbol{x}_t = terminal(G_i) \\ 0 & \text{otherwise} \end{cases} \tag{3.2}$$

where $R_{goal} > 0, R_{term} \geq 0$, $goal(G_i)$ returns the goal state of task G_i, and $terminal(G_i)$ returns a non-goal, terminal state of task G_i, *e.g.*, a colliding state. When $\boldsymbol{x}_t = goal(G_i)$, the robot finishes task G_i and starts executing task G_{i+1}.

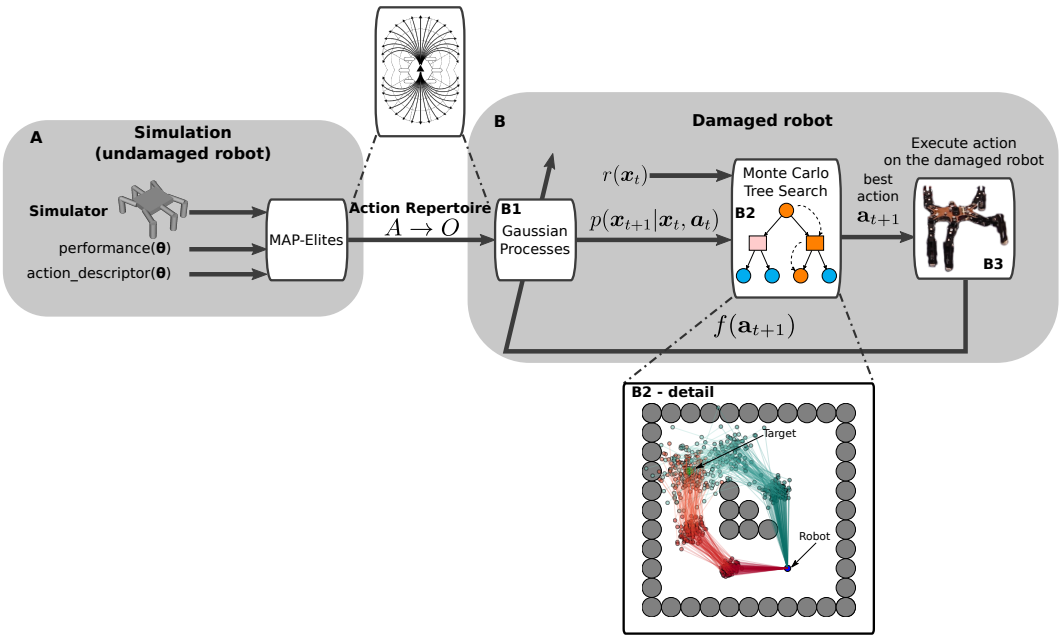


Figure 3.2: Overview of Reset-free Trial-and-Error (RTE) algorithm. **A.** Before deploying the robot, simulations with the intact robot are used to generate an action repertoire with the MAP-Elites algorithm. **B.** This repertoire is refined using a probabilistic learning model (Gaussian processes here — **B1**). This model is then used as the black-box simulator of a probabilistic planner (Monte Carlo Tree Search here — **B2**), which computes and outputs the best action to complete the task taking into account the uncertainty of the model. To better illustrate what happens in this phase, we *"zoom in"* and illustrate one simple case (**B2-detail**). The robot (blue circle) has to reach the target (green circle) without hitting the obstacles (gray circles) and its model is uncertain. The algorithm explores several alternative paths to the goal (here only 2 for illustration purposes) and chooses the path that achieves the largest expected return (here we select the red one, as the green one collides more often). The lines of the same color are sampled from the distribution of choosing the specific action sequence. Once the best path is selected, the physical damaged robot executes the first action (**B3**) of the path and updates the repertoire with the new gathered data. The algorithm then re-explores new ways to reach the goal and the process continues until the task is completed.

3.3 Approach

3.3.1 Overview

RTE allows robots to "learn while doing" instead of "learning and then doing". This is achieved by:

- pre-computing an action repertoire with relatively low-fidelity simulations (*e.g.*, perfect velocity actuators) of the intact robot (generated by MAP-Elites (Mouret and Clune, 2015), Fig. 3.2A) that also **(a)** creates a mapping between the task space and the parameters of the low-level controller and **(b)** reduces the dimensionality of the action space;

- using a probabilistic model (Gaussian processes) to learn how to correct the prediction of the outcome of each action for the damaged robot (Fig. 3.2B1);

- re-planning at every episode with a probabilistic planner (Monte Carlo Tree Search) that selects the next action to execute, based on the predictions of the probabilistic model, the uncertainty of those predictions, the environment, the current state of the robot, and the target state (Fig. 3.2B2). More specifically, we solve a path/motion planning problem with uncertain transitions (Fig. 3.2B2-detail). Clearly, the further we plan into the future, the more uncertain our estimates will be about where the robot will end up (Fig. 3.2B2-detail); therefore, an ideal planner would select the action that has the best utility (in terms of expected discounted cumulative reward) by considering these future estimates (i.e., how close they arrive to the target, how often they hit obstacles).

In summary, if damage occurs, RTE performs the following loop (Fig. 3.2B): (1) uses MCTS to select the next best action from the repertoire to complete the task, (2) executes the action for a given time duration (*e.g.*, 3 seconds or 100 simulation steps), that is, *perform an episode*, (3) updates the Gaussian processes (GPs) to improve the prediction of the outcome of each action of the repertoire and (4) repeats (1)-(3) until the task(s) are completed.

3.3.2 Learning the Action Repertoire

Controllers for complex robots, for instance legged robots, usually involve numerous parameters, which makes control policies challenging to learn within a few trials. We circumvent this issue by using the transferability hypothesis (Sec. 2.8.2) and learn, before deploying the robot, a repertoire of controllers with a simulated intact robot. The predicted outcomes of the actions will be refined online after each action is executed (i.e., at the end of each episode) by the damaged robot (Sec. 3.3.3).

We assume that the robot is controlled by a low-level controller that is parametrized by a vector $\boldsymbol{\theta} \in \mathbb{R}^d$. We also assume that each point in the task space can be described by a vector $\boldsymbol{a} \in \mathbb{R}^{n_a}$, which we call an "action descriptor". We would like to create a repertoire that covers the task space as well as possible (Cully et al., 2015; Duarte et al., 2017), i.e., to both determine a good set of actions A and a mapping between A and Θ ($A \rightarrow \Theta$). This mapping also reduces the dimensionality of the search space since the task space is usually much lower dimensional than the controller space.

If we take a robotic manipulator as an example, the controller space could be joint positions, the task space could be the (x, y, z) coordinates of the end-effector, and the repertoire will map (x, y, z) positions to joint positions, that is, it would be a discrete representation of the inverse kinematics of the arm. Nonetheless, while an inverse kinematics solver could be used to create a repertoire for a manipulator, most robots do not have access to such inverse models. This is true for walking robots, in particular.

As a consequence, instead of using an inverse model, we learn the action repertoire with an iterative algorithm called MAP-Elites (Mouret and Clune, 2015; Cully et al., 2015) and a forward model (*e.g.*, a dynamic simulator). As

Algorithm 5 MAP-Elites

1: **procedure** MAP-ELITES
2: $(\mathbf{P} \leftarrow \emptyset, \mathbf{\Theta} \leftarrow \emptyset)$ ▷ *Performance and feature grids*
3: **for** $i = 1 \rightarrow G$ **do** ▷ *Initialization: G random θ*
4: $\boldsymbol{\theta} = \text{random\_solution}()$
5: $\text{add\_to\_repertoire}(\boldsymbol{\theta}, \mathbf{P}, \mathbf{\Theta})$
6: **end for**
7: **for** $i = 1 \rightarrow I$ **do** ▷ *Main loop, I iterations*
8: $\boldsymbol{\theta} = \text{random\_selection}(\mathbf{\Theta})$
9: $\boldsymbol{\theta}' = \text{random\_variation}(\boldsymbol{\theta})$
10: $\text{add-to-repertoire}(\boldsymbol{\theta}', \mathbf{P}, \mathbf{\Theta})$
11: **end for**
12: **return** repertoire and performance ($\mathbf{\Theta}$, $\mathbf{P}$)
13: **end procedure**
14: **procedure** ADD-TO-REPERTOIRE$(\boldsymbol{\theta}, \mathbf{P}, \mathbf{\Theta})$
15: $\boldsymbol{a} = \text{action\_descriptor}(\boldsymbol{\theta})$ ▷ *Use the forward model*
16: $p = \text{performance}(\boldsymbol{\theta})$ ▷ *Use the forward model*
17: **if** $\mathbf{P}(\boldsymbol{a}) = \emptyset$ or $\mathbf{P}(\boldsymbol{a}) < p$ **then** ▷ *Replace if better*
18: $\mathbf{P}(\boldsymbol{a}) = p$
19: $\mathbf{\Theta}(\boldsymbol{a}) = \boldsymbol{\theta}$
20: **end if**
21: **end procedure**

with the inverse kinematics of redundant manipulators, the mapping from the parameter space to the task space is typically many-to-one. Thus, we need to define a performance function to select the best $\boldsymbol{\theta}$ for each point of the task space. This performance function is designed so as to promote certain type of behaviors (Sec. 3.6.1) and does not coincide with the reward function of the MDP.

Essentially, MAP-Elites discretizes the n_a-dimensional task space to an n_a-dimensional grid, and then attempts to fill each of the cells using a variation-selection loop (Mouret and Clune, 2015; Cully et al., 2015; Cully and Demiris, 2017). Algorithmically, it starts with G random parameter vectors, simulates the robot with these parameters, and records both the position of the robot in the task space and the performance (Algo. 5, 3-5). If the cell is free, then the algorithm stores the parameter vector in that cell; if it is already occupied, then the algorithm compares the performance values and keeps only the best parameter vector (Algo. 5, 10-15). Once this initialization is done, MAP-Elites iterates a simple loop (Algo. 5, 6-9): (1) randomly selects one of the occupied cells, (2) adds a random variation to the parameter vector, (3) simulates the behavior, (4) inserts the new parameter vector into the grid if it performs better or end-ups in an empty cell (discard the new parameter vector otherwise).

While MAP-Elites is computationally expensive, it can be straightforward to parallelize and can run on large clusters before deploying the robot. So far, it has been successfully used to generate: behaviors for legged robots (Cully et al., 2015), robotic arms (Cully et al., 2015; Mouret and Clune, 2015) and

wheeled robots (Duarte et al., 2016; Pugh et al., 2016; Duarte et al., 2017); designs for airfoils (Gaier et al., 2017), as well as for the morphologies of walking "soft robots" (Mouret and Clune, 2015); adversarial images for deep neural networks (Nguyen et al., 2015a); "innovation engines" which generate images that resemble natural objects (Nguyen et al., 2016); and 3D-printable objects using feedback from neural networks trained on 2D images (Lehman et al., 2016). MAP-Elites has also been extended to effectively handle task spaces of arbitrary dimensionality (Vassiliades et al., 2017).

3.3.3 Learning with Gaussian Processes

MAP-Elites provides not only the set of actions to be used by the planner, but also a prior on how an action modifies the state variables, i.e., a mapping from actions to relative outcomes, $f : A \rightarrow O$. Since this prior comes from a simulator and the simulator uses a model of the intact robot, it is only an approximation. Therefore, to make the physical damaged robot perform well, there needs to be a way to correct this mapping.

To do so, we use n Gaussian Processes (where n is the number of dimensions of O) with a mean function that corresponds to the prior provided by MAP-Elites. In other words, the mapping computed with the simulator serves as a prior for the GPs (see Sec. 2.7.2). For each dimension $d = 1 \dots n$, we use a separate GP that it is a distribution over functions specified by its mean function, $\mu_d(\cdot)$ and covariance function, $k_d(\cdot, \cdot)$:

$$f_d(\boldsymbol{a}) \sim GP(\mu_d(\boldsymbol{a}), k_d(\boldsymbol{a}, \boldsymbol{a}')) \tag{3.3}$$

3.3.4 Probabilistic Optimal Planning using MCTS

At the end of each episode, we need to solve an MDP with an action set that contains thousands of actions in a continuous state space. Since GPs are probabilistic models, they provide both a prediction and the uncertainty associated with each prediction, which can be exploited by probabilistic planners. Here we use Monte Carlo Tree Search (MCTS) (Chaslot et al., 2008), as it has already been successfully used to solve (Partially Observable)-MDPs with stochastic transition functions (Silver and Veness, 2010; Browne et al., 2012), continuous state spaces, and high branching factors (Browne et al., 2012; Couëtoux et al., 2011).

MCTS is a best-first, sample-based search algorithm for finding optimal decisions in a given domain by taking random samples in the decision space and building a search tree according to the results. Every state in the search tree is evaluated by the average outcome of Monte Carlo rollouts from that state. These rollouts are typically random or directed by a simple, domain-dependent heuristic (Browne et al., 2012).

MCTS (Algo. 6) is an anytime planning algorithm, i.e., it runs until some predefined computational budget (typically, a time, memory or iteration con-

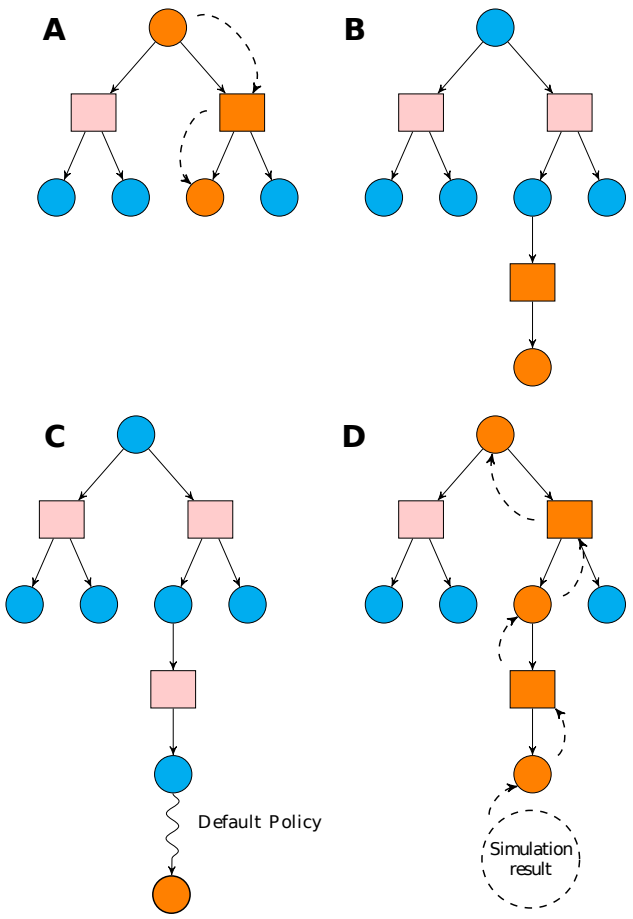


Figure 3.3: The four generic steps of Monte Carlo Tree Search algorithms. The circular nodes represent *decision* nodes (states from where actions are selected) and the rectangular nodes represent *random* nodes (state-action pairs where random outcomes can happen). See (Couëtoux et al., 2011) for further details. **A.** The most urgent expandable node (i.e., one with no previous visits) is selected using a selection policy. **B.** The tree is expanded according to the available actions. **C.** A rollout is performed from the new node according to the default policy. **D.** The rollout result is "backed up" through the selected nodes.

straint) is reached, at which point the search is halted and the best-performing root action is returned. Four steps are applied per search iteration:

- *SelectionPolicy:* Starting at the root node, a child selection policy is recursively applied to descend through the tree until the most urgent expandable node is reached (Fig. 3.3A).
- *ExpansionPolicy:* One child node, along with the state's associated reward $\rho = r(\boldsymbol{x})$, is added to expand the tree, according to the available actions (Fig. 3.3B).
- *Rollout:* A rollout is performed from the new node according to the default policy to get an estimate value for this node, Δ (Fig. 3.3C). We do this by constructing a generative model using the prediction of the GPs.

Algorithm 6 Generic Monte Carlo Tree Search

1: **procedure** MCTS-SEARCH($\boldsymbol{x}_0$)
2: **while** within computational budget **do**
3: $\boldsymbol{x} = \boldsymbol{x}_0$
4: **do**
5: $\boldsymbol{a}$ = SelectionPolicy($\boldsymbol{x}$)
6: $Children(\boldsymbol{x}) = Children(\boldsymbol{x}) \cup (\boldsymbol{x}, \boldsymbol{a})$
7: $(\boldsymbol{x}', \rho)$ = ExpansionPolicy($\boldsymbol{x}, \boldsymbol{a}$) ▷ *see (Couëtoux et al., 2011)*
8: $Children(\boldsymbol{x}, \boldsymbol{a}) = Children(\boldsymbol{x}, \boldsymbol{a}) \cup \boldsymbol{x}'$
9: $R(\boldsymbol{x}, \boldsymbol{a}) = \rho$
10: $\boldsymbol{x} = \boldsymbol{x}'$
11: **while** $n(\boldsymbol{x}) > 0$ and $\boldsymbol{x}$ not a terminal state ▷ $n(\cdot)$ *returns the number of visits of a state*
12: Δ = Rollout($\boldsymbol{x}$) ▷ *Use GPs (Sec. 3.3.3, 3.5.3)*
13: BackUp($\boldsymbol{x}, \Delta, R$)
14: **end while**
15: **return** BestChild($\boldsymbol{x}_0$)
16: **end procedure**

- *BackUp:* The rollout result is "backed up" through the selected nodes to update their statistics (Fig. 3.3D).

3.3.5 Reset-free Trial-and-Error Learning Algorithm

Connecting all the pieces together, RTE first generates an action repertoire with the MAP-Elites algorithm (Algo. 7, lines 2-3); then, while in mission, it re-plans at each episode using MCTS and the current belief of the outcome of the actions (prediction of the GPs), taking into account the uncertainty of the predictions and potential final states (*e.g.*, collisions with obstacle) (lines 9-13); at the end of each episode, the GPs are updated with the recorded data (lines 14-15).

3.4 Experimental Setup

We investigate the following scenario: a waypoint-controlled robot is damaged in a way that is unknown to its operator (*e.g.*, a leg is partially cut or a motor working at half speed); to get out of the building, the robot must recover its locomotion abilities so that it can reach the waypoints fixed by its operator. As already stated, we assume that no diagnosis is available or that the diagnosis failed. In addition, for the sake of simplicity, the environment is known to the robot and the robot knows its position (via a Motion Capture system). The robot has to reach 30 equidistant target waypoints in an arena with obstacles. We perform these experiments with a differential drive robot (in simulation) and with a 6-legged (hexapod) robot (in simulation and with a physical robot).

Algorithm 7 Reset-free Trial-and-Error Learning

procedure RTE
  Create Action Repertoire, A, with MAP-Elites (Sec. 3.3.2)
  Construct mean function M from MAP-Elites data
  for $i = 1 \rightarrow \dim(O)$ **do**
    $GP_i : A \rightarrow O_i$ with M_i as prior (Sec. 3.3.3, 3.5.2)
  end for
  while in mission and stopping criteria not met **do**
    RTE-EPISODE(t)
    $t = t + 1$
  end while
end procedure
procedure RTE-EPISODE(t)
  $\boldsymbol{x}_t$ = state of robot at time t
  $\boldsymbol{a}_{t+1}$ = MCTS-SEARCH($\boldsymbol{x}_t$) (Sec. 3.3.4, 3.5.3)
  $f(\boldsymbol{a}_{t+1})$ = execute_action($\boldsymbol{a}_{t+1}$) ▷ *Execute the action and observe its outcome*
  $D_{1:t+1} = \{D_{1:t}, f(\boldsymbol{a}_{t+1})\}$
  for $i = 1 \rightarrow \dim(O)$ **do**
    Update GP_i using $D^i_{1:t+1}$ (Sec. 3.3.3)
  end for
end procedure

We compare three algorithms: (1) RTE, (2) a variant of RTE where the learning part is removed (i.e., MCTS-based planning with the original action repertoire — we call this variant MCTS) and (3) a variant of TEXPLORE (we call it GP-TEXPLORE — Algo. 8) where: (i) the reward function is known, (ii) we use a variant of MCTS for continuous action spaces, and (iii) instead of learning the full transition dynamics, only the relative outcome of each action is learned. The main difference of GP-TEXPLORE and RTE is that the latter uses the discrete action space as defined by the learned repertoire for model learning and planning, whereas GP-TEXPLORE plans and learns the model in the full controller space. We also use GPs, without taking into account the uncertainty, instead of random forests that are used in the original TEXPLORE paper (Hester and Stone, 2013). With (2), we try to get closer to a classic planning algorithm with re-planning after each episode. With (3) we try to make TEXPLORE better fit our problem and we expect the original TEXPLORE algorithm to not work as well as the baseline used here. However, exploring more in these directions is beyond the scope of this paper.

3.5 Mobile Robot Results

The robot is a classic velocity-actuated differential drive mobile robot (Fig. 3.4A). The state of the robot consists of the (x, y) position and the orientation θ of the robot, i.e., $\boldsymbol{x}_{mob} = [x, y, \theta]$. The robot moves by applying velocities to the two wheels (v_{left} and v_{right}). We use the *libfastsim* library for simulating the

Algorithm 8 Modified TEXPLORE

procedure GP-TEXPLORE
  for $i = 1 \rightarrow \dim(O)$ **do**
    $GP_i : \Theta \rightarrow O_i$ ▷ controller space to outcome space
  end for
  while in mission and stopping criteria not met **do**
    GP-TEXPLORE-EPISODE(t)
    $t = t + 1$
  end while
end procedure
procedure GP-TEXPLORE-EPISODE(t)
  $\boldsymbol{x}_t$ = state of robot at time t
  $\boldsymbol{\theta}_{t+1}$ = MCTS-SEARCH($\boldsymbol{x}_t$) (Sec. 3.3.4) ▷ MCTS in controller space
  $f(\boldsymbol{\theta}_{t+1})$ = execute_action($\boldsymbol{\theta}_{t+1}$) ▷ *Execute the action and observe its outcome*
  $D_{1:t+1} = \{D_{1:t}, f(\boldsymbol{\theta}_{t+1})\}$
  for $i = 1 \rightarrow \dim(O)$ **do**
    Update GP_i using $D^i_{1:t+1}$ (Sec. 3.3.3)
  end for
end procedure

robot (Mouret and Doncieux, 2012)[1]. At each *episode* of the learning algorithm, the velocity pair is executed for 100 time-steps (for all algorithms).

3.5.1 Learning the Action Repertoire

The robot's task is to reach points in Cartesian space (x, y), therefore MAP-Elites should produce a repertoire of actions, each of which reaches a different point in the Cartesian space. Since many controllers can reach the same position, we select those that make the robot follow a continuous-curvature trajectory and for which the body points towards the tangent of the overall trajectory at the end of the behavior. To capture this idea, we set the MAP-Elites performance of the i_{th} individual to:

$$p_i = |\theta_i - \theta_d| \tag{3.4}$$

where θ_i is the orientation of the robot and θ_d is the desired orientation of the robot at the end of the movement. To describe the circular trajectories we only need to keep the (x, y) position of the robot at the end of the movement (since we can compute the desired angle for any point in the 2-D space). In this way we can use a 2-D action descriptor to describe the 3-D task space. The 2-D descriptor of the i_{th} individual is:

$$\boldsymbol{a}_i = [\frac{x - x_{min}}{x_{max} - x_{min}}, \frac{y - y_{min}}{y_{max} - y_{min}}] \tag{3.5}$$

where x_{min}, x_{max}, y_{min} and y_{max} are the boundaries of the reachable space ($[-100, 100]$ units here). We ran MAP-Elites for 100000 evaluations and we

[1] https://github.com/jbmouret/libfastsim

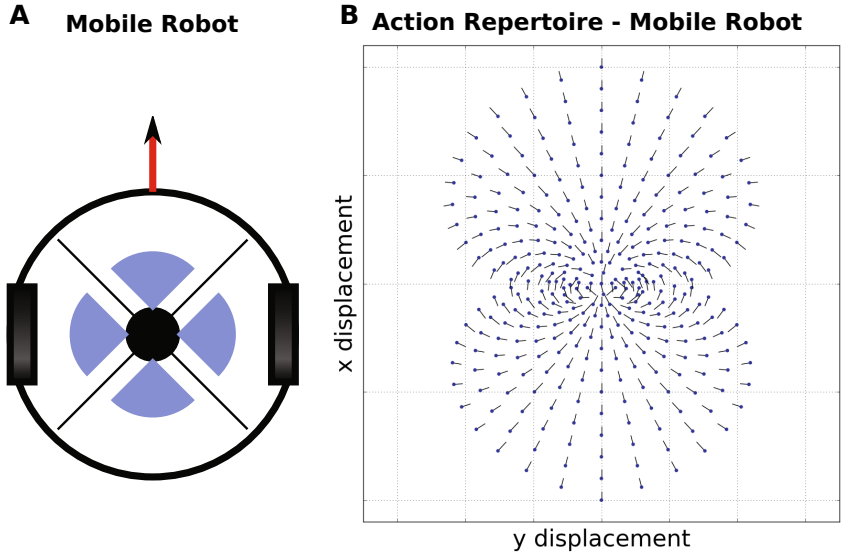


Figure 3.4: A. The velocity-actuated differential drive mobile robot used in our experiments. **B.** Repertoire for the simple robot locomotion task produced by the MAP-Elites algorithm. This repertoire maps the 2-D action descriptor (of the 3-D task space) to the 2-D controller space. Each dot represents a different action (and its x, y position), while the lines indicate the orientation of the robot at the end of each behavior. All the behaviors are relative to the zero position that is located in the middle of the figure and relative to the forward orientation (line pointing up).

got a repertoire with 331 different actions[2] (Fig. 3.4B). Our implementation relies on the Sferes$_{v2}$ (Mouret and Doncieux, 2010) library.

3.5.2 Learning with Gaussian Processes

The GP inputs are the 2-D descriptors of the actions, and the outputs are predictions of the relative x, y and θ displacements. To avoid angle discontinuities, instead of learning the raw angle θ we learn the $cos\theta$ and the $sin\theta$. Thus, we learn a mapping from actions to relative outcomes:

$$\boldsymbol{a} \rightarrow (\Delta x, \Delta y, cos\Delta\theta, sin\Delta\theta) \tag{3.6}$$

We use the *Squared Exponential Kernel* (SE) as the covariance function (Rasmussen and Williams, 2006):

$$k(\boldsymbol{a}, \boldsymbol{a}') = \sigma_{se}^2 \exp\Big(- \frac{||\boldsymbol{a} - \boldsymbol{a}'||^2}{l^2} \Big) \tag{3.7}$$

where we set $\sigma_{se}^2 = 0.5$ and $l = 1$ in the mobile robot experiments. We also use the limbo C++11 library (Cully et al., 2018) for the GP regression.

[2]MAP-Elites always produces the same repertoire because the problem is easy. Note that the repertoire for such a simple robot could be generated with many other methods: here we use MAP-Elites so that we can demonstrate identical approaches for both the wheeled and the legged robot.

Algorithm 9 Simple Progressive Widening

procedure SPW-SELECTIONPOLICY($\boldsymbol{x}$)
  if $n(\boldsymbol{x})^\alpha > \#Children(\boldsymbol{x})$ **then** ▷ $0 < \alpha < 1$
    $\boldsymbol{a} = \text{sample\_action}(\boldsymbol{x})$ ▷ Sample new action from $\boldsymbol{x}$ (Sec. 3.5.4)
  else ▷ Choose the action with the best UCT value (Browne et al., 2012)
    $\boldsymbol{a} = \text{argmax}_{\boldsymbol{a} \in Children(\boldsymbol{x})} \hat{Q}(\boldsymbol{x}, \boldsymbol{a})$, with $\hat{Q}(\boldsymbol{x}, \boldsymbol{a}) = \frac{\mathbf{R}(\boldsymbol{x},\boldsymbol{a})}{n(\boldsymbol{x},\boldsymbol{a})} + c\sqrt{\frac{ln(n(\boldsymbol{x}))}{n(\boldsymbol{x},\boldsymbol{a})}}$
  end if
  return $\boldsymbol{a}$
end procedure

3.5.3 Solving the problem with MCTS

At the end of each episode, we need to solve an MDP with an action set that contains thousands of actions in a continuous state space and uncertain transitions (i.e., when an action is taken from the same state, the result is not the same).

In order to solve this problem, we instantiate MCTS with the following choices:

Selection Policy Simple Progressive Widening (SPW — Algo. 9) (Rolet et al., 2009) that properly handles cases where the action space is continuous. We set $\alpha = 0.5$ and $c = 150$.

Expansion Policy Double Progressive Widening (DPW — Algo. 10) (Couëtoux et al., 2011) that properly handles cases where the state space is continuous. We set $\beta = 0.6$.

Action Sampling Policy We use A* on a simplified problem to guide the sampling procedure (Sec. 3.5.4).

Generative Model We construct a generative model using the prediction of the GPs:

$$p(\boldsymbol{x}_{t+1}|\boldsymbol{x}_t, \boldsymbol{a}_t) \sim \mathcal{N}(\boldsymbol{x}_t + f(\boldsymbol{a}_t), \Sigma_{\boldsymbol{a}_t}) \tag{3.8}$$

Default Policy for evaluation Uniformly-distributed random actions from the repertoire (Browne et al., 2012).

Best child criterion Greedy selection, i.e., we select the action that has the maximum average cumulative reward (Browne et al., 2012).

Reward function $R_{goal} = 100$, reward for reaching the goal, and $R_{term} = 1000$, penalty for colliding, for each target point. We also set the reward discount factor, $\gamma = 0.9$.

- For the sake of simplicity, we only used circular obstacles and a circular collision shape for the robot. Nevertheless, any shapes with the appropriate collision query functions would be compatible with our approach, since the reward function is a black-box to MCTS.

Algorithm 10 Double Progressive Widening

procedure DPW-ExpandPolicy($\boldsymbol{x}, \boldsymbol{u}$)
  if $n(\boldsymbol{x}, \boldsymbol{a})^\beta > \#Children(\boldsymbol{x}, \boldsymbol{a})$ **then** ▷ $0 < \beta < 1$
    Draw s' from $p(\boldsymbol{x}'|\boldsymbol{x}, \boldsymbol{a})$ ▷ see Eq. 3.8
    $\rho = r(\boldsymbol{x}')$
  else
    Choose $\boldsymbol{x}' \in Children(\boldsymbol{x}, \boldsymbol{a})$ with prob $\frac{n(\boldsymbol{x},\boldsymbol{a},\boldsymbol{x}')}{\sum_{\boldsymbol{x}_i} n(\boldsymbol{x},\boldsymbol{a},\boldsymbol{x}_i)}$
    $\rho = r(\boldsymbol{x}')$
  end if
  return $[\boldsymbol{x}', \rho]$
end procedure

To make the search faster, we implemented root parallelization (Cazenave and Jouandeau, 2007) in MCTS with 4 parallel trees giving a budget of 5000 iterations to each. This implementation is available in our C++14 lightweight MCTS library[3].

3.5.4 Guiding MCTS using A* on a simplified problem

MCTS traditionally samples actions randomly. To speed up the process, we first discretize the space and create a grid map; then, we simulate a virtual point robot with 8 actions (one for each neighboring cell — allowing diagonal moves) and solve the path planning problem using A*. Solving this simplified task requires very little computation. We use the optimized path to calculate an approximate desired direction for the next MCTS action. Next, we sample N (100 in our case) random actions from the repertoire and return the one that best matches this direction. Note that we are using the prediction of the GPs to decide which action we should choose. This simple procedure has the desirable effect of reducing the running time of MCTS (less than $40 - 50\,s$ to choose the next action), without sacrificing the quality of the returned actions. We use this "trick" because our problem is path-planning, but similar tricks can be used in other problems. More generic approaches would be the *Blind Value* action sampling or the continuous Rapid Action Value Estimation (cRAVE) (Couetoux et al., 2011).

3.5.5 Experimental results

A damaged velocity-controlled differential drive robot (the right wheel's velocity command is halved) has to reach 30 random equidistant sequential targets in an arena with an obstacle in the middle (Fig. 3.5). The scenario is replicated 50 times for statistics.

We count the number of episodes (100 steps of simulation with the same velocity commands) required by the different algorithms to reach each target.

[3] https://github.com/resibots/mcts

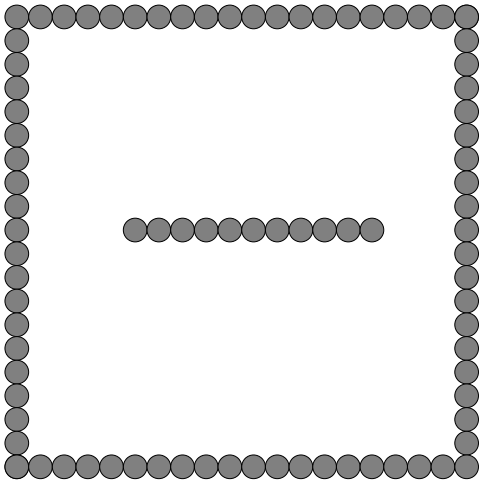


Figure 3.5: The environment used for the mobile robot task (800 × 800 units). The radii of the robot and the obstacles are the same (20 units).

The results show that RTE requires significantly fewer episodes (22.28 episodes, 25^{th} and 75^{th} percentiles [21.4, 22.9]) to reach each target than the re-planning baseline (32.12 episodes, [30.17, 34.97]) and GP-TEXPLORE (26.03 episodes, [25.37, 26.9]) (Fig. 3.6).

Further analysis shows that the median number of episodes to reach each target decreases over time (until it reaches a steady value) when the robot uses RTE or GP-TEXPLORE, whereas it stays constant with MCTS alone (Fig. 3.7). Furthermore, after the first target RTE is able to correct its repertoire and outperforms GP-TEXPLORE although the latter is capable of planning in the full action space.

Table 3.1: Recovered locomotion capabilities - Mobile Robot Task

Intact	**RTE**	**GP-TEXPLORE**	**MCTS**	**Recovered capabilities**		
Episodes per target				**RTE**	**GP-TEXPLORE**	**MCTS**
14.08	22.28	26.03	32.12	63.20%	54.10%	43.85%

We also performed the following evaluation test. We use the repertoire created by MAP-Elites with the intact robot and solve the same scenario (using MCTS as the planner — no model learning, no variance). We replicate the scenario 50 times and take the median number of episodes required to reach a target. We then compute the percentage of the recovered capabilities using RTE, GP-TEXPLORE and MCTS-based planning. The results show that RTE recovers more locomotion capabilities than GP-TEXPLORE (Table 3.1); RTE is able to recover around 63% of the original capabilities, whereas GP-TEXPLORE only recovers around 54%. Using only the repertoire generated with MAP-Elites and planning with MCTS is even worse, leading to only around 44% of recovered capabilities. These results justify (1) that the repertoire itself is not enough for the robot to recover its abilities and (2) that using prior

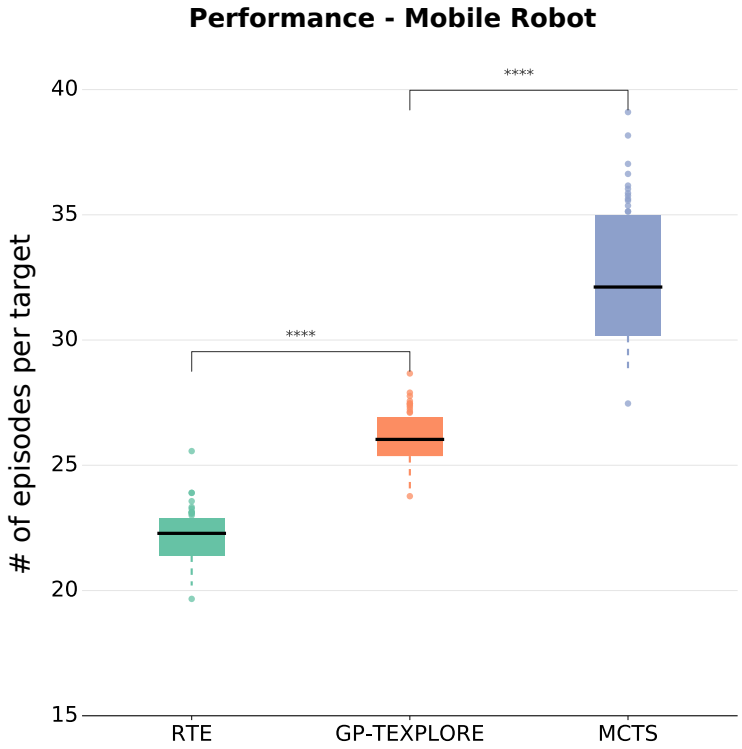


Figure 3.6: Comparison between RTE, GP-TEXPLORE and MCTS-based planning — Differential drive robot simulation results. A damaged velocity-controlled differential drive robot (the right wheel's velocity command is halved) has to reach 30 random equidistant sequential targets. We replicated the scenario 50 times. RTE significantly outperforms (lower is better) both the re-planning baseline and GP-TEXPLORE. The number of stars indicates that the p-value of the Mann-Whitney U test is less than 0.05, 0.01, 0.001 and 0.0001 respectively.

information (i.e., the repertoire) combined with learning (RTE) is beneficial compared to learning from scratch (GP-TEXPLORE).

Finally, we observed that in this simple task, RTE and GP-TEXPLORE produce fairly similar paths, with the ones produced by RTE being slightly safer (i.e., not too close to the obstacles — Fig. 3.8). In addition, both RTE and GP-TEXPLORE produce faster and safer paths than the MCTS baseline (Fig. 3.8). We also observed that the MCTS baseline often got stuck at the walls of the arena.

3.6 Hexapod Robot Results

Each leg of the hexapod robot that we used in our experiments has 3 degrees of freedom (DOF). This makes a total of 18-DOF for the whole robot. Nevertheless, since we are focusing on a path planning task, the state of the robot we are interested in consists of the (x, y) position and the yaw angle θ of the center of mass (COM) of the robot, i.e., $\boldsymbol{x}_{hexa} = [x, y, \theta]$. The hexapod robot task and the simple mobile robot task share the same experimental setup and parameters, with the main differences between them being the following:

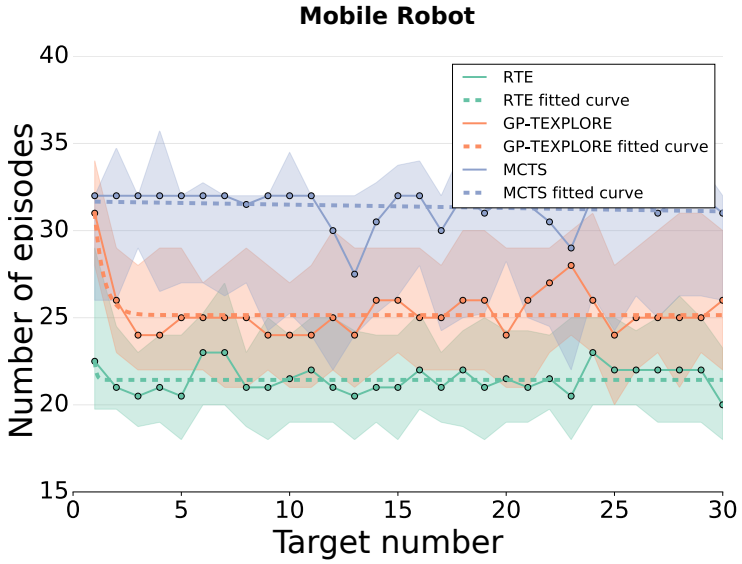


Figure 3.7: Median number of episodes to reach each target for a typical run of the algorithm for the mobile robot task. Over time, the robot using RTE or GP-TEXPLORE is able to reduce the number of required episodes to reach the next target (bottom lines), whereas MCTS alone uses a constant number of episodes (top line). Furthermore, RTE is able to correct its repertoire after the first target and outperforms GP-TEXPLORE, although the latter is capable of planning in the full action space. Most of the variance comes from the fact that the random targets are equidistant, but not of the same difficulty. The thick lines represent the medians over 50 runs and the shaded regions the 25^{th} and 75^{th} percentiles.

- In order to produce periodic gaits for the hexapod, we do not control the robot in joint space, but use a low-level controller (Sec. 3.6.1).
- The reachable space bounds for MAP-Elites are $[-2, 2]$ meters and we set $l = 0.03$ for the exponential kernel for the GP regression. In addition, to avoid depending on a specific repertoire, we ran MAP-Elites twice for 100000 evaluations, leading to two distinct repertoires with about 1500 different actions each (Fig. 3.9). The hexapod is simulated using the *DART* simulator (Lee et al., 2018).

3.6.1 Parametric Low-level Controller

The low-level controller is the same as in (Cully et al., 2015). It is intentionally kept simple, so that this paper can focus on the learning algorithm. The angular position of each degree of freedom is governed by a periodic function Γ parametrized by its amplitude v, its phase ϕ, and its duty cycle τ (the duty cycle is the proportion of one period in which the joint is in its higher position). This function is a square signal of frequency 1Hz, amplitude v, and duty cycle τ. A Gaussian filter is applied on the signal in order to remove sharp transitions, and it is then shifted according to the phase ϕ. The position of the third joint of each leg is the opposite of the position of the second one, so that the last segment is always vertical. This results in 36 real-valued parameters. Different values for these parameters can produce diverse gaits, from purely quadruped

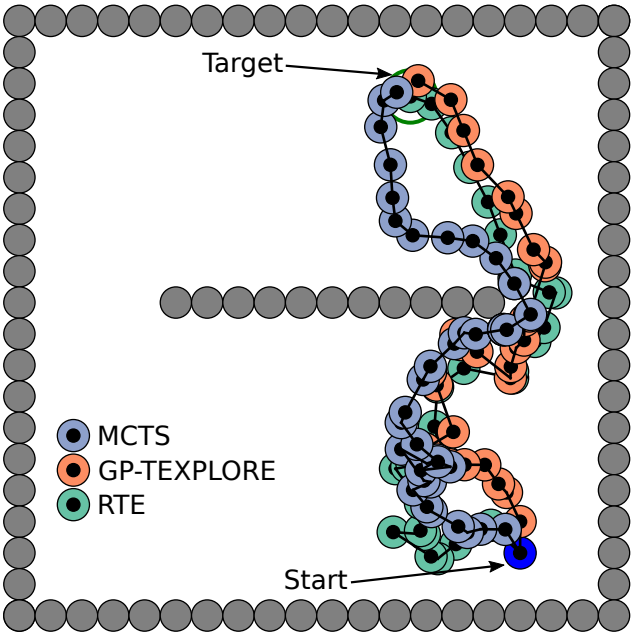


Figure 3.8: Sample trajectories of RTE, GP-TEXPLORE and MCTS in the mobile robot task. In this simple task, RTE and GP-TEXPLORE do not differ a lot (although RTE produces *safer* and slightly faster paths — Fig.3.6) and produce higher performing paths than the MCTS baseline.

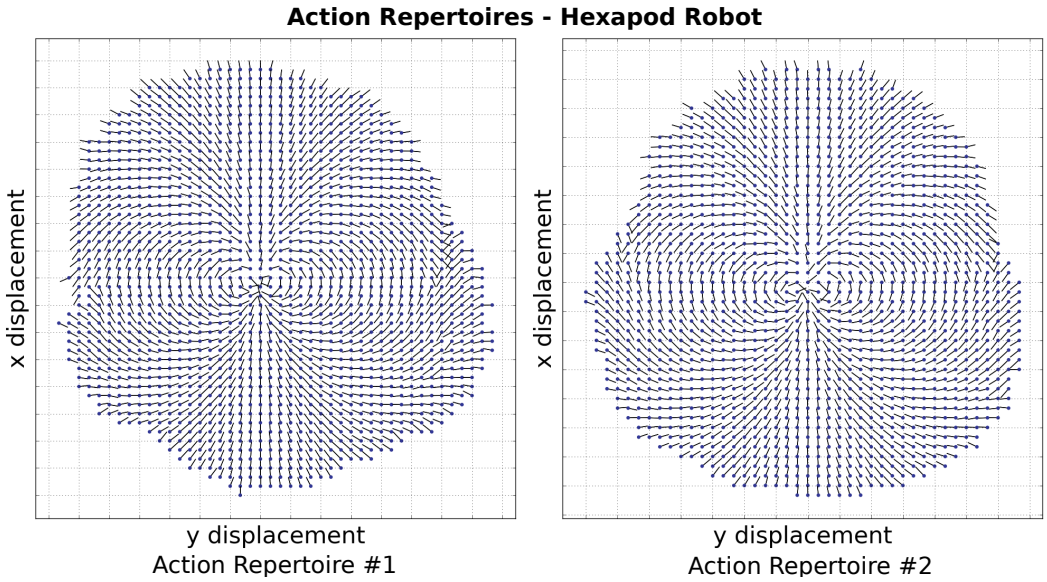


Figure 3.9: Repertoires for hexapod locomotion produced by the MAP-Elites algorithm. These repertoires map the 2-D action descriptor (of the 3-D task space) to the 36-D controller space. Each dot represents a different action (and its x, y position), while the lines indicate the orientation of the robot at the end of each behavior. All the behaviors are relative to the zero position that is located in the middle of the figures and relative to the forward orientation (line pointing up).

gaits to classic tripod gaits. At each *episode* of the learning algorithm, the low-level controller is executed for 3 seconds with the specified parameters (for

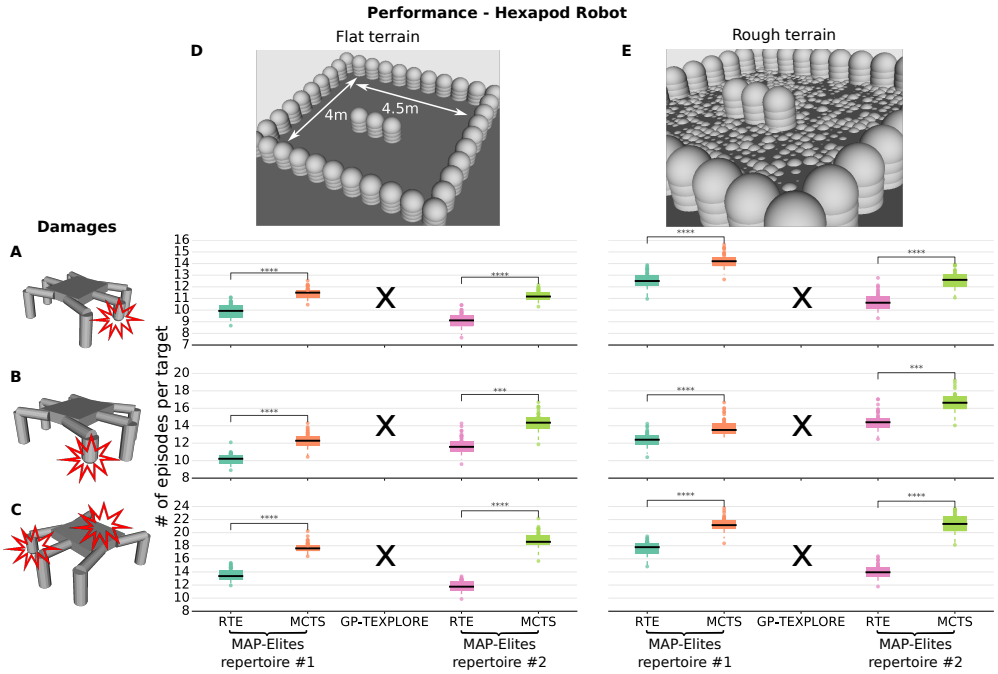


Figure 3.10: Comparison between RTE, GP-TEXPLORE and MCTS-based planning — Hexapod robot simulation results. We investigate 3 different kinds of damage (**A** - middle leg shortening, **B** - back leg shortening, **C** - back leg shortening and middle leg removal), 2 different environments (**D** and **E**) and 2 different action repertoires. We replicated each scenario 50 times. The task is to reach 30 random equidistant sequential targets (distance of 3.5 m). RTE outperforms the re-planning baseline (lower is better). GP-TEXPLORE was not able to solve the task. The number of stars indicates that the p-value of the Mann-Whitney U test is less than 0.05, 0.01, 0.001 and 0.0001 respectively.

all algorithms).

3.6.2 Simulation results

We count the number of episodes (3s actions) required to sequentially reach 30 equidistant (distance of 3.5 m) random targets. We investigate 3 different types of damage, 2 different environments (one with flat terrain and one with rough terrain), and 2 different action repertoires (Fig. 3.10). Each scenario is replicated 50 times for statistics.

The results show that RTE requires significantly fewer episodes to reach each target than the re-planning baseline (Fig. 3.10). Interestingly, RTE is able to reach the target points in the rough terrain scenario even though the action repertoire is learned on a flat terrain. This illustrates the capacity of RTE to compensate for unforeseen situations (i.e., damage and unmodeled terrain). Nevertheless, we observe slightly deteriorated performance and bigger differences between the MAP-Elites archives. This of course makes sense as the repertoires might have converged to different families of behaviors or one of them might be over-optimized for the flat terrain.

On the other hand, GP-TEXPLORE was not able to solve the task: with a budget of around 600 total episodes (due to computation time of GPs), it

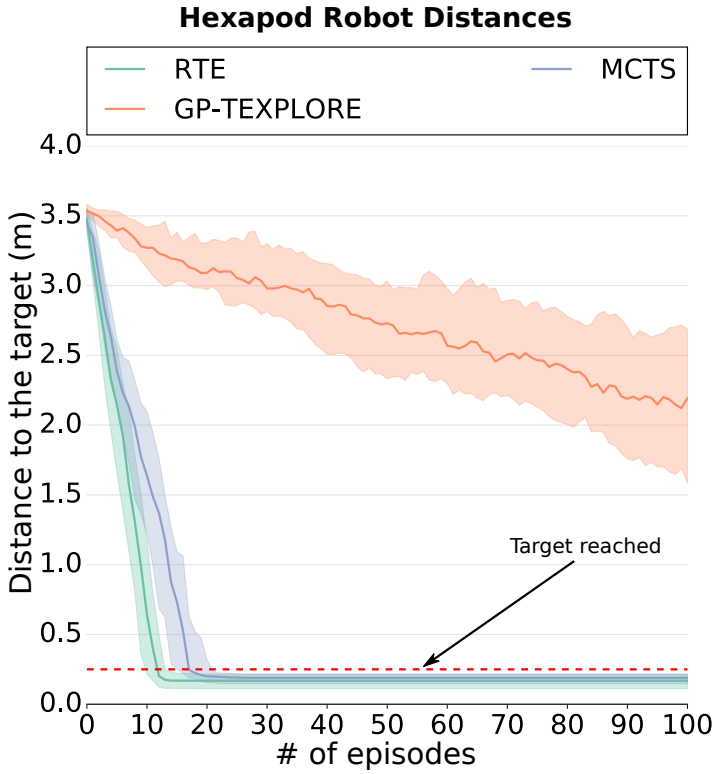


Figure 3.11: Comparison between RTE, GP-TEXPLORE and MCTS-based planning — Hexapod robot simulation results. We measure the distance to the 5th target of RTE and GP-TEXPLORE as the number of episodes increases for the damage in Fig. 3.10C, environment #1 (Fig. 3.10D) and the second repertoire. RTE clearly outperforms GP-TEXPLORE and the re-planning baseline; the robot with RTE reaches the target in about 10 episodes, whereas with MCTS it needs more than 20 episodes and with GP-TEXPLORE is not able to reach the target even after 100 episodes. The lines represent medians over 50 runs and the shaded regions the 25^{th} and 75^{th} percentiles.

did not succeed in reaching a target (the robot would be reset to the next target position every 100 episodes). This is because learning a full dynamics model of a complex robot cannot be done with a few samples (i.e., less than 1000-2000 (Droniou et al., 2012)).

The results show that as the number of episodes increases, the robot that uses GP-TEXPLORE gets closer to the target, but cannot reach it when provided with a budget of 100 episodes (Fig. 3.11). On the contrary, the robot with RTE reaches the target in a small number of episodes (around 10 episodes in Fig. 3.11). Moreover, the robot that uses MCTS (the re-planning baseline) is still able to reach the target, but requires more episodes (around 20 episodes in Fig. 3.11). These results show that the pre-computed repertoire breaks the complexity of the problem and makes it tractable, but refining the repertoire is essential for damage recovery (or to handle the reality gap as illustrated below).

Further analysis shows that the median number of episodes to reach each target decreases over time when the robot uses RTE, whereas it stays constant with MCTS alone (Fig. 3.12). After the first few targets (2-4), RTE is able to make the robot reach each target in around $30s$ compared to MCTS alone that

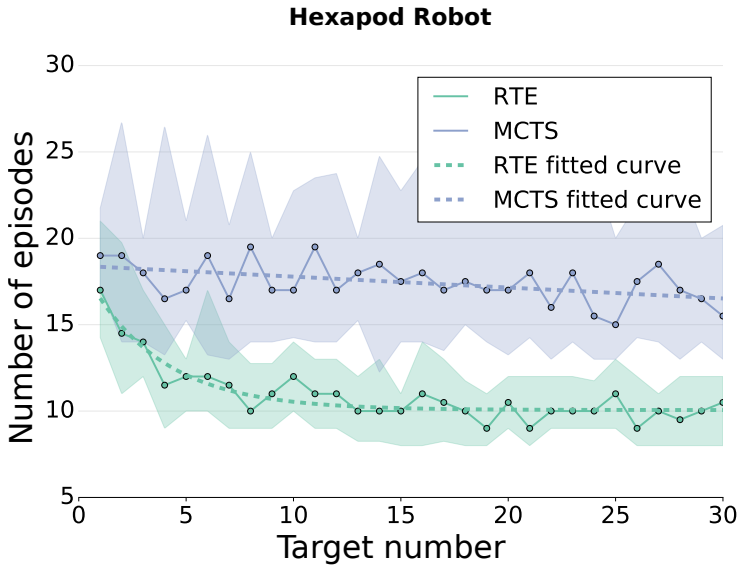


Figure 3.12: Median number of episodes to reach each target for a typical run of the algorithm in the hexapod task (in simulation — for damage in Fig. 3.10C, environment #1 Fig. 3.10D and the second action repertoire). Over time, the robot using RTE is able to reduce the number of required episodes to reach the next target (bottom line), whereas MCTS alone uses a constant number of episodes (top line). Most of the variance is due to the random targets being equidistant, but not of the same difficulty. The thick lines represent the medians over 50 runs and the shaded regions the 25^{th} and 75^{th} percentiles.

needs around $50 - 60\,s$.

Table 3.2: Recovered locomotion capabilities - Hexapod Robot Task (Flat terrain scenarios)

Flat terrain scenarios		**Intact**	**RTE**	**MCTS**	**Recovered capabilities**	
		Episodes per target			**RTE**	**MCTS**
Repertoire #1	Damage 1 (Fig. 3.10A)	7.70	9.93	11.5	77.52%	66.96%
	Damage 2 (Fig. 3.10B)		10.22	12.28	75.37%	62.69%
	Damage 3 (Fig. 3.10C)		13.37	17.6	57.61%	43.75%
Repertoire #2	Damage 1 (Fig. 3.10A)	7.12	9.12	11.17	78.10%	63.76%
	Damage 2 (Fig. 3.10B)		11.58	14.35	61.44%	49.59%
	Damage 3 (Fig. 3.10C)		11.75	18.6	60.57%	38.26%

We also use the repertoire created by MAP-Elites with the intact robot to solve the same additional scenarios that were presented in the mobile robot case. We replicate the scenarios 50 times and take the median number of episodes required to reach a target. We then compute the percentage of the recovered capabilities using RTE and MCTS-based planning for all the damage conditions in the flat and the rough terrain environments. The results show that RTE is able to almost always recover more than 60% of the original capabilities (see Tables 3.2 and 3.3). These results are consistent with both the flat and the rough terrain evaluations. This demonstrates the robustness and the capacity of our approach to adapt to unforeseen situations. In addition,

Table 3.3: Recovered locomotion capabilities - Hexapod Robot Task (Rough terrain scenarios)

Rough terrain scenarios		Intact	RTE	MCTS	Recovered capabilities	
		Episodes per target			RTE	MCTS
Repertoire #1	Damage 1 (Fig. 3.10A)	9.23	12.5	13.02	73.84%	65%
	Damage 2 (Fig. 3.10B)		12.4	13.52	74.44%	68.29%
	Damage 3 (Fig. 3.10C)		17.78	21.15	51.90%	43.64%
Repertoire #2	Damage 1 (Fig. 3.10A)	8.55	10.63	12.60	80.41%	67.86%
	Damage 2 (Fig. 3.10B)		14.4	16.63	59.38%	51.40%
	Damage 3 (Fig. 3.10C)		13.95	21.33	61.29%	40.10%

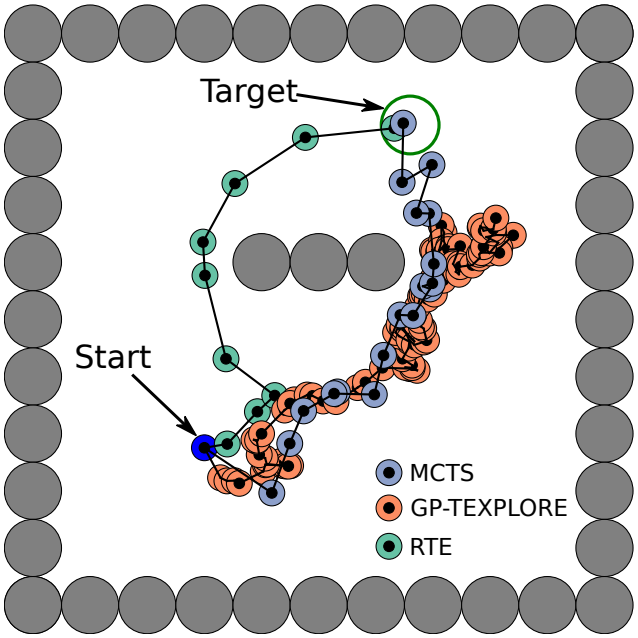


Figure 3.13: Sample trajectories of RTE, GP-TEXPLORE and MCTS in the simulated hexapod robot task. RTE produces faster and safer (i.e., not too close to the obstacles) paths than the MCTS baseline. GP-TEXPLORE cannot reach the target within a budget of 100 episodes, but does get closer to the target (as validated in Fig.3.11).

using the repertoire alone with MCTS planning is not enough for the robot to recover its capabilities as in half of the scenarios it fails to recover more than 60% of the original capabilities and always recovers less than RTE.

Finally, we observed that RTE produces paths that are faster and safer than the MCTS baseline (Fig. 3.13). While GP-TEXPLORE cannot reach the target, it does get closer to the target point as the number of episodes increases (Fig. 3.13 and Fig. 3.11). It is worth noting that GP-TEXPLORE takes actions that produce small displacements. This is probably due to the fact that the transition model cannot be accurately learned with a few data points, owing to the high dimensionality (36D) of the action space. As a consequence, the MCTS planner chooses actions that have already been selected. Since the search space is big and the first actions are selected almost randomly (there is

Figure 3.14: Physical damaged hexapod robot. The middle right leg is removed.

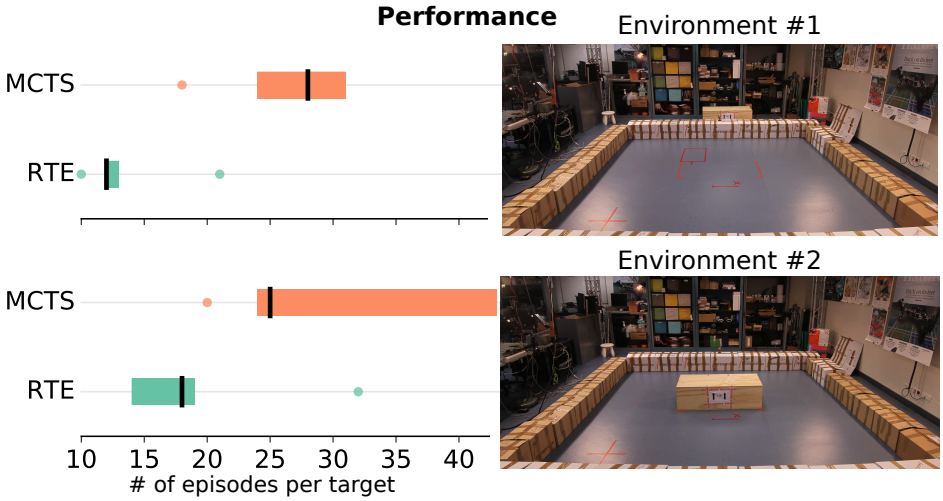


Figure 3.15: Comparison between RTE and MCTS-based planning — Physical hexapod robot experiments. We investigate 1 damage, 2 different environments and 1 action repertoire. We replicated each scenario 5 times. The task is to reach 10 and 5 random equidistant ($2\sqrt{2}\,m$) sequential targets for the environment #1 and #2 respectively. RTE needs on average between 1.39 and 2.33 times fewer episodes to reach each target. The results are statistically significant (Mann-Whitney U test $p < 0.05$).

no previous information), it is highly unlikely that taking these actions will actually lead to meaningful behaviors.

3.6.3 Physical robot results

We then evaluate RTE on the physical robot with a single damage (right middle leg removed — Fig. 3.14), in two environments (with and without a central obstacle) and the first action repertoire; the robot is required to reach 10 and 5 targets for each environment respectively, and the distance between the targets is $2\sqrt{2}\,m$. Each scenario is replicated 5 times. The environment (location of the

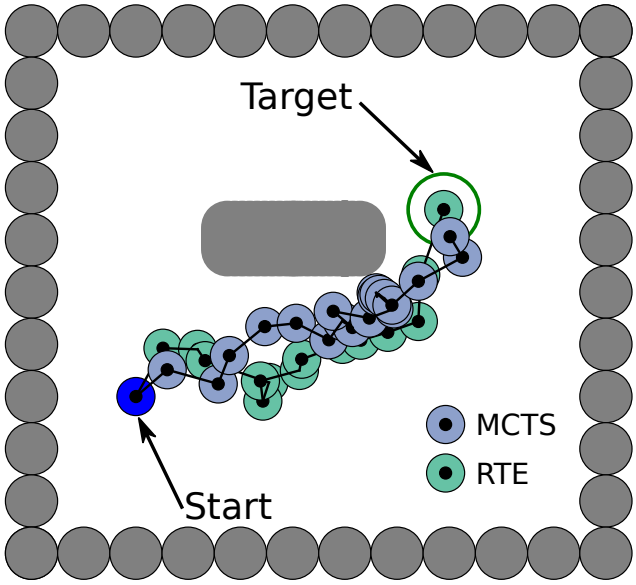


Figure 3.16: Sample trajectories of RTE and MCTS in the physical hexapod robot task. RTE comes up with faster and safer paths than the MCTS baseline to reach the goal point. The robot with the MCTS baseline tended to get stuck in the obstacle and struggle to get out of it and continue.

obstacles) and the robot are tracked with an external motion capture system (Optitrack).

The results show that RTE needs fewer episodes to reach each target (Environment 1: 13.0 episodes, 25^{th} and 75^{th} percentiles $[12.0, 14.0]$, Environment 2: 18.0 episodes, $[14.0, 19.0]$) than MCTS alone (Environment 1: 28.0 episodes, $[24.0, 31.0]$, Environment 2: 25.0 episodes, $[24.0, 43.0]$) (Fig. 3.15). These results are consistent with the simulations, but learning makes a bigger difference in the physical robot case. This is because the algorithm has to deal with the reality gap in addition to the damage in the physical robot case. Finally, as in the simulated experiments, RTE produces safer and faster paths than the MCTS baseline (Fig. 3.16). The robot with the MCTS baseline tended to get stuck in the obstacle and struggle to get out of it and continue. A demonstration of our approach on the real robot is available at `https://youtu.be/IqtyHFrb3BU`.

3.7 Conclusion and Discussion

With robots, like with many complex systems, "we should not wonder *if* some mishap may happen, but rather ask *what* one will do about it when it occurs" (Corbato, 2007). This advice is especially important if we want to be able to send advanced and expensive robots into dangerous places like a destroyed nuclear plant (Guizzo, 2011), even with tele-operated robots. In such situations, a damaged robot would greatly benefit from last-resort algorithms that would allow it to come back to its operators.

The RTE learning algorithm makes it possible for robots to overcome such

failures without the need for resets and human intervention. We successfully tested it on a simple mobile robot and on a hexapod robot that were damaged in several ways. Unlike most previous work, our algorithm does not require the robot to be returned to the same position after each trial: the robot learns autonomously, while taking into account its environment (obstacles). To our knowledge, this learning algorithm is one of the first algorithms that allows a physical legged robot to learn to walk without any human intervention, especially when there are obstacles.

The main limitation of RTE is that it chooses the optimal action for the damaged robot *among the actions that were found offline* with a different model. As a consequence, it is very likely that there exist better actions for the damaged robot in the full controller space, but RTE cannot use them. Nonetheless, this approximation seems to be sufficient in our experiments (i.e., the robot was able to complete its tasks) and it is one of the reasons why RTE scales significantly better than traditional RL approaches. In addition, it seems possible to periodically analyze the data collected (*e.g.*, once a day), update the original simulation, and re-generate the repertoire.

It is important to highlight that RTE is not a policy search method, like for example PILCO (Deisenroth et al., 2015): RTE uses an approximate planner (MCTS) to derive a policy given the current model which, in turn, allows the robot to collect samples from the environment, refine the model, and thereby improve the policy, that is, the planner. Thus, the online phase of RTE can be seen as an on-policy, model-based RL procedure. In addition, the first phase of RTE (MAP-Elites), learns "elementary behaviors" (actions) in simulation, which are similar to parametrized policies or movement primitives (Ijspeert et al., 2003). Nevertheless, the first phase of RTE does not only do that, but also creates a mapping from the high-dimensional controller space to the lower-dimensional task space, which proves to be beneficial when dealing with complex robots.

Ultimately, RTE should run continuously on the robot to compensate for potential wear or damage, that is, it should be a continuous learning, rather than a damage recovery, algorithm. However, the current version has a bottleneck: the computational complexity of the prediction of the GPs is quadratic in the number of samples, which prevents the robot from using more than a few hundred episodes. A potential solution is reducing the query time of GPs by using a time-window and/or using sparse GPs (Quiñonero-Candela and Rasmussen, 2005) or local GPs (Park and Apley, 2017). Another solution is to replace the GPs with neural networks and take advantage of the recent advances in neural networks with uncertain predictions (Gal and Ghahramani, 2016).

In these first experiments, we assumed that the robot had perfect knowledge of its position and of the environment, which made it possible to cast our problem to an MDP. The next step is to relax this assumption and let the robot discover its environment with a SLAM algorithm (Durrant-Whyte and Bailey, 2006). In this case, we could look at the problem from two different perspectives: (1) still treat the problem as an MDP and take into account the

uncertainty of the map in the planning phase (MCTS), (2) treat the problem as a POMDP (Partially Observable MDP) and try to solve it with MCTS (Silver and Veness, 2010). The first perspective might not be enough to solve the problem (i.e., the robot would struggle to execute good plans), whereas the second one will increase the computation time of MCTS.

Furthermore, here we assumed that the outcome of each action is independent of the state it was taken in, which is the case for mobile robots when (1) the robot can be stopped to take a decision and (2) the terrain does not change dramatically. Nevertheless, RTE was able to cope with cases where this assumption did not really hold; in particular, the hexapod was able to walk on rough terrain, even though the action repertoire was optimized for flat terrain. In future work, we will look at this in greater depth, and try to relax these assumptions. For example, we could produce priors that are state-dependent and learn the full transition model and/or the reward function.

In this work, we chose to use MCTS for the planning phase of our approach, because it has been successfully used in the context of RL (Silver and Veness, 2010; Browne et al., 2012; Hester and Stone, 2013) and because it makes no assumptions about the dynamics/model of the system. This allows us to incorporate prior knowledge about the problem (Silver et al., 2016) and to use actions of any type, as we did in our work. Nevertheless, traditional sample-based planners, like RRT, could provide more accurate solutions and/or be faster in some cases. In future work, we will investigate and experiment with different probabilistic planners.

Lastly, while we performed our experiments with a legged and a mobile robot, the algorithm introduced here is general enough to be extended to many other robots and tasks. For instance, it could be employed on an arm mounted on a mobile platform that had incurred damage (*e.g.*, a blocked joint). In this case, the algorithm will learn a mapping between the (x,y,z) position of the end-effector and the joint/wheel positions, similarly to how it learned a mapping between the (x,y) position of the hexapod robot and the parametric controller. After each trial, the robot might be in a different position relative to the target object (*e.g.*, a door knob), but thanks to RTE, it should not have to go back to its starting position to try a different behavior.

Chapter 4

Flexible and Fast Model-Based Policy Search Using Black-Box Optimization

The results and text of this chapter have been partially published in the following articles.

Articles:

- **Chatzilygeroudis, K.**, Rama, R., Kaushik, R., Goepp, D., Vassiliades, V. and Mouret, J.-B., 2017. *Black-Box Data-efficient Policy Search for Robotics*. **IEEE/RSJ International Conference on Intelligent Robots and Systems** (Chatzilygeroudis et al., 2017).
- Kaushik, R., **Chatzilygeroudis, K.**, and Mouret, J.-B., 2018. *Multi-objective Model-based Policy Search for Data-efficient Learning with Sparse Rewards*. **International Conference on Robot Learning** (Kaushik et al., 2018).

Other contributors:

- Roberto Rama (Master student)
- Rituraj Kaushik (PhD student)
- Dorian Goepp (Engineer)
- Vassilis Vassiliades (Post-doc)
- Jean-Baptiste Mouret (Thesis supervisor)

Author contributions:

- For the IROS paper: **KC** and JBM organized the study. **KC**, RR and RK wrote the code. **KC** and RR performed the simulated experiments. **KC** and DG performed the physical robot experiments. **KC**, VV and JBM analyzed the results and wrote the paper.
- For the CoRL paper: RK and JBM organized the study. RK and **KC** wrote the code. RK performed the experiments. RK, **KC** and JBM analyzed the results and wrote the paper.

4.1 Introduction

In the background chapter, we realized that there exist two main strategies for tackling the challenge of micro-data RL: *leveraging prior knowledge* and *building surrogate models*. In this chapter, we will focus on the second strategy and introduce a new *model-based policy search* approach that is flexible, data-efficient, and fast[1].

When data are scarce, a general principle is to extract as much information as possible from them. In the case of robotics, this means that all the state variables that are available should be collected at every time-step and be used by the learning algorithm. This contrasts with many direct policy search approaches (*e.g.*, policy gradient algorithms (Deisenroth et al., 2013; Kohl and Stone, 2004) or Bayesian optimization (Cully et al., 2015; Calandra et al., 2015; Lizotte et al., 2007)) which only use the (cumulative) reward at the end of each episode.

One of the best ways to take advantage of this sequential state recording is to learn a dynamical model of the robot (Nguyen-Tuong and Peters, 2011), and then exploit it either for model-predictive control (Camacho and Alba, 2013) or to find an optimal policy offline (Deisenroth et al., 2013). However, such approaches assume that the model is "good enough" to predict future states for all the possible states. This is often not the case when only a few episodes have been performed, as many states have not been observed yet. Learning with a dynamical model therefore often requires acquiring enough points to learn an accurate model, which, in turn, increases the interaction time.

This challenge can be overcome by taking into account the uncertainty of the dynamical model: if the algorithm "knows" that a prediction is unreliable, it can balance the risks of trying something that might fail with the potential benefits. The PILCO (Probabilistic Inference for Learning COntrol) algorithm (Deisenroth et al., 2015), which is one of the state-of-the-art algorithms for data-efficient model-based policy search, follows this strategy by alternating between two steps, (1) learning a dynamical model with Gaussian processes (Rasmussen and Williams, 2006), (2) using a gradient-based optimizer to search for a policy that maximizes the expected reward, taking the uncertainty of the model into account. Thanks to this process, PILCO achieves remarkable data-efficiency; for instance it can find a good policy for the cart-pole swing-up in around 4 trials of 4 seconds (Deisenroth et al., 2015), whereas classic algorithms for RL in continuous states (*e.g.*, CACLA (Hasselt and Wiering, 2007)) typically require more than 50 trials.

Nevertheless, analytical algorithms like PILCO have two main issues that may not be apparent at first sight. First, they impose several constraints on the reward functions and policies that prevent the use of arbitrary rewards (*e.g.*, PILCO can only be used with distance-based rewards so far) and of non-derivable policies (*e.g.*, parameterized state automata, like in (Calandra et al., 2015)). Second, they require a large *computation time* to optimize the

[1] More specifically, we can exploit multi-core architectures to speed up the computations.

policy (*e.g.*, typically more than 5 minutes on a modern computer between each episode for the cart-pole benchmark), because they rely on computationally expensive methods to do approximate inference for each step of the policy evaluation (Deisenroth et al., 2015).

In this chapter, we introduce a novel policy search algorithm that tackles these two problems while maintaining the data-efficiency of analytical algorithms. Our main insight is that while the analytic approach is efficient on a sequential computer, it cannot take advantage of the multi-core architectures now present in every computer. By contrast, Monte Carlo approaches and population-based black-box optimizers like CMA-ES (Hansen and Ostermeier, 2001) (1) do not put any constraint on the reward functions and policies, and (2) are straightforward to parallelize, which can make them competitive with analytical approaches when several cores are available. Our second insight is that it is not necessary to explicitly (or fully) compute accurate approximations of the expected reward when the optimization is performed with rank-based algorithms designed for noisy functions (*e.g.*, CMA-ES (Hansen and Ostermeier, 2001)), which saves a lot of computation: only the ranking of potential solutions matters. Thus, it is possible to define a data-efficient, black-box policy search algorithm that is competitive with gradient-based, analytical approaches.

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

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

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

4.2 Problem Formulation

Here, we adapt the generic problem formulation of Section 2.2 to our specific case. We consider dynamical systems of the form:

$$\boldsymbol{x}_{t+1} = \boldsymbol{x}_t + f(\boldsymbol{x}_t, \boldsymbol{u}_t) + \boldsymbol{w} \tag{4.1}$$

with continuous-valued states $\boldsymbol{x} \in \mathbb{R}^E$ and controls $\boldsymbol{u} \in \mathbb{R}^F$, i.i.d. Gaussian system noise $\boldsymbol{w} \sim \mathcal{N}(0, \Sigma_w)$, and unknown transition dynamics f.

In this chapter, we will focus on *deterministic policies*, and thus our objective is to find a deterministic policy π, $\boldsymbol{u} = \pi(\boldsymbol{x}|\boldsymbol{\theta})$, which maximizes the *expected long-term reward* (see Eq. (2.4)). We assume that π is a function parameterized by $\boldsymbol{\theta} \in \mathbb{R}^\Theta$ and that the immediate reward function $r(\boldsymbol{x}) \in \mathbb{R}$ (see Eq. (2.3)) might be unknown to the learning algorithm.

4.3 Approach

The Black-DROPS algorithm relies on the model-based policy search framework as defined in Sec. 2.7.1. Thus, it consists of two steps that are being alternated after each policy is executed on the system: (a) a model learning step where the unknown dynamics model, f, and possibly the immediate reward function, r, are learned, and (b) an optimization step where the optimal policy according to the learned models is searched.

In our approach, we exploit population, rank-based optimizers, and in particular CMA-ES (Hansen, 2006) variants, in order to combine the policy evaluation with the optimization procedure. In particular, we exploit the *implicit averaging* property that these algorithms possess (Jin and Branke, 2005; Miller and Goldberg, 1996) in order to efficiently perform sampling based evaluation of the trajectories. Our key idea is to formulate model-based policy search under uncertain dynamics as a noisy optimization problem. Viewing it likes this allows us to use any of the well-known and tested optimizers that are robust to noisy or stochastic objective functions, without the need of performing many Monte-carlo rollouts.

4.3.1 Learning the dynamics model

Assuming $\boldsymbol{D}_{1:n} = \{f(\tilde{\boldsymbol{x}}_1), ..., f(\tilde{\boldsymbol{x}}_n)\}$ is a set of observations and $\tilde{\boldsymbol{x}}_t = (\boldsymbol{x}_t, \boldsymbol{u}_t) \in \mathbb{R}^{E+F}$, the goal of the model learning step is to learn a mapping, $\hat{f}$, from $\tilde{\boldsymbol{x}}_t$ to $f(\tilde{\boldsymbol{x}}_t) = \boldsymbol{x}_{t+1} - \boldsymbol{x}_t$; a possible second goal would be to learn a mapping, $\hat{r}$, from $\boldsymbol{x}_t$ to $r(\boldsymbol{x}_t)$ if the immediate reward function needs to be learned. As we have already discussed in the background section, Deisenroth et al. (Deisenroth et al., 2015, 2013) first showcased that probabilistic models can be more effective than deterministic ones, because the learning algorithms can get information about the regions where the model is certain about its predictions and where it is not.

PILCO (Deisenroth and Rasmussen, 2011) exploits the Gaussian properties of GPs in order to derive an analytical formulation for the policy evaluation

step (see Sec. 2.7.1.2 and Sec. 4.3.2). In Black-DROPS, the underlying model does not necessarily have to be a probabilistic one nor of a specific type (*e.g.*, a GP), since Black-DROPS treats the model as a black box. In this chapter, we are using Gaussian processes, like PILCO, for the models, and in the next chapter we will be combining Black-DROPS with an open-loop periodic policy. Overall, in Black-DROPS the user can choose the model type that best fits their use case.

4.3.2 Policy Evaluation

Our goal is to maximize the expected cumulative reward (Eq. (2.4)), which requires predicting the state evolution given an uncertain transition model. This distribution, in our case, is given by:

$$\hat{P}(\boldsymbol{\tau}|\boldsymbol{\theta}) = p(\boldsymbol{x}_0) \prod_t \hat{p}(\boldsymbol{x}_{t+1}|\boldsymbol{x}_t, \boldsymbol{u}_t)\boldsymbol{u}_t \tag{4.2}$$

where $\boldsymbol{u}_t = \pi(\boldsymbol{x}_t|\boldsymbol{\theta})$ and $\hat{p}(\boldsymbol{x}_{t+1}|\boldsymbol{x}_t, \boldsymbol{u}_t)$ is given by the approximated model of the dynamics.

To do so in a deterministic way[2], PILCO proposes to approximate the distribution of state $\boldsymbol{x}_{t+1}$ given the distribution of state $\boldsymbol{x}_t$ and the action $\boldsymbol{u}_t$ using moment matching (Deisenroth et al., 2015), and then propagates from state to state until reaching the end of the episode. However, this sequential approach accumulates errors over time (due to the approximations made at each time-step), is not easy to parallelize, and is computationally expensive (Kupcsik et al., 2017). As an alternative, we can compute a Monte Carlo approximation of the final distribution: at each step we sample a new state according to the GP of the model and its reward according to the reward model, query the policy to choose the actions, and use this new state to sample the next state. By performing this process many times, we can get a good estimate of the expected cumulative reward, but many samples are needed to obtain a good estimate (Kupcsik et al., 2017; Deisenroth et al., 2013) (see Sec. 2.7.1.2 for more details).

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

$$\begin{aligned} G(\boldsymbol{\theta}) &= J(\boldsymbol{\theta}) + N(\boldsymbol{\theta}) \\ &= \sum_{t=1}^{T} \hat{r}(\boldsymbol{x}_{t-1} + \hat{f}(\boldsymbol{x}_{t-1}, \boldsymbol{u}_{t-1})) \end{aligned} \tag{4.3}$$

where $\hat{f}(\boldsymbol{x}_{t-1}, \boldsymbol{u}_{t-1}) \sim \mathcal{N}(\mu_{\hat{f}}(\tilde{\boldsymbol{x}}_{t-1}), \Sigma_{\hat{f}}(\tilde{\boldsymbol{x}}_{t-1}))$ is a realization of a normally distributed random vector according to Eq. 2.45, $\hat{r}(\boldsymbol{x}) \sim \mathcal{N}(\mu_r(\boldsymbol{x}), \sigma_r^2(\boldsymbol{x}))$ is a

[2]PILCO also requires the reward function to be known a priori, whereas Black-DROPS does not make this assumption.

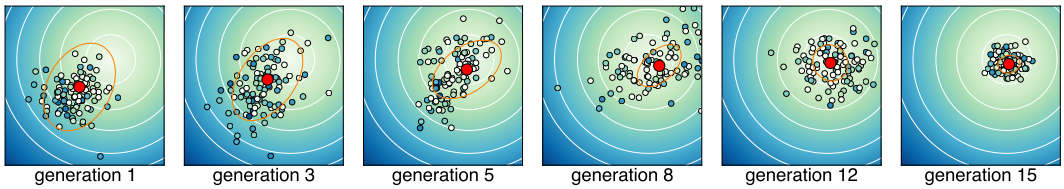


Figure 4.1: Illustration of an actual run with CMA-ES on a simple 2-D, noisy problem. The performance landscape is pictured on the background. Each colored disk is a candidate from the population (the color represents its performance with the same color scale as the landscape in the background). If the color of a disk is the same as the background, then noise did not change the performance. The mean m_k of the best μ candidates is pictured as a red disk, and the covariance of the μ best individuals as an orange ellipse. In only a few generations, and in spite of the noise, CMA-ES identifies the optimum of the function. Please note that in this example we use a bigger population than in our work for illustration purposes. In Black-DROPS, we use small populations as advised by the authors of CMA-ES (Hansen and Ostermeier, 2001).

realization of a normally distributed random value according to Eq. 2.43 and $\boldsymbol{u}_{t-1} = \pi(\boldsymbol{x}_{t-1}|\boldsymbol{\theta})$.

In order to maximize the expected return, $J(\boldsymbol{\theta})$, we need to maximize the expectation of the "perturbed" function $G(\boldsymbol{\theta})$:

$$\begin{aligned}\mathbb{E}\Big[G(\boldsymbol{\theta})\Big] &= \mathbb{E}\Big[J(\boldsymbol{\theta}) + N(\boldsymbol{\theta})\Big] \\ &= \mathbb{E}\Big[J(\boldsymbol{\theta})\Big] + \mathbb{E}\Big[N(\boldsymbol{\theta})\Big]\end{aligned} \tag{4.4}$$

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

$$\mathbb{E}\Big[G(\boldsymbol{\theta})\Big] = J(\boldsymbol{\theta}) + \mathbb{E}\Big[N(\boldsymbol{\theta})\Big] \tag{4.5}$$

We assume that $\mathbb{E}[N(\boldsymbol{\theta})] = 0$ for all $\boldsymbol{\theta} \in \mathbb{R}^{\Theta}$ (see Eq. (4.1)) and therefore maximizing $\mathbb{E}[G(\boldsymbol{\theta})]$ is equivalent to maximizing $J(\boldsymbol{\theta})$.

4.3.3 Policy search with CMA-ES

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: we only use the noisy measurements in the optimization algorithm. To do so, we use the Covariance Matrix Adaptation Evolutionary Strategy (CMA-ES) (Hansen and Ostermeier, 2001), which is one of the most successful evolutionary algorithms for optimizing noisy and black-box functions (Jin and Branke, 2005; Hansen et al., 2009; Hansen, 2009). CMA-ES performs four steps at each generation k (Fig. 4.1):

(1) sample λ new candidates according to a multi-variate Gaussian distribution of mean m_k and covariance $\sigma_k^2 C_k$, that is, $\boldsymbol{\theta}_i \sim \mathcal{N}(m_k, \sigma_k^2 C_k)$ for $i \in 1, \cdots, \lambda$;

(2) rank the λ sampled candidates based on their (noisy) performance $G(\boldsymbol{\theta}_i)$;

(3) compute m_{k+1} using a weighted average of the μ best candidates: $m_{k+1} = \sum_{i=1}^{\mu} w_i \boldsymbol{\theta}_i$, where $\sum_{i=1}^{\mu} w_i = 1$;

(4) update the covariance matrix to reflect the distribution of the successful search steps.

Overall, these steps are only marginally impacted by noise in the performance function, as confirmed by empirical experiments with noisy functions (Jin and Branke, 2005). More precisely, the only decision that matters is whether a solution belongs to the μ best ones (step 2), that is, a precise ranking is not needed and errors can only happen at the boundaries between the low-performing and high-performing solutions. In addition, if a candidate is misclassified because of the noise, the impact of this error will be smoothed out by the average when computing m_{k+1} (step 3). One can also observe that because CMA-ES samples several solutions around a mean m_k, it performs many evaluations of similar parameters, which are then averaged: this *implicit averaging* (Jin and Branke, 2005; Miller and Goldberg, 1996) has many similarities with re-evaluating noisy solutions to estimate their expectation.

In the extreme case where the population is infinite, it can be shown that the selection part of evolutionary algorithms is not affected at all by noise (Miller and Goldberg, 1996). Moreover, it is shown that in genetic algorithms with ranking-based selection, adding random perturbations to the design variables at each generation is equivalent to optimizing the expected fitness function (Tsutsui and Ghosh, 1997).

In the Black-DROPS algorithm we ulitize a recent variant of CMA-ES that uses a similar scheme and combines random pertubations with re-evaluation for uncertainty handling (Hansen et al., 2009) along with restart strategies for better exploration (Auger and Hansen, 2005). In particular, we use active IPOP-CMA-ES with restarts (Auger and Hansen, 2005) and we follow the strategy proposed by Hansen et al. (2009) to improve the behavior of CMA-ES with noisy functions (called UH-CMA-ES). The starting idea is that uncertainty is a problem for a rank-based algorithm if and only if, for two potential candidates $\boldsymbol{\theta}_1$ and $\boldsymbol{\theta}_2$ the variation due to $N(\boldsymbol{\theta}_1)$ and $N(\boldsymbol{\theta}_2)$ exceeds the difference $|J(\boldsymbol{\theta}_1) - J(\boldsymbol{\theta}_2)|$ and thus their ordering is changed. If the variation tends to exceed this difference, we cannot conclude only from two measurements $G(\boldsymbol{\theta}_1)$, $G(\boldsymbol{\theta}_2)$, whether $J(\boldsymbol{\theta}_1) > J(\boldsymbol{\theta}_2)$ or $J(\boldsymbol{\theta}_1) < J(\boldsymbol{\theta}_2)$ holds. If we view $|J(\boldsymbol{\theta}_1) - J(\boldsymbol{\theta}_2)|$ as the signal and the variations due to $N(\boldsymbol{\theta})$ as noise, then it follows that one way to improve the quality of the ranking without re-evaluating solutions many times (which would reduce noise) is to increase the signal.

We therefore implement the following strategy: (1) at each generation, we quantify the uncertainty of the ranking by re-evaluating $\lambda_{\text{reev}} < \lambda$ randomly selected candidates from the population and count the number of rank changes (see (Hansen et al., 2009) for a detailed description of uncertainty quantification), (2) if the uncertainty is above a user-defined threshold, then we increase the variance of the population (σ_k in step 1 of CMA-ES). In addition to reducing the uncertainty of the ranking when needed, this strategy has an interesting

Algorithm 11 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 LearnStrategy 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**

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

4.3.4 Handling noisy systems

The Black-DROPS algorithm, as we have described it so far, will struggle when the system noise, $\boldsymbol{w}$, is not negligible or when the initial state distribution $p(\boldsymbol{x}_0)$ is wide. This is because we sample only one trajectory (according to Eq. (4.2)) to get the noisy estimate of the expected return, $G(\boldsymbol{\theta})$. When the system noise (and thus the learned variance of the models) is significant or the initial distribution is wide, this estimate is too noisy for CMA-ES to be able to properly optimize its expectation; remember that we do not have infinite population nor infinite search generations in practical implementations (for the theoretical guarantees to hold; see Sec. 4.3.3).

A simple way to handle these cases is to increase the number of sampled rollouts (or trajectories) so that the estimate of the expected return is not too noisy and CMA-ES can properly maximize its expectation. Of course, we do not need to increase the number of samples as much as in pure Monte-carlo estimation, as we only need to reduce the variance of the estimation. We found 5 sampled rollouts to be sufficient for our experiments.

4.3.5 Black-box Data-efficient Robot Policy Search

The Black-DROPS algorithm follows the generic policy search algorithm (as defined in Algo. 1 and re-stated in Algo. 11) and the generic model-based policy search algorithm (as defined in Algo. 4), and implements InitStrategy, CollectStrategy, LearnStrategy and UpdateStrategy (Algo. 12). In our experiments, we evaluate a few random policies of the robot in order to gather some initial data for the models. We, then, collect samples of the form $(\boldsymbol{x}_t, \boldsymbol{u}_t) \rightarrow \boldsymbol{x}_{t+1}$ and $\boldsymbol{x}_t \rightarrow r(\boldsymbol{x}_t)$ according to Sec. 4.3.1 and learn the models appropriately (in this chapter we use GPs for the models). Lastly, we select

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

the policy to try next by optimizing $\mathbb{E}\big[G(\boldsymbol{\theta})\big]$ using CMA-ES with uncertainty handling (Sec. 4.3.3) and the learned models.

Algorithm 12 Black-DROPS

procedure InitStrategy
  Select $\boldsymbol{\theta}_{init}$ randomly ▷ We could also evaluate multiple random policies
end procedure
procedure CollectStrategy
  Collect samples of the form $(\boldsymbol{x}_t, \boldsymbol{u}_t) \rightarrow \boldsymbol{x}_{t+1}$
  Collect samples of the form $\boldsymbol{x}_t \rightarrow r(\boldsymbol{x}_t)$
end procedure
procedure LearnStrategy
  Learn model $\hat{f} : (\boldsymbol{x}_t, \boldsymbol{u}_t) \rightarrow \boldsymbol{x}_{t+1} - \boldsymbol{x}_t$
  Learn model $\hat{r} : \boldsymbol{x}_t \rightarrow r(\boldsymbol{x}_t)$ ▷ if necessary
end procedure
procedure UpdateStrategy
  $\boldsymbol{\theta}_{n+1} = \text{argmax}_{\boldsymbol{\theta}} \mathbb{E}\big[G(\boldsymbol{\theta}|\hat{f}, \hat{r})\big]$
end procedure

4.4 Experimental Setup

4.4.1 Policy Representations

To highlight the flexibility of Black-DROPS, we use a GP-based policy (Deisenroth et al., 2015) and a feed-forward neural network-based one. We chose to evaluate Black-DROPS with the GP policy in order to have a fair comparison to PILCO; *i.e.*, use exactly the same scenarios and parameters. We chose the neural network policy as this is most commonly used in RL works (Polydoros and Nalpantidis, 2017; Deisenroth et al., 2013; Kober et al., 2013), and because it does not provide any real prior information about the underlying task (as opposed to Dynamical Movement Primitives for example).

Nevertheless, any other parameterized policy can be used with Black-DROPS. For example, in the next chapter we will use an open-loop parameterized policy to control a six-legged robot.

4.4.1.1 GP Policy

If we only consider the mean, a Gaussian process can be used to map states to actions, that is, to define a policy (this is identical to the PILCO paper (Deisenroth et al., 2015)):

$$\begin{aligned} \pi(\boldsymbol{x}) &= \boldsymbol{u}_{\max} \kappa_{\text{squash}}(\mu_{\text{policy}}(\boldsymbol{x})) \\ &= \boldsymbol{u}_{\max} \kappa_{\text{squash}}(\boldsymbol{k}_{\text{policy}}^T (K_{\text{policy}} + \sigma_n^2 I)^{-1} \boldsymbol{x}) \end{aligned} \tag{4.6}$$

where $\boldsymbol{u}_{\max}$ is the maximum value of $\boldsymbol{u}$ (can be different for each action dimension), $\kappa_{\text{squash}} \in [-1, 1]$ is a squashing function to keep the actions selected

in bounds, $\boldsymbol{x}$ is the input state vector to the policy, $\boldsymbol{k}_{\text{policy}}$ and K_{policy} are computed using the exponential kernel (Rasmussen and Williams, 2006) and we set $\sigma_n = 0.01$. The hyperparameters of this GP and the pseudo-observations (*inputs* and *targets*) constitute the parameters of the policy. We define the squashing function as follows (following Deisenroth et al. (2015)'s suggestion):

$$\kappa_{\text{squash}}(\boldsymbol{x}) = \frac{9}{8}\sin(\boldsymbol{x}) + \frac{1}{8}\sin(3\boldsymbol{x}) \tag{4.7}$$

4.4.1.2 Neural Network Policy

The network function of the i^{th} layer of the network is given by $\boldsymbol{y}_i = \phi_i(\boldsymbol{W}_i\boldsymbol{y}_{i-1} + \boldsymbol{b}_i)$, where $\boldsymbol{W}_i$ and $\boldsymbol{b}_i$ are the weight matrix and bias vector, $\boldsymbol{y}_{i-1}$ and $\boldsymbol{y}_i$ are the input and output vector and ϕ_i is the activation function. In this chapter, we use configurations with one hidden layer and the hyperbolic tangent as the activation function ϕ for all the layers, leading to:

$$\begin{aligned} \pi(\boldsymbol{x}) = \boldsymbol{u}_{\max}\boldsymbol{y}_1 = \boldsymbol{u}_{\max}\phi(\boldsymbol{W}_1\boldsymbol{y}_0 + \boldsymbol{b}_1) \\ \text{and } \boldsymbol{y}_0 = \phi(\boldsymbol{W}_0\boldsymbol{x} + \boldsymbol{b}_0) \end{aligned} \tag{4.8}$$

4.4.2 Metrics

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

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

4.4.3 Experiments

We evaluate Black-DROPS on the pendulum and cart-pole tasks and compare it to PILCO using 80 replicates over different CPU configurations. As an additional baseline, we evaluate a variant of our approach using deterministic GP models of the dynamics (*i.e.*, using only the mean of the GPs) to quantify the importance of considering the uncertainty (variance) of the model in policy optimization. We replicate the experiments for both noiseless and noisy systems with mild noise (for the noisy settings, we use 5 rollouts). For Black-DROPS and the baseline we use two different policies: a neural network policy (with one hidden layer and 10 hidden units) and a GP policy (with 10 pseudo-observations). For PILCO we use only the GP policy with the same parameters as for the other algorithms. In both cases, we additionally compare PILCO and Black-DROPS (20 replicates) on different rewards to showcase the flexibility and robustness of our approach.

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

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

4.5 Results

4.5.1 Task 1: Inverted Pendulum

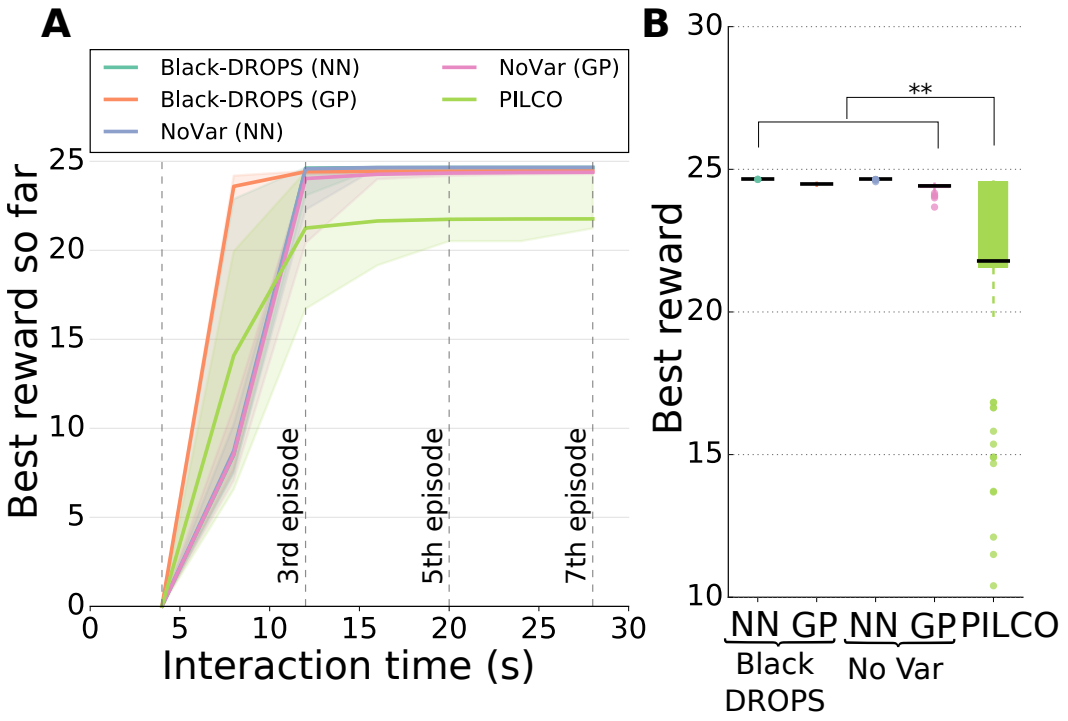


Figure 4.2: Results for the noiseless pendulum task with the saturating reward (80 replicates): (**A**) Best reward found per episode. The lines are median values and the shaded regions the 25^{th} and 75^{th} percentiles. Black-DROPS converges to higher quality solutions in fewer episodes than PILCO and has considerably less variance. (**B**) Best reward after 10 episodes. The box plots show the median (black line) and the interquartile range (25^{th} and 75^{th} percentiles); the whiskers extend to the most extreme data points not considered outliers, and outliers are plotted individually. Our approach outperforms PILCO in the quality of the controllers found. The number of stars indicates that the p-value of the Mann-Whitney U test is less than 0.05, 0.01, 0.001 and 0.0001 respectively.

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

- **State:** $\boldsymbol{x}_{pend} = [\dot{\theta}, \theta] \in \mathbb{R}^2$, $\boldsymbol{x}_0 = [0, 0]$.

- **Actions:** $\boldsymbol{u}_{pend} = u_{pend} \in \mathbb{R}$, $-2.5 \leq u_{pend} \leq 2.5\,N$.
- To avoid angle discontinuities, we transform the input of the GPs, the reward function, and the policy to be:

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

  The MATLAB implementation of PILCO uses this transformation by default[4].

- **Reward #1:** We use two different rewards. The first one is the same reward function as PILCO[5]. This is a saturating distance-based reward function:

$$r_{\text{sat}}(\boldsymbol{x}) = \exp(-\frac{1}{2\sigma_c^2}(\boldsymbol{x} - \boldsymbol{x}_*)^T \boldsymbol{Q}(\boldsymbol{x} - \boldsymbol{x}_*)) \tag{4.9}$$

  where σ_c controls the width of the reward function, $\boldsymbol{Q}$ is a weight matrix, $\boldsymbol{x}_*$ is the target state and $r_{\text{sat}}(\boldsymbol{x}) \in [0, 1]$. We set $\boldsymbol{x}_* = [*, \cos(\pi), \sin(\pi)]$, $\sigma_c = 0.5$ and $\boldsymbol{Q}$ to ignore the angular velocity $\dot{\theta}$ of the pendulum.

- **Reward #2:** The second reward is a more classic one: *i.e.*, a transformed distance to the goal:

$$r_{\text{quad}}(\boldsymbol{x}) = -(\boldsymbol{x} - \boldsymbol{x}_*)^T \boldsymbol{Q}(\boldsymbol{x} - \boldsymbol{x}_*) \tag{4.10}$$

  where $\boldsymbol{Q}$ is a weight matrix and $\boldsymbol{x}_*$ is the target state. We set $\boldsymbol{x}_* = [*, \cos(\pi), \sin(\pi)]$, and $\boldsymbol{Q}$ to ignore the angular velocity $\dot{\theta}$ of the pendulum.

- In the noisy setting, $p(\boldsymbol{x}_0) \sim \mathcal{N}(0, 0.01)$ and a small Gaussian measurement noise is applied to the states, $\sim \mathcal{N}(0, \begin{bmatrix} 0.01 & 0 \\ 0 & 0.0001 \end{bmatrix})$.

Table 4.1: Success Rates for Pendulum

Algorithm	**Success Rate**	
	Noiseless Pendulum	**Noisy Pendulum**
Black-DROPS (NN)	100%	98.73%
Black-DROPS (GP)	100%	100%
No Var (NN)	100%	97.5%
No Var (GP)	100%	90%
PILCO	80%	82.5%

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

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

4.5.1.1 Noiseless System Results

In the noiseless system, both Black-DROPS and the baselines solve the task in about 3 episodes (12 s of interaction time — including the random episode, Fig. 4.2**A**). In this simple and noiseless system, the baselines that do not take into account the uncertainty of the model perform as well as Black-DROPS (the "No Var" baselines perform the same as Black-DROPS, see Fig. 4.2). This result most probably stems from the fact that the dynamics of the system are simple enough for the GPs to model almost perfectly with one or two episodes. Interestingly, PILCO does not perform as well and finds sub-optimal policies. In addition, given a budget of 15 episodes, Black-DROPS succeeds more often than PILCO in finding a working policy (Table 4.1): Black-DROPS always solves the task whereas PILCO fails 20% of the time.

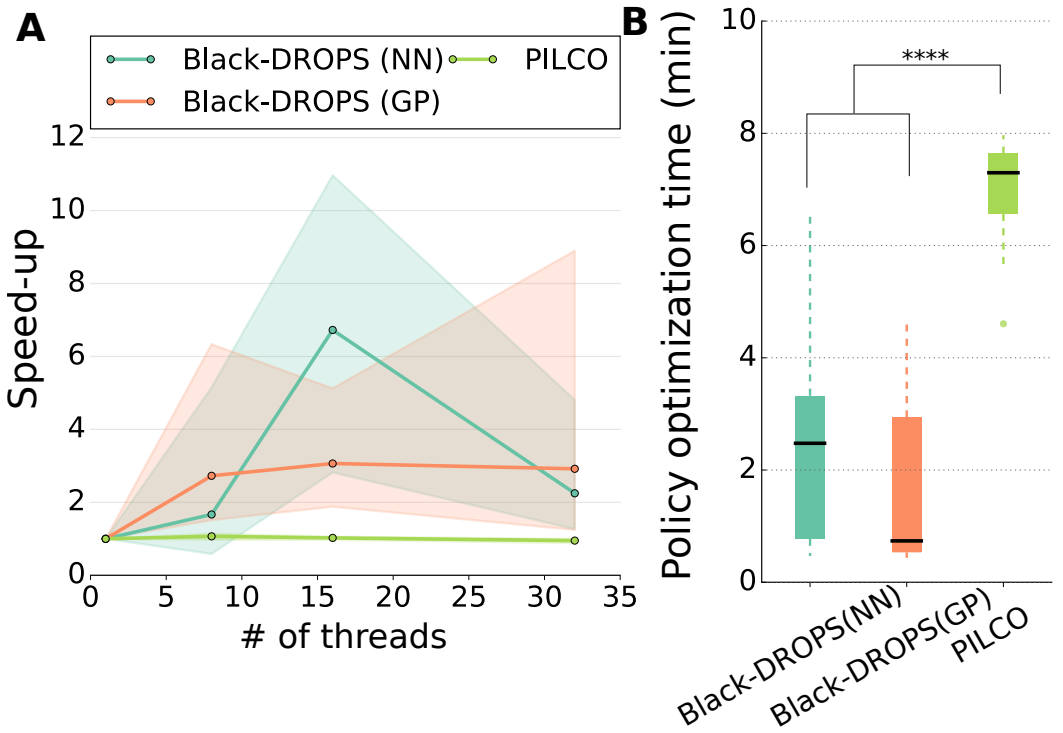


Figure 4.3: Timing for the the pendulum task with the saturating reward: (**A**) Speed-up (for total policy optimization time after 10 episodes) achieved when using multiple cores. The lines are median values over 30 runs and the shaded regions the 25^{th} and 75^{th} percentiles. As more threads are being used, Black-DROPS benefits from it and has up to 4x speed-up when 32 threads are used. (**B**) Total policy optimization time after 10 episodes when 32 threads are available.

When the number of threads is increased, the computation time required by Black-DROPS decreases and we can have a speed-up of up to 4x (in the cases that we considered — Fig. 4.3**A**), whereas PILCO does not benefit almost at all from having multiple threads. With more than 12 threads, Black-DROPS outperforms PILCO in computation speed and can be from 3 to 10 times faster when 32 threads are available[6] (Fig. 4.3**B**).

[6]While some of the runtime differences can stem from the language used (*e.g.*, C++

It is worth noticing that we are not having a linear increase in the speed-up when more threads are being used. This is mainly because of the following reasons:

- CMA-ES usually uses small population of candidate policies per generation and thus the speed-up that we can get saturates at around 16 cores (at least in this case);
- The problem is very low dimensional and CMA-ES converges in very few generations for the speed-up due to parallelization to be visible.

Moreover, the plots differ a bit from our IROS paper (Chatzilygeroudis et al., 2017), because we changed the CMA-ES variant from BIPOP-CMA-ES to IPOP-CMA-ES that converges faster, and overall provides a better tradeoff between the quality of the solutions and the convergence time.

4.5.1.2 Noisy System Results

In this setting, both Black-DROPS and PILCO solve the task in about 3 episodes (12 s of interaction time — including the random episode, Fig. 4.4**A**), but Black-DROPS finds higher-performing policies (Fig. 4.4**A-B**), with both the neural network and the GP policy. In this noisy system, using the variance helps more than the noiseless system: the variants of Black-DROPS without uncertainty handling are less data-efficient and have more variance. Additionally, Black-DROPS is able to achieve a success rate of more than 98%, whereas PILCO achieves only 90% (see Table 4.1).

In this noisy scenario, however, Black-DROPS requires more computational time than PILCO, even with 32 threads (that was the maximum number we could use). Overall, PILCO was from 1.2x to 2x faster than Black-DROPS. Nevertheless, this result is affected by many parameters, and most notably by:

1. Stopping criteria — in PILCO we used 75 gradient steps, whereas in Black-DROPS we let the CMA-ES population to converge;
2. Number of rollouts — we used 5 sampled trajectories, but depending on the noise level we could choose less or more and affect the convergence properties of CMA-ES.

In section 4.6, we explore one potential solution to this issue of Black-DROPS and get some promising preliminary results.

4.5.1.3 Quadratic Reward Results

Changing the reward to the negative distance to the target as per Eq. (4.10) does not affect Black-DROPS at all and we get very similar results (Fig. 4.5).

being faster than MATLAB or MATLAB being faster at matrix computations), what matters is that a parallel algorithm with enough CPUs can eventually outperform a sequential gradient-based approach.

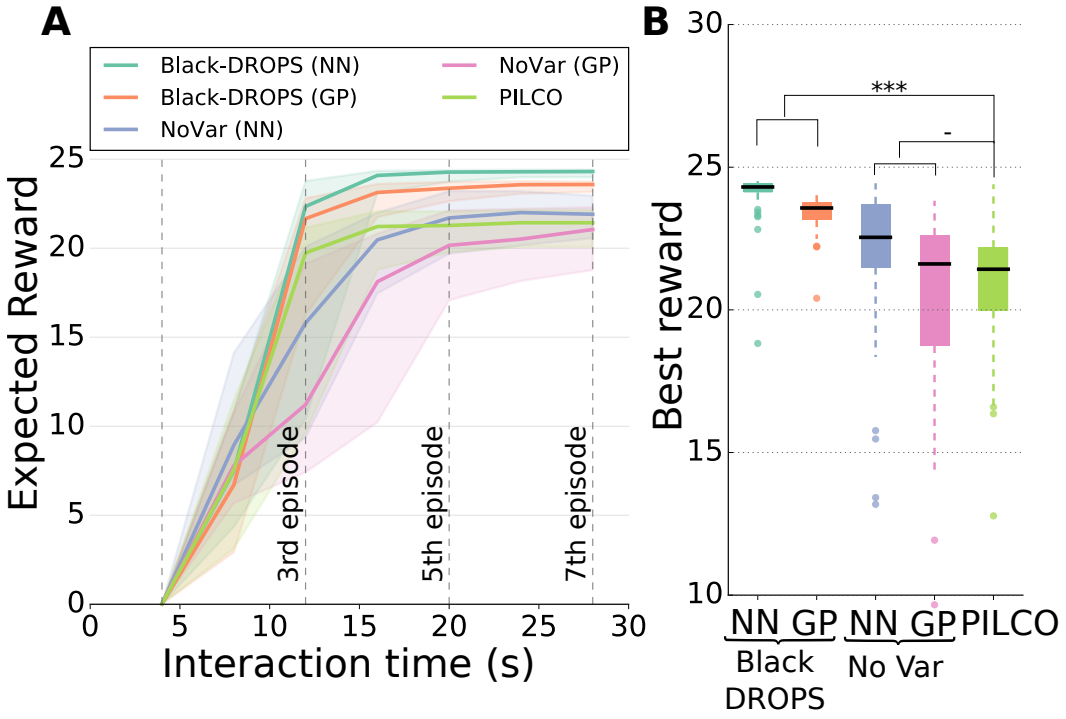


Figure 4.4: Results for the noisy pendulum task with the saturating reward (80 replicates): (**A**) Expected reward per episode; note that this metric is not available to the algorithm and only used for evaluation. Black-DROPS converges to higher quality solutions in fewer episodes than PILCO and has considerably less variance. (**B**) Expected reward after 10 episodes. Our approach outperforms PILCO in the quality of the controllers found. See Fig. 4.2 for legend.

Similarly, PILCO is not affected too much by the type of the reward function and achieves similar data-efficiency as with the saturating reward.

4.5.2 Task 2: Cart-pole Swing-Up

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

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

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

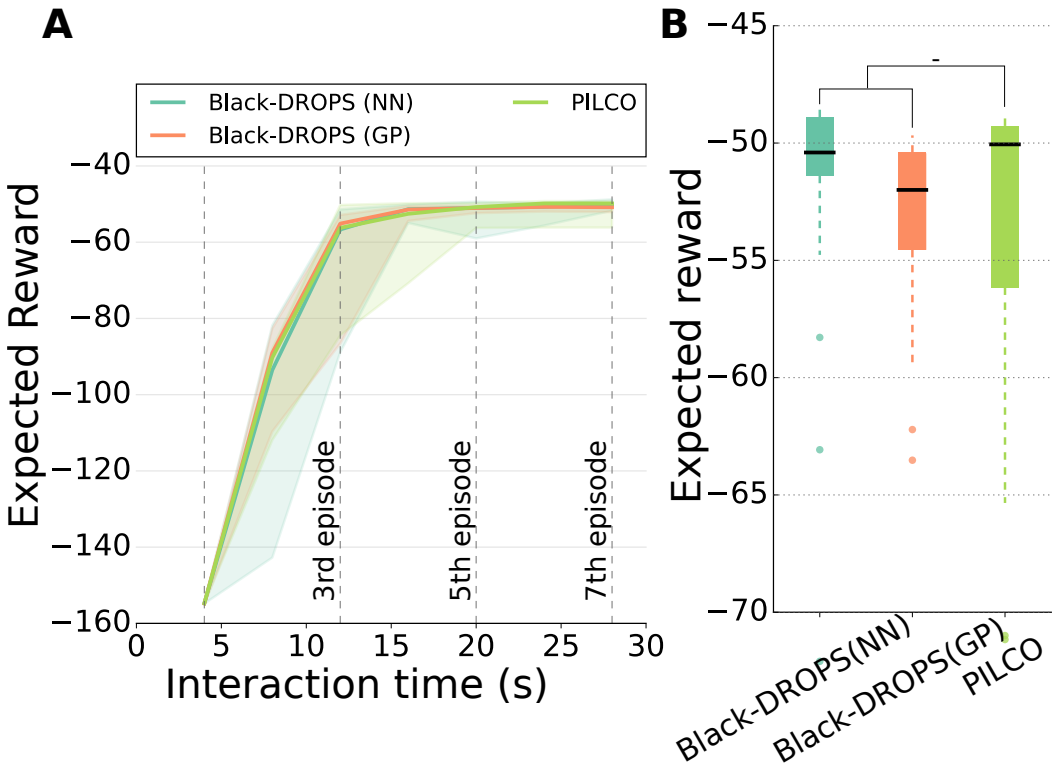


Figure 4.5: Results for the noisy pendulum task with the quadratic reward (20 replicates): (**A**) Expected reward per episode; note that this metric is not available to the algorithm and only used for evaluation. Black-DROPS and PILCO converge to high performing policies in similar interaction time compared with the smooth reward. (**B**) Expected reward after 10 episodes. See Fig. 4.2 for legend.

- **Reward #1:** We again use two reward functions. For the first one, we set $\boldsymbol{x}_* = [*, 0, *, \cos(\pi), \sin(\pi)]$, $\sigma_c = 0.25$, $\boldsymbol{Q}$ to ignore $\dot{x}$ and $\dot{\theta}$, and use Eq. 4.9.
- **Reward #2:** For the second one, we set $\boldsymbol{x}_* = [*, 0, *, \cos(\pi), \sin(\pi)]$, $\boldsymbol{Q}$ to ignore $\dot{x}$ and $\dot{\theta}$, and use Eq. 4.10.
- In the noisy setting, $p(\boldsymbol{x}_0) \sim \mathcal{N}(0, 0.01)$ and a small Gaussian measurement noise is applied to the states, $\sim \mathcal{N}(0, 0.0001)$.

Table 4.2: Success Rates for Cart-pole

Algorithm	**Success Rate**	
	Noiseless Cart-pole	**Noisy Cart-pole**
Black-DROPS (NN)	98.52%	98.75%
Black-DROPS (GP)	98.65%	98.5%
No Var (NN)	100%	3.75%
No Var (GP)	100%	1.25%
PILCO	92.5%	87.5%

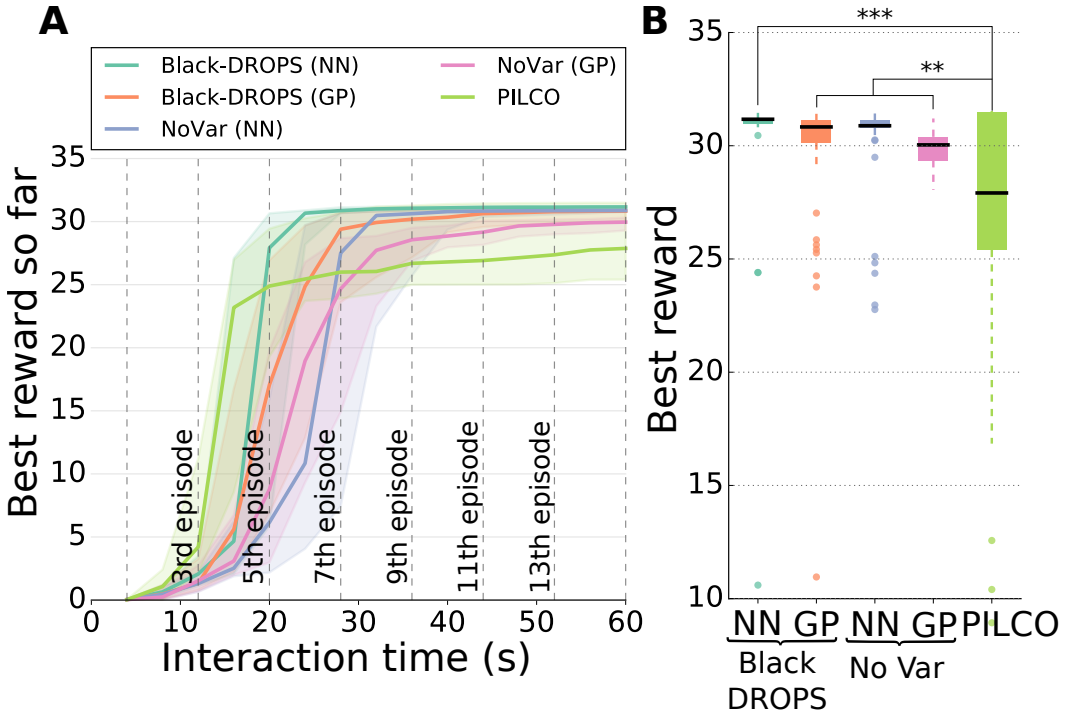


Figure 4.6: Results for the noiseless cart-pole task with the saturating reward (80 replicates): (**A**) Best reward found per episode. Black-DROPS converges to higher quality solutions in about the same number of episodes as PILCO and has less variance. (**B**) Best reward after 15 episodes. Our approach outperforms PILCO in the quality of the controllers found. See Fig. 4.2 for legend.

4.5.2.1 Noiseless System Results

In the noiseless setting, the results for the cart-pole are very similar to those obtained with the inverted pendulum (Fig. 4.6**A-B**): Black-DROPS and the variants without the variance perform similarly and solve the task in around 5 episodes (20 s of interaction time) to solve the task. Nevertheless, using the variance helps more in this task than in the pendulum task: the variants of Black-DROPS without uncertainty handling are a bit less data-efficient and have more variance. PILCO seems to struggle again with the narrow initial state distribution and improves slowly. Similar to the pendulum task, here also Black-DROPS takes advantage of multiple threads to highly speed-up its computation and is 1.6 times faster than PILCO when 32 threads are available (Fig. 4.7).

4.5.2.2 Noisy System Results

In this setting, both Black-DROPS and PILCO solve the task in about 4-5 episodes ($16 - 20\,s$ of interaction time — including the random episode, Fig. 4.8**A**), but Black-DROPS finds higher-performing policies (Fig. 4.8**A-B**) and with less variance, with both the neural network and the GP policy. In this noisy and more interesting system using the variance is crucial for finding effective controllers: the variants of Black-DROPS without uncertainty handling

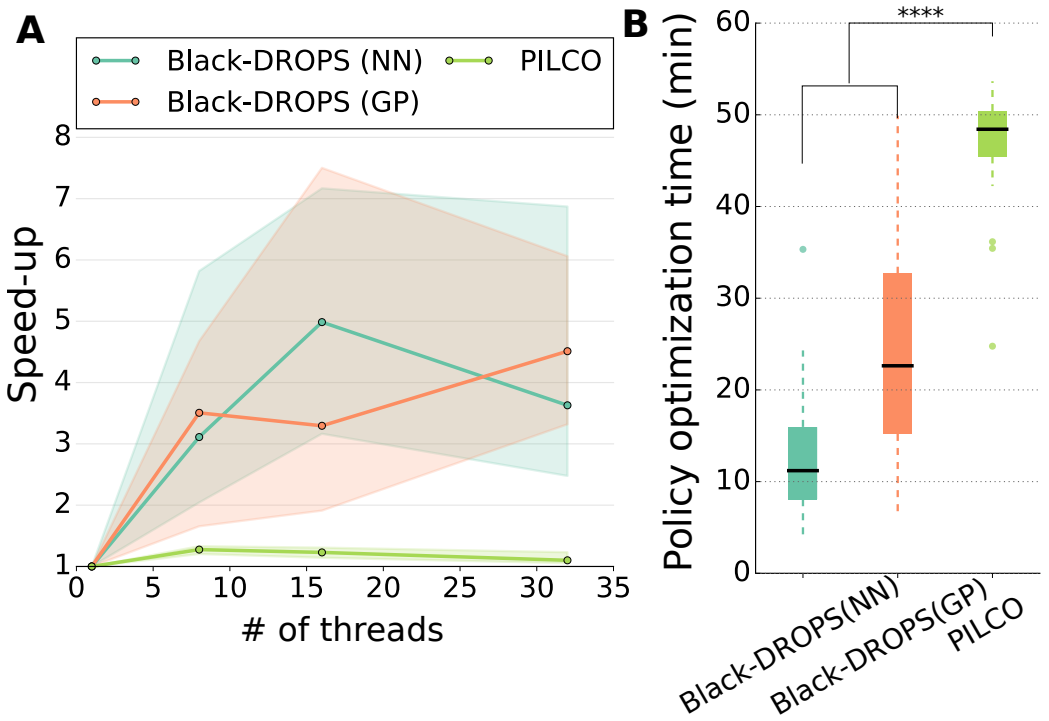


Figure 4.7: Timing for the cart-pole task with the saturating reward: (**A**) Speed-up (for total policy optimization time after 15 episodes) achieved when using multiple cores. As more cores are being used, Black-DROPS benefits from it and has a $4-6$x speed-up when 12 cores are used. (**B**) Total policy optimization time after 15 episodes when 32 threads are available. Black-DROPS is around 2.1 to 4.3 times faster than PILCO. See Fig. 4.3 for legend and number of replicates.

are not able to solve the task at all (Fig. 4.8**A**). Additionally, Black-DROPS is able to achieve a success rate of almost 100%, whereas PILCO achieves only 87.5% (see Table 4.2).

4.5.2.3 Quadratic Reward Results

Like in the pendulum case, changing the reward to the negative distance to the target as per Eq. (4.10) does not affect Black-DROPS at all and we get very similar results with the saturating reward (Fig. 4.9). Additionally, Black-DROPS with the neural network policy is able to find slightly better performing behaviors and with less variance than PILCO. Overall, Black-DROPS finds high performing solutions 1-2 episodes faster than PILCO.

4.5.3 Task 3: 4-DOF Manipulator

We, also, applied Black-DROPS on a physical velocity-controlled 4-DOF robotic arm (Fig. 4.10, 10 replicates). We assume that we can only observe the angles of the joints of the arm and that the reward function r_{arm} is initially unknown. The arm begins in the up-right position and the objective is to learn a controller

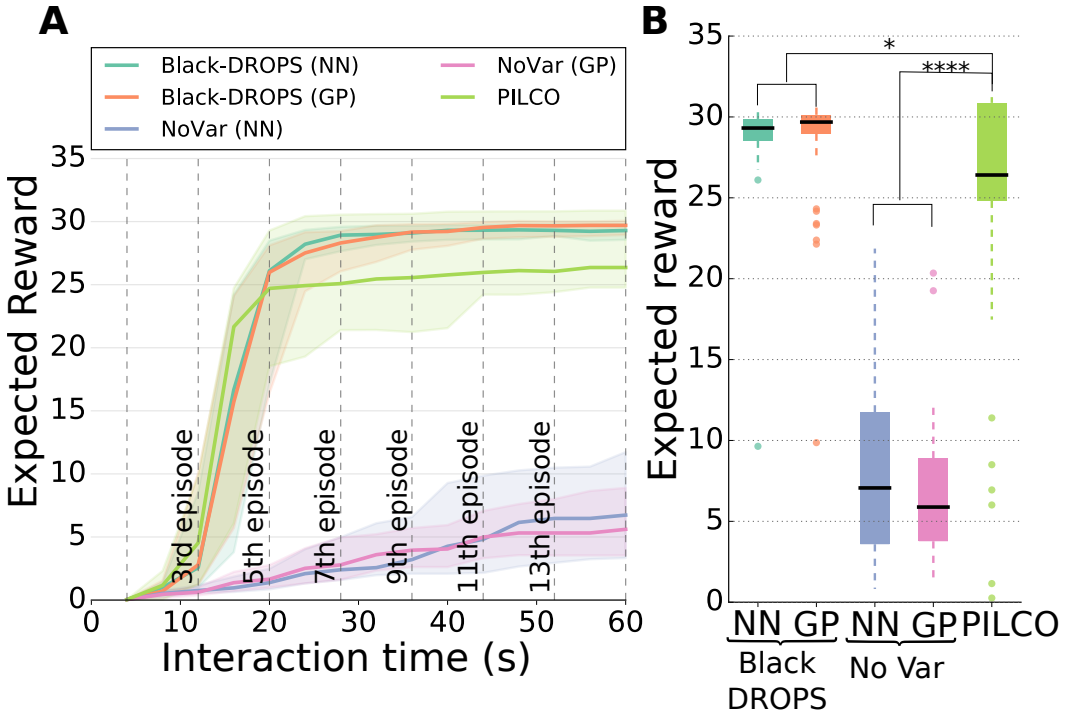


Figure 4.8: Results for the noisy cart-pole task with the saturating reward (80 replicates): (**A**) Expected reward per episode. Black-DROPS converges to higher quality solutions in about the same number of episodes as PILCO and has less variance. (**B**) Expected reward after 15 episodes. Our approach outperforms PILCO in the quality of the controllers found. See Fig. 4.2 for legend.

so that the end-effector quickly reaches a certain position (shown in Fig. 4.10**A**). We compare Black-DROPS with the baseline without variance.

- **State:** $\boldsymbol{x}_{arm} = [q_0, q_1, q_2, q_3] \in \mathbb{R}^4$, $\boldsymbol{x}_0 = [0, 0, 0, 0]$.
- **Actions:** $\boldsymbol{u}_{arm} = [v_0, v_1, v_2, v_3] \in \mathbb{R}^4$, where $-1.0 \leq v_i \leq 1.0\, rad/s$
- **Reward:** The unknown (to the algorithm) reward function has a form similar to Eq. 4.9:

$$r_{arm}(\boldsymbol{x}) = \exp(-\frac{1}{2\sigma_c^2}\|\boldsymbol{p}_{\boldsymbol{x}} - \boldsymbol{p}_*\|) \tag{4.11}$$

  where $\sigma_c = 0.1$, $\boldsymbol{p}_{\boldsymbol{x}}$ corresponds to the end-effector position in state $\boldsymbol{x}$, $\boldsymbol{p}_*$ is the goal position of the end-effector and $r_{arm}(\boldsymbol{x}) \in [0, 1]$.

- To avoid angle discontinuities, we transform the input to the GPs and the policy to be:

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

The results show that Black-DROPS is able to find a working policy within 5 episodes (including the initial random one) and outperforms the baseline which

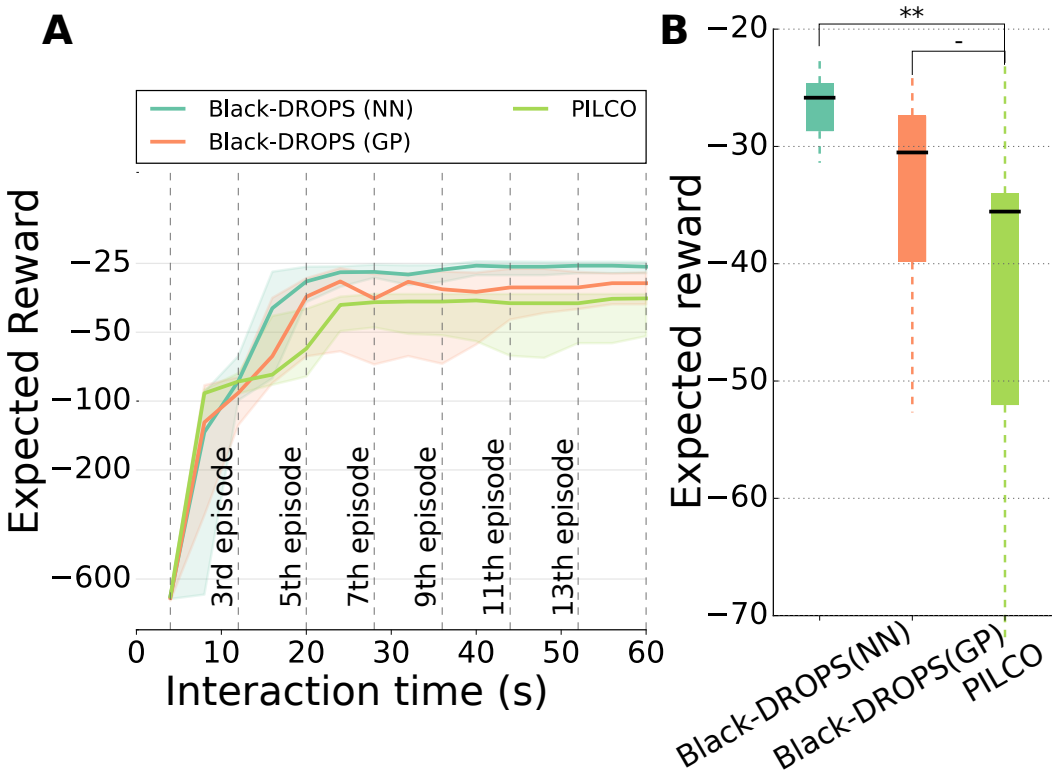


Figure 4.9: Results for the noisy cart-pole task with the quadratic reward (20 replicates): (**A**) Expected reward per episode; note that this metric is not available to the algorithm and only used for evaluation. The y-axis is in log-space. Black-DROPS and PILCO converge to high performing policies in similar interaction time compared with the smooth reward. Black-DROPS with the neural network policy outperforms PILCO both in the quality of the results and the variance. (**B**) Expected reward after 15 episodes. See Fig. 4.2 for legend.

needs around 6 episodes (Fig. 4.10**B**). Black-DROPS, also, shows less variance and converges to high quality controllers faster (6 episodes vs 8-9). A video of a typical run is available at `https://youtu.be/kTEyYiIFGPM`.

4.6 Improving performance and computation times with empirical bootstrap and races

Like we already said, the Black-DROPS algorithm will struggle when the system noise, $\boldsymbol{w}$, is not negligible or when the initial state distribution $p(\boldsymbol{x}_0)$ is wide. In the previous sections, we used a naïve approach to handle these cases; *i.e.*, to increase the number of sampled rollouts (or trajectories) so that the estimate of the expected return is not too noisy and CMA-ES can properly maximize its expectation. Nevertheless, this adds one more hyper-parameter to the user that might not be obvious how to set. Additionally, the Black-DROPS algorithm, as we have described it so far, still requires big computation times between each episode and the practical gains from parallelization are limited to 12-24 threads (that is, around 4-8x speed-ups compared to 1 thread implementation).

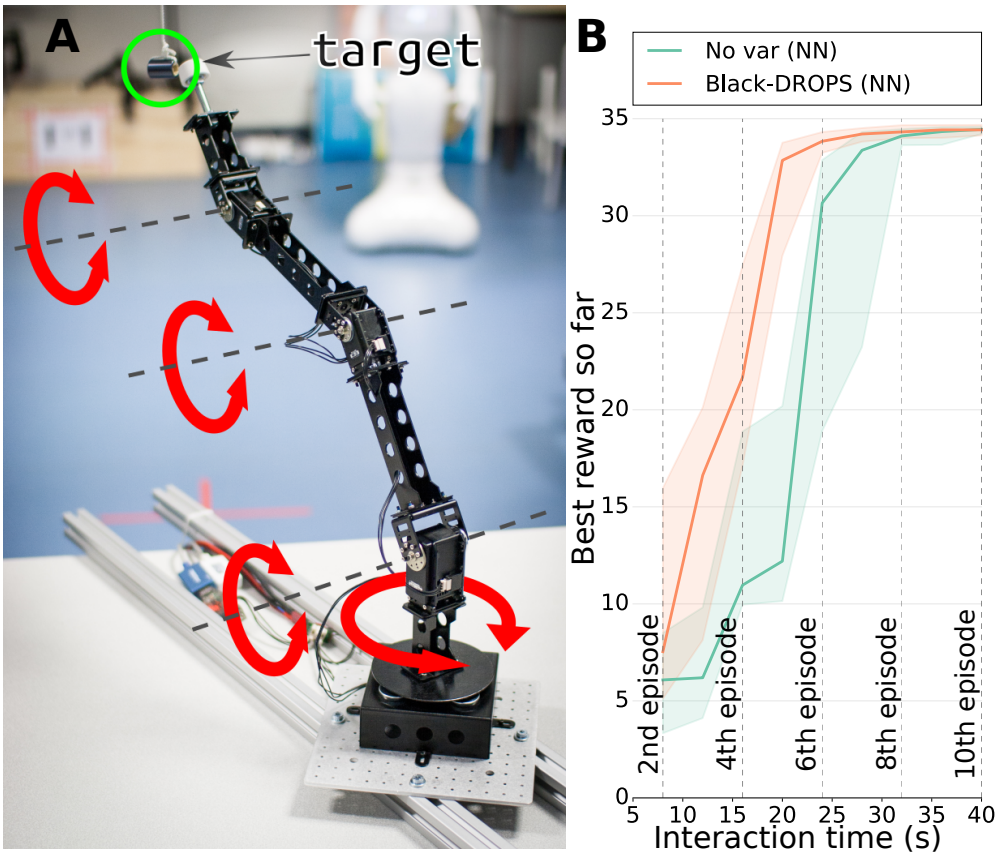


Figure 4.10: Manipulator task (10 replicates for each treatment).

In this section, we will describe how the empirical bootstrap method (Efron and Tibshirani, 1994) and the idea of racing (Heidrich-Meisner and Igel, 2009) can be used to both improve the quality of the solutions found by Black-DROPS (*e.g.*, in noisy systems as described before), but also improve the computation times. Our goal is to allocate as efficiently as possible the available budget of function evaluations[7], that is, to spend more function evaluations where it is actually needed and not blindly, like pure Monte-carlo methods.

At any given generation of CMA-ES, we have λ candidates and we are seeking the μ best of them. As we only have access to a noisy estimate of each candidate, we should re-evaluate them to get more accurate estimations. But how many evaluations of each individual is enough? Since the variance of the GPs is not the same in the entire state-action space (we have more data in some regions, less in others), each candidate needs different number of evaluations. This makes our task even more difficult, because we cannot make one decision for the whole population; this would "waste" some evaluations on individuals that do not need them or not use some evaluations where they are needed.

We take inspiration from Heidrich-Meisner and Igel (2009) and exploit the fact that only the ranking — and not the precise values of the expected return — affects the convergence properties of population, rank-based optimizers, like

[7]In our case, one function evaluation is one sampled trajectory or rollout.

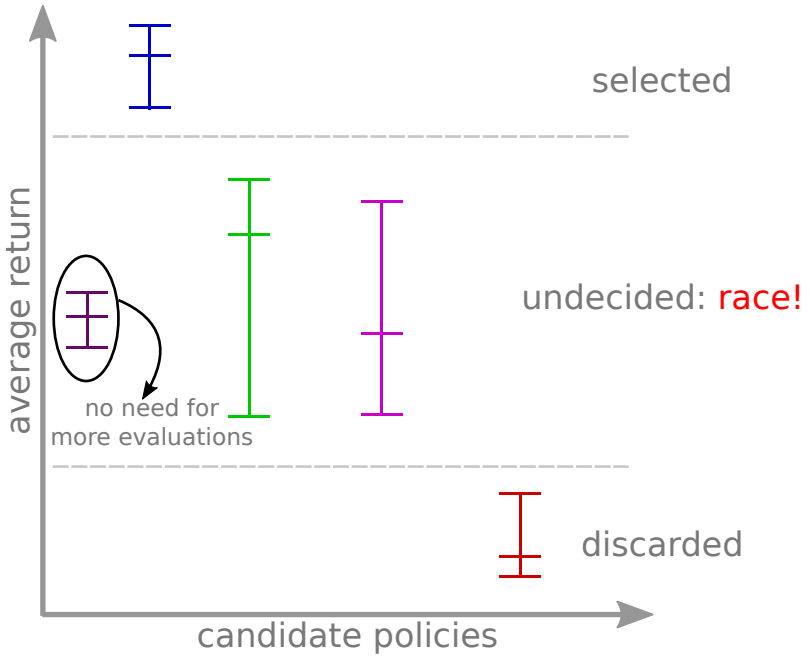


Figure 4.11: Illustration of racing with the empirical bootstrap method.

CMA-ES. Our main idea is to not use the same evaluation budget for all the candidate policies, but let them "race" (Fig. 4.11). In other words, we would like to stop evaluating a candidate policy when we are confident that (a) it is already in the best policies and this cannot change (with some probability), or (b) it is already in the worst policies and it cannot change its ranking (with some probability). Furthermore, we would also like to stop evaluating an individual when we are confident about our current estimate of the expected return. This criterion will save function evaluations by exploiting the fact that an individual might not need additional evaluations to have an accurate estimate of the expected return and its confidence intervals.

More concretely, we perform evaluation "waves", that is, iterations where some candidates might not survive and stop being reevaluated. We define the above two criteria for stopping the reevaluations of candidates, that is: (a) selection races that select or discard candidates based on their confidence intervals (*i.e.*, stop reevaluating an individual because it is already very good or very bad; see Fig. 4.11), and (b) variation of the candidates' estimation accuracy.

To evaluate these two criteria, we use the empirical bootstrap method in two different ways: (a) to compute an estimate of the accuracy of the mean prediction of each candidate policy, and (b) to compute confidence intervals (*i.e.*, bounds) of each policy. Although there exist a few other techniques for computing confidence intervals, like the Hoeffding inequality (Serfling, 1974), they usually require knowledge about the initial bounds and over-estimate the intervals (*i.e.*, their computed bounds are too wide) (Heidrich-Meisner and Igel, 2009). Moreover, the bootstrap method is very easy to implement and we can take advantage of multi-core architectures to speed it up.

In more detail, initially all policies (*i.e.*, candidates) are evaluated n_{step} times and then labelled active and undecided (see Algo. 18 in appendix). In every following "wave", for each active policy we compute N bootstrapped differences of the mean estimation. Using this bootstrapped values, we update the estimated mean performance and confidence intervals. If a policy's estimated mean is accurate enough, we remove it from the active set. If the lower bound of an active policy is better than the upper bounds of at least $\lambda - \mu$ other candidates, it is selected (and removed from the undecided and active sets). If the upper bound of an active policy is worse than the lower bounds of at least μ other candidates, it is discarded (and removed from the undecided and active sets). Finally, all the candidate policies that survived both processes and remain in the active set are reevaluated n_{step} times.

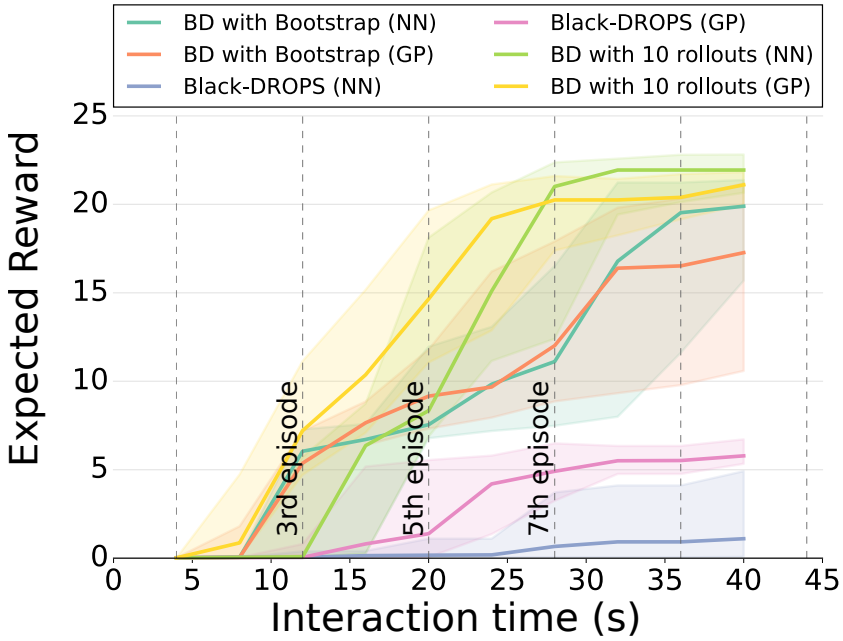


Figure 4.12: Expected reward per episode for the very noisy pendulum task (20 replicates). Black-DROPS with bootstrap and races is able to outperform vanilla Black-DROPS. Choosing the number of rollouts by using expert knowledge, however, provides the best results. See Fig. 4.2 for legend.

This approach has a few hyper-parameters: $n_{\text{step}}, n_{\text{max}}, N, \tau_{\text{boot}}, \tau_{\text{race}}$. Nevertheless, they are easy to set as they correspond to intuitive properties. In particular, $n_{\text{step}} \in \mathbb{Z}$ defines how many evaluations per wave we want to perform to the active candidates; in other words, it defines how many reevaluations make it more likely for our mean estimate to be more accurate. We define $n_{\text{max}} \in \mathbb{Z}$ as the maximum number of reevaluations allowed per CMA-ES iteration and $N \in \mathbb{Z}$ as the number of bootstrap samples. The more bootstrap samples the better as they will improve the accuracy of our computations. We, also, define $\tau_{\text{boot}} \in [0, 1]$ as the percentage of the mean value that the average error should be smaller to consider it accurate enough. Lastly, $\tau_{\text{race}} \in [0, 1]$ defines the percentage of the confidence interval we would like to compute.

4.6.1 Preliminary Results

To showcase the effectiveness of this extension, we experiment with a very noisy inverted pendulum task. This task is identical with the one used in the previous sections, but with bigger noise. In more detail:

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

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

- **Reward:** The reward is the same as per Eq. (4.9).
- In this setting, we assume $p(\boldsymbol{x}_0) \sim \mathcal{N}(0, 0.01)$ and a rather big Gaussian measurement noise is applied to the states, $\sim \mathcal{N}(0, \begin{bmatrix} 0.2 & 0 \\ 0 & 0.1 \end{bmatrix})$.

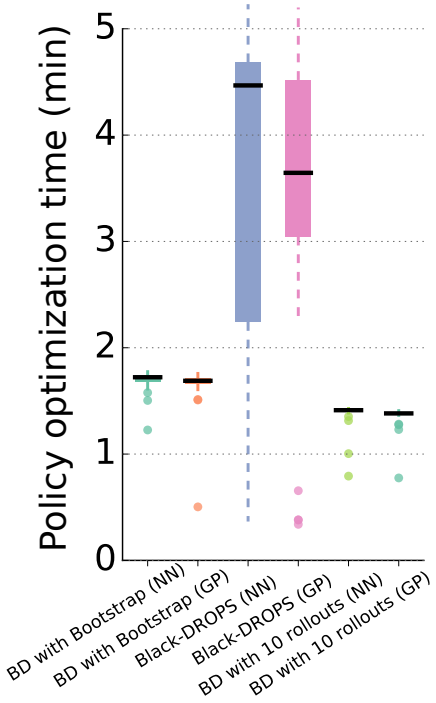


Figure 4.13: Policy optimization time for the very noisy pendulum task (20 replicates). Black-DROPS with bootstrap and races requires similar time to the expert number of rollouts and much less time than the vanilla Black-DROPS algorithm. See Fig. 4.2 for legend.

We compare three variants of Black-DROPS:

- Black-DROPS with bootstrap and races — the new extension[8];
- Black-DROPS with 1 sampled rollout — vanilla Black-DROPS;
- Black-DROPS with 10 sampled rollouts — setting the number of rollouts to a number specified by an expert.

The results show that our extension greatly improves the quality of solutions compared to the vanilla Black-DROPS (Fig. 4.12). Moreover, if we set the number of rollouts by using expert knowledge (*i.e.*, 10 in this case), we get the best results. It is very interesting to observe that by determining automatically the number of rollouts with our proposed approach, we get similar computation time compared to using expert knowledge and can be much faster than vanilla Black-DROPS (Fig. 4.13).

These results showcase that determining the proper number of rollouts for each candidate plays a catalytic role to the convergence behavior of CMA-ES, and thus affects both the quality and the computation time of the results

[8] We set $n_{\text{step}} = 5$, $n_{\max} = 50$, $N = 100$, $\tau_{\text{boot}} = 0.1$, and $\tau_{\text{race}} = 0.8$.

in Black-DROPS. The empirical bootstrap and racing idea is a first attempt to address the computational time limitations of Black-DROPS in order to make it a more practical algorithm. Nevertheless, further investigation is required so that we can have a procedure in Black-DROPS that allocates the available function evaluations as efficiently as possible, while not hurting the performance.

4.7 Handling sparse reward scenarios

So far[9], we have assumed an informative immediate reward function that gives reinforce signals throughout the whole behavior of the robot. This essentially means that Black-DROPS is greedy and mostly exploiting. However, rewards are much more sparse in many interesting learning scenarios: typically, we would like to reward the robot when it successfully achieves the task, and not for all the intermediate steps that we think should lead to success. For example, a robot might need to open a drawer and get rewarded by how much the drawer is open: in most of the state space, the reward is zero because the robot does not even touch the handle, meaning that Black-DROPS need to open the drawer purely by chance to start learning a policy. In most realistic scenarios, this is unlikely to happen.

In this section, we give a first potential solution to this issue by combining model-based policy search with novelty-based ideas (Lehman and Stanley, 2008, 2011). In Novelty Search (NS) (Lehman and Stanley, 2011), the task performance is substituted by a novelty measure (usually the distance of candidates in some *feature* or *behavior* space) and thus, NS promotes behavioral diversity and accumulates potential stepping stones for building more complex solutions. These novelty-based methods are tightly related to approaches that are based on intrinsic motivation (Forestier et al., 2017) or curiosity (Laversanne-Finot et al., 2018; Oudeyer, 2018) and try to create agents that perform activities for their inherent satisfaction rather than for some separable consequence (Ryan and Deci, 2000). Following both lines of work, we propose an algorithm that tries to handle the trade-off between novelty (exploration) and cumulative reward (exploitation) inside a learned dynamics model. We call this algorithm Multi-DEX, for Multi-objective Data-Efficient eXploration.

The main difference from the Black-DROPS algorithm is that we replace CMA-ES with NSGA-II, which is a multi-objective optimization algorithm (Deb et al., 2002) based on the Pareto-optimality concept (also called Pareto-efficiency) (Deb and Kalyanmoy, 2001):

Definition 1 A solution $\boldsymbol{\theta}_1$ is said to dominate another solution $\boldsymbol{\theta}_2$, if:

1. the solution $\boldsymbol{\theta}_1$ is not worse than $\boldsymbol{\theta}_2$ with respect to all objectives,

[9] Rituraj Kaushik, and not I, is the lead author of this work. We here give a summary of this work. Please refer to (Kaushik et al., 2018) for details.

2. the solution $\boldsymbol{\theta}_1$ is strictly better than $\boldsymbol{\theta}_2$ with respect to at least one objective.

In more detail, we frame exploration as an additional objective (apart from the cumulative reward) that promotes policies that will most likely produce novel state trajectories in the real system. To keep the system within the more certain regions of the dynamics model so that prediction error can be avoided as much as possible, we set an additional objective to reduce the prediction variance in the trajectory. The three objectives are typically antagonistic, as the variance is likely to be higher for novel trajectories, and novel trajectories are likely to have lower rewards than those with the highest rewards. In more detail, we frame the model-based policy search as a multi-objective optimization problem with three objectives:

- **Cumulative reward:** To compute the cumulative reward for a policy, we propagate through the learned GP model of the system for T time steps using the policy as per Eq. (4.2), but using only the mean prediction of the GPs. Then, we aggregate all the immediate rewards to get the cumulative reward. As this is evaluated in the model, optimizing this objective means exploiting the learned dynamics to find a policy that is likely to improve the reward on the real system.

- **Novelty:** We compute the novelty score of a policy by comparing its expected state trajectory with the expected state trajectories of already executed policies. To keep the computation time tractable, we sub-sample them into s_n equally spaced time-steps. We concatenate these state samples into one vector that we call *state trajectory vector* $\boldsymbol{\beta}$, which represents the "expected state trajectory" of a policy. These vectors are archived in a set $\mathbb{B}$ of fixed size b_n and we re-compute them every time the GP model is updated. When $\mathbb{B}$ is full, we drop the least novel policy. We define the novelty score for any policy $\pi_{\boldsymbol{\theta}}$ as the minimum Euclidean distance of $\boldsymbol{\beta}_{\boldsymbol{\theta}}$ to the state trajectory vectors in $\mathbb{B}$:

$$\hat{J}_n(\boldsymbol{\theta}) = min(||\boldsymbol{\beta}_{\boldsymbol{\theta}} - \boldsymbol{\beta}||^2)_{\forall \boldsymbol{\beta} \in \mathbb{B}} \tag{4.12}$$

  This objective is promoting exploration policies that are likely to lead to state trajectories that are different from those already observed.

- **Cumulative model-variance:** We define the cumulative model-variance objective for a policy $\pi_{\boldsymbol{\theta}}$ as the negative mean[10] of the step-by-step model prediction variances:

$$\hat{J}_{\sigma^2}(\boldsymbol{\theta}) = -\frac{1}{T}\sum_{t=1}^{T} ||\sigma_{\boldsymbol{x}_t, \boldsymbol{u}_t}||^2 \tag{4.13}$$

[10] We use the negative mean because we are maximizing and we want minimum variance.

where $\boldsymbol{x}_t$ and $\boldsymbol{u}_t$ are given by applying the policy $\pi_{\boldsymbol{\theta}}$ on the model as in the cumulative reward case. This objective tries to keep the system close to regions where the model uncertainty is low and thereby avoids prediction error on the real system.

4.7.1 Learning system dynamics with sparse transitions

When the system dynamics have some sparse transitions that strongly affect the cumulative reward, learning the model in a naïve way (*i.e.*, using all the data points) can lead to suboptimal models and the policy search will struggle to find good policies. For example, in a sequential goal reaching task the state can include a Boolean switch that indicates whether the arm passed through the first way-point or not; because only one data point will have transition from $false$ to $true$ in any episode if the arm passes though the switch, learning a model with the full data (that mostly have no points with the switch transition) will most probably ignore these transition data points because they are likely to be (rightfully) considered as outliers. In other words, rare transitions have little impact on the mean squared error or likelihood of the dynamical model, whereas they are critical to leverage the dynamical model to learn a policy.

The intuition here is to have a balanced blend of ordinary trajectories and trajectories with rare transitions (leading to high reward) for model learning. Learning a dynamics model this way will retain information not only about the rare and high rewarding transitions but also about the regions where no reward was observed. As a result, this model can efficiently be used for exploration as well as exploitation. To do this, we maintain two fixed sized experience buffers, $\mathbb{P}$ and $\mathbb{H}$, where we keep the trajectory data of the episodes for model learning. For each new trajectory executed on the robot, we insert a new observed trajectory into $\mathbb{P}$ in a FIFO fashion if no reward is observed; otherwise, we insert it into $\mathbb{H}$. Note that, we keep the maximum buffer size of $\mathbb{P}$ equal to the data size of $\mathbb{H}$ to have a uniform blend of "high rewarding" and "no/low rewarding" trajectories for model learning.

4.7.2 Results

We evaluate Multi-DEX on a deceptive pendulum reward scenario and a drawer opening task. We compare it to several model-based and model-free state-of-the-art approaches: (1) Black-DROPS (chapter 4), a model-based policy search algorithm; (2) TRPO (Schulman et al., 2015), a model-free policy gradient approach; (3) TRPO with the VIME exploration strategy (Houthooft et al., 2016); (4) CMA-ES (Hansen, 2006), a black-box optimizer effective for direct policy search, and (5) GEP-PG (Colas et al., 2018), a curiosity-driven model-free approach. In all the tasks, we give a budget of 2500 trials to all the model-free approaches and 250 trials to the model-based ones.

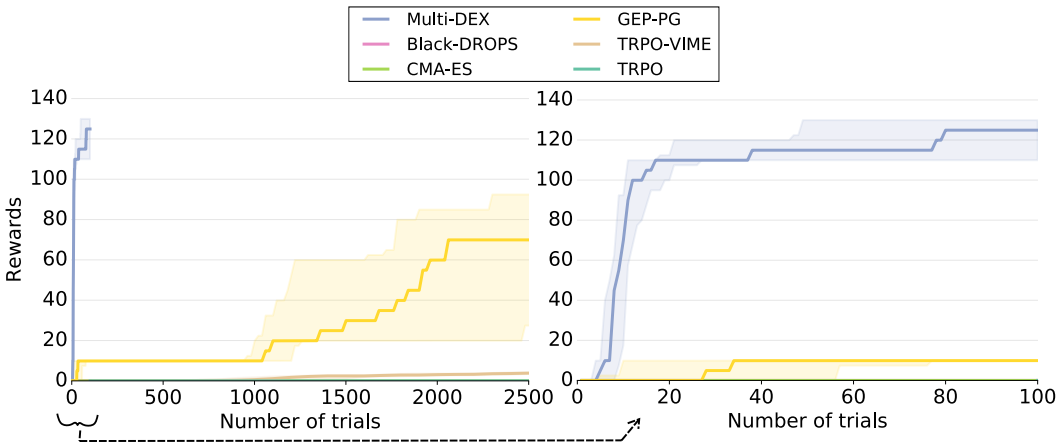
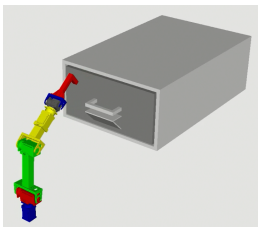


Figure 4.14: Results for the deceptive pendulum swing-up task. The plot shows the best reward found over the number of trials. The lines are median values and the shaded regions the 25th and 75th percentiles. The plot shows that Multi-DEX clearly outperforms all the competing approaches and solves the task in just 100 trials (approx 6.6 minutes of total interaction).

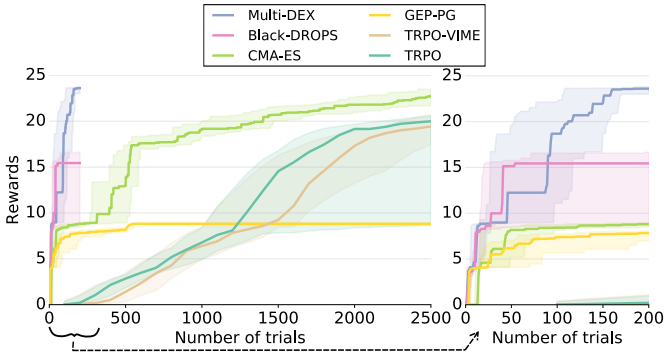
4.7.2.1 Deceptive Pendulum Swing-up Task

This simulated task consists of a pendulum powered by an underpowered torque-controlled actuator. The goal in this task is to swing the pendulum to the upright position applying torques as small as possible (*i.e.*, using minimum power) to the actuator and hold it in that position. The learning algorithm gets a constant positive reward of +10 every time-step the pendulum is in upright position (within some region). It also gets a negative reward proportional to the square of torque for every time step. Because of this two rewards, the total reward function possesses a deceptive landscape and algorithms without efficient exploration will converge to a reward of zero which is achieved when no torque is applied to the pendulum. This type of *"Gradient Cliff"* reward landscape was first proposed by (Lehman et al., 2017). We use a neural network policy with one hidden layer and 10 hidden units, the control frequency is $10Hz$ and every episode is 4 seconds long. In more detail:

- **State:** $\boldsymbol{x}_{pend\_sys} = [\theta, \dot{\theta}] \in \mathbb{R}^2$, where θ is the joint angle and $\dot{\theta}$ is the joint angular velocity. The initial state of the system is $x_0 = [0, 0]$.
- **Actions:** $\boldsymbol{u}_{pend\_sys} = [\tau] \in \mathbb{R}, -2.0 \leq \tau \leq 2.0$, where τ joint torque command for the arm.
- **Reward:** In this task, the reward is known to the algorithm as it is a function of the observable states and applied action to the system. The



(a)



(b)

Figure 4.15: Results for the drawer opening task: (a) Setup of the drawer opening task. The goal of the 2-DOF arm is to open the drawer and go back to the up-right position. (b) Best reward found per trial (20 replicates). The lines are median values and the shaded regions the 25th and 75th percentiles. Multi-DEX outperforms all the other approaches and finds working policies in about only 14 minutes of interaction (200 trials).

total reward is given by:

$$r(\boldsymbol{x}_{\text{pend\_sys}}, \tau) = r_1(\boldsymbol{x}_{\text{pend\_sys}}, \tau) + r_2(\boldsymbol{x}_{\text{pend\_sys}}, \tau) \tag{4.14}$$

$$r_1(\boldsymbol{x}_{\text{pend\_sys}}, \tau) = \begin{cases} 0, & \text{if } \pi - \theta < \pi/60 \\ +10, & \text{otherwise.} \end{cases} \tag{4.15}$$

$$r_2(\boldsymbol{x}_{\text{pend\_sys}}, \tau) = -0.001 * \tau^2 \tag{4.16}$$

The results show that Multi-DEX quickly reaches to a very high positive reward with very small variance within the budget of 100 trials (Fig. 4.14). On the contrary, GEP-PG and TRPO-VIME could not reach to the same quality of solutions even after 2500 trials on the system. As TRPO, Black-DROPS and CMA-ES do not have any directed exploration, they could not improve the reward and stay close to zero by minimizing the applied torque only.

4.7.2.2 Drawer opening task with a robotic arm

The goal of this simulated task is to open a drawer with a 2-DOF robotic arm and go back to the up-right position (Fig. 4.15a). Similarly to the previous task, the length of each episode is 4 seconds ($10Hz$ control) and all the algorithms use neural network policies (one hidden layer and 10 hidden units). In more detail:

- **State:** $\boldsymbol{x}_{\text{drawer}} = [\theta_0, \theta_1, \delta] \in \mathbb{R}^3$, where θ_i are joint angles and δ the drawer displacement.
- **Actions:** $\boldsymbol{u}_{\text{drawer}} = [v_0, v_1] \in \mathbb{R}^2, -1 \leq v_0, v_1 \leq 1$ rad/s are joint velocity commands.
- **Reward:** In this task, the reward depends only on the current state and is known to the algorithm as it is a function of the observable state. The

total reward is given by:

$$r(\boldsymbol{x}_{\text{drawer}}) = r_{\text{return}}(\boldsymbol{x}_{\text{drawer}}) + \delta \tag{4.17}$$

$$r_{\text{return}}(\boldsymbol{x}_{\text{drawer}}) = \begin{cases} 0, & \text{if } \delta \leq 0.2 \\ \exp(-\theta_0^2 - \theta_1^2), & \text{otherwise.} \end{cases} \tag{4.18}$$

In this task, the reward space is moderately sparse and additionally has a misleading reward space because of the composition of two different rewards. Any algorithm without efficient exploration will converge to the reward associated with the drawer displacement only. Multi-DEX is able to find policies that complete the task in just 200 trials (around 14 minutes of interaction), whereas all the other approaches are deceived by the reward space (Fig. 4.15b — 20 replicates). Black-DROPS and TRPO, without any directed exploration, fall in the sub-optimal solution (*i.e.*, just opening the drawer quickly). CMA-ES, TRPO-VIME and GEP-PG either fail to converge or need more than 2500 trials to reach the same quality of solutions as Multi-DEX.

4.8 Conclusion and discussion

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

However, even with 24 threads, Black-DROPS still requires around 80 minutes for completing 15 episodes in the noisy cart-pole task. The main issue is the quadratic computational complexity of the prediction of the GPs (we are doing around 64,000,000 GP queries per episode). Possible solutions include using local GPs (Park and Apley, 2017; Deisenroth and Ng, 2015) or to stop using GPs and make use of recent advances in neural networks with uncertain predictions (Gal et al., 2016; Gal and Ghahramani, 2016; Chua et al., 2018; Higuera et al., 2018). The second issue is the saturating parallelization abilities of CMA-ES: CMA-ES usually uses small population sizes and thus the gain of parallelization saturates to around 16-24 threads (at least in all the cases that we considered). A possible solution to this problem is to replace CMA-ES with another black-box population-based optimizer, like for example NSGA-II (Deb et al., 2002) or MAP-Elites (Mouret and Clune, 2015), that can scale better with the number of cores. This would lead to better speed-ups as the number of cores increases.

Using the variance in the optimization is one of the key components to learn with as little interaction time as possible. However, the learned dynamics models are only confident in areas of the state space previously visited and thus could drive the optimization into local optima when multiple and

diverse solutions exist. This exploration-exploitation dilemma is central in reinforcement learning (Sutton and Barto, 1998). In section 4.7, we introduced Multi-DEX, an algorithm that effectively explores in a learned dynamics model by solving a multi-objective optimization problem where both the cumulative reward and the novelty of the state trajectories are being taken into account. Nevertheless, we only tested Multi-DEX on fairly low-dimensional robots and it will struggle to work in noisy settings as is (because NSGA-II is elitist). Further investigation is needed to see how we can scale up to higher dimensional systems, and replacing NSGA-II by multi-objective optimizers that perform better in noisy settings (Eskandari and Geiger, 2009) might allows us to use Multi-DEX with noisy systems.

Chapter 5

Combining Model Identification and Gaussian Processes for Fast Learning

The results and text of this chapter have been partially published in the following articles.

Articles:

- **Chatzilygeroudis, K.**, and Mouret, J.-B., 2018. *Using Parameterized Black-Box Priors to Scale Up Model-Based Policy Search for Robotics.* **International Conference on Robotics and Automation** (Chatzilygeroudis and Mouret, 2018).

Other contributors:

- Jean-Baptiste Mouret (Thesis supervisor)

Author contributions:

- **KC** and JBM organized the study. **KC** wrote the code. **KC** performed the experiments. **KC** and JBM analyzed the results and wrote the paper.

5.1 Introduction

In the previous chapter, we introduced Black-DROPS, a model-based policy search algorithm for robotics that is purely black-box and can take advantage of parallel computations. We saw that Black-DROPS achieves similar data-efficiency to state-of-the-art approaches like PILCO (Deisenroth et al., 2015) (*e.g.*, less than 20s of interaction time to solve the cart-pole swing-up task), while being faster on multi-core computers, easier to set up, and much less limiting (*i.e.*, it can use any policy and/or reward parameterization; it can even learn the reward model).

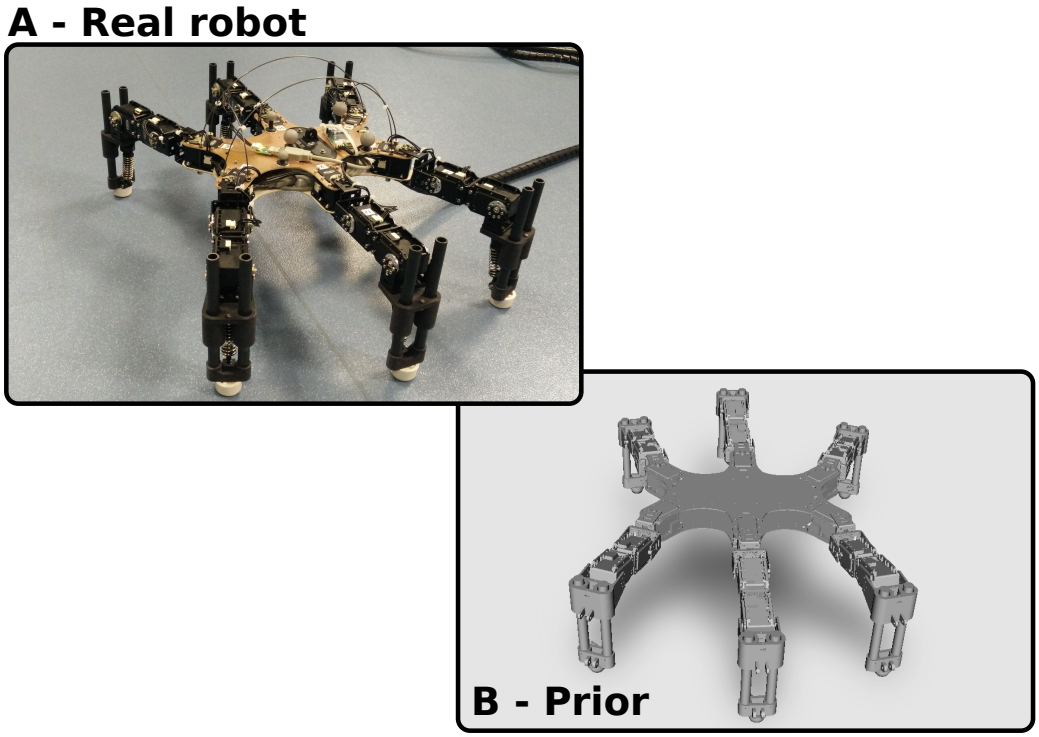


Figure 5.1: A. The physical hexapod robot used in the experiments (48D state space and 18D action space). **B.** The simulated hexapod that is used as a prior model for our approach in the experiments.

However, while Black-DROPS scales well with the number of cores, the main challenge of model-based policy search is scaling up to complex robots: since the algorithms model the transition dynamics, they require more data to acquire an accurate model as the dimensionality of state/action space increases. In the general case, the quantity of data to learn a good approximation of a surrogate model scales exponentially with the dimensionality of the learned function (this is the curse of dimensionality, see (Bellman, 1957)). As a consequence, the data-efficiency of model-based approaches greatly suffers from the increase of the dimensionality of the model. In practice, model-based policy search algorithms can currently be employed only with simple systems up to 10-15D state and action space combined (*e.g.*, double cart-pole or a simple manipulator).

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

- scale to high dimensional and complex robots (*e.g.*, walking or soft robots);
- take advantage of multi-core architectures to speed-up computation times;
- perform the search in the full policy space (*i.e.*, the more real trials, the better expected reward);

- make as few assumptions as possible about the type of robot and the prior information (*i.e.*, require no specific structure or differentiable models);
- be able to select among several prior models or to tune the prior model.

A few algorithms leverage prior information to speed-up learning on the real system (Cutler and How, 2015; Lee et al., 2017; Saveriano et al., 2017; Bischoff et al., 2014; Cully et al., 2015; Marco et al., 2017), but none of them fulfills all of the above properties. In this chapter, we propose a novel, purely black-box, flexible and data-efficient model-based policy search algorithm that combines ideas from the Black-DROPS algorithm, from simulation-based priors, and from recent model learning algorithms (Nguyen-Tuong and Peters, 2010; Camoriano et al., 2016) in order to get closer to the ideal algorithm. We will show that our approach is capable of learning policies in about 30 seconds to control a damaged physical hexapod robot (48D state space, 18D action space). We will, also, extensively evaluate our approach in a simulated pendubot experiment and show that it outperforms state-of-the-art model-based policy search algorithms without (PILCO (Deisenroth et al., 2015), Black-DROPS) and with priors (PILCO with priors (Cutler and How, 2015)), as well as prior-based Bayesian optimization (IT&E (Cully et al., 2015)).

5.2 Problem Formulation

Here, we adapt the generic problem formulation of Section 2.2 to our specific case. We consider dynamical systems of the form:

$$\boldsymbol{x}_{t+1} = \boldsymbol{x}_t + F(\boldsymbol{x}_t, \boldsymbol{u}_t) + \boldsymbol{w} \tag{5.1}$$

with continuous-valued states $\boldsymbol{x} \in \mathbb{R}^E$ and controls $\boldsymbol{u} \in \mathbb{R}^U$, i.i.d. Gaussian system noise $\boldsymbol{w}$, and unknown transition dynamics F. We assume that we have an initial guess of the dynamics, the function $M(\boldsymbol{x}_t, \boldsymbol{u}_t)$, that may not be accurate either because we do not have a precise model of our system (*i.e.*, what is called the *"reality-gap"* (Mouret and Chatzilygeroudis, 2017)) or because the robot is damaged in an unforeseen way (*e.g.*, a blocked joint or faulty motor/encoder) (Cully et al., 2015; Chatzilygeroudis et al., 2018a).

Contrary to previous works (Lee et al., 2017; Nguyen-Tuong and Peters, 2010; Camoriano et al., 2016), we assume no structure or specific properties of our initial dynamics model M (*i.e.*, we treat it as a black-box function), other than it has some tunable parameters, $\boldsymbol{\phi}_M$, which change its behavior. Examples of these parameters can be some optimization parameters (*e.g.*, type of optimizer) of a dynamic simulator involving contacts and collisions or some physical parameters of the robot (*e.g.*, masses of the bodies). Finally, we add a non-parametric model, f (with associated hyper-parameters $\boldsymbol{\phi}_K$), to model whatever is not possible to capture with M:

$$\boldsymbol{x}_{t+1} = \boldsymbol{x}_t + M(\boldsymbol{x}_t, \boldsymbol{u}_t, \boldsymbol{\phi}_M) + f(\boldsymbol{x}_t, \boldsymbol{u}_t, \boldsymbol{\phi}_K) + \boldsymbol{w} \tag{5.2}$$

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

$$J(\boldsymbol{\theta}) = \mathbb{E}\left[\sum_{t=1}^{T} r(\boldsymbol{x}_t)\Big|\boldsymbol{\theta}\right] \tag{5.3}$$

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

In model-based policy search with priors, we follow the generic policy search algorithm (as defined in Algo. 1) and implement CollectStrategy, LearnStrategy and UpdateStrategy as in Black-DROPS (see Algo. 12). InitStrategy is implemented differently: the initial parameters $\boldsymbol{\theta}_1$ are found by optimizing the prior model; this implies that *there is no prior information on the policy parameters*.

In more detail, we begin by optimizing the policy on the prior model and applying it on the real system to gather the initial data. Afterwards, a loop is iterated where we first learn a model using the prior model and the collected data and then optimize the policy given this newly learned model. Finally, the policy is applied on the real system, more data is collected and the loop re-iterates until the task is solved.

5.3 Approach

5.3.1 Gaussian processes with the simulator as the mean function

We would like to have a model $\hat{F}$ that approximates as accurately as possible the unknown dynamics F of our system given some initial guess, M. As in the previous chapter, we rely on Gaussian processes (GPs) to do so as they have been successfully used in many model-based reinforcement learning approaches (Deisenroth et al., 2015; Chatzilygeroudis et al., 2017; Engel et al., 2005; Nguyen-Tuong and Peters, 2011; Deisenroth et al., 2013; Chatzilygeroudis et al., 2018a; Polydoros and Nalpantidis, 2017).

Similarly to Black-DROPS and PILCO, as inputs we use tuples made of the state vector $\boldsymbol{x}_t$ and the action vector $\boldsymbol{u}_t$, that is, $\tilde{\boldsymbol{x}}_t = (\boldsymbol{x}_t, \boldsymbol{u}_t) \in \mathbb{R}^{E+U}$; as training targets, we use the difference between the current state vector and the next one: $\boldsymbol{\Delta}_{\boldsymbol{x}_t} = \boldsymbol{x}_{t+1} - \boldsymbol{x}_t \in \mathbb{R}^E$. We use E independent GPs to model each dimension of the difference vector $\boldsymbol{\Delta}_{\boldsymbol{x}_t}$. Assuming $D_{1:n} = \{F(\tilde{\boldsymbol{x}}_1), ..., F(\tilde{\boldsymbol{x}}_n)\}$ is a set of observations and $M(\tilde{\boldsymbol{x}})$ being the simulator function (*i.e.*, our initial guess of the dynamics — tunable or not; we drop the $\boldsymbol{\phi}_M$ parameters here for brevity), we can query the GP at a new input point as per Eq. (2.48)-(2.49).

This model learning procedure has been used in several articles (Ko et al., 2007; Nguyen-Tuong and Peters, 2010, 2011) (see also Sec. 2.7.2) and in particular to learn the cumulative reward model for a BO procedure highlighted in the IT&E approach (Cully et al., 2015). GP-ILQG (Lee et al., 2017) and

PI-REM (Saveriano et al., 2017) formulate a similar model learning procedure for optimal control (under model uncertainty) and policy search respectively. GP-ILQG additionally assumes that the prior model M is differentiable, which is not always true and might be too slow to perform via finite differences (*e.g.*, when using black-box simulators for M). PILCO with priors (Cutler and How, 2015) utilizes a similar scheme but assumes that the prior model M is a GP learned from simulation data that is gathered from running PILCO on the prior system.

We use the exponential kernel with automatic relevance determination (Rasmussen and Williams, 2006) as defined in Eq. (2.23) ($\boldsymbol{\phi}_K$ are the kernel hyper-parameters). When searching for the best kernel hyper-parameters through Maximum Likelihood Estimation (MLE) for a GP with a non-tunable mean function M, we seek to maximize (Rasmussen and Williams, 2006):

$$p(D_{1:n}|\tilde{\boldsymbol{x}}_{1:n}, \boldsymbol{\phi}_K) = \frac{1}{\sqrt{(2\pi)^t |K|}} e^{-\frac{1}{2}(D_{1:n} - M(\tilde{\boldsymbol{x}}_{1:n}))^T K^{-1} (D_{1:n} - M(\tilde{\boldsymbol{x}}_{1:n}))} \tag{5.4}$$

The gradients of this likelihood function can be analytically computed, which makes it possible to use any gradient based optimizer. Since we have E independent GPs, we have E independent optimizations. We use the limbo C++11 library for GP regression (Cully et al., 2018).

5.3.2 Mean functions with tunable parameters

We would like to use a mean function $M(\tilde{\boldsymbol{x}}, \boldsymbol{\phi}_M)$, where each vector $\boldsymbol{\phi}_M \in \mathbb{R}^{n_M}$ corresponds to a different prior model of our system (*e.g.*, different lengths of links). Searching for the $\boldsymbol{\phi}_M$ that best matches the observations can be seen as a model identification procedure, which could be solved via minimizing the mean squared error; nevertheless, the GP framework allows us to jointly optimize for the kernel hyper-parameters and the mean parameters, which allows the modeling procedure to balance between non-parametric and parametric modeling. We can easily extend Eq. (5.4) to include parameterized mean functions:

$$p(D_{1:n}|\tilde{\boldsymbol{x}}_{1:n}, \boldsymbol{\phi}_K, \boldsymbol{\phi}_M) = \frac{1}{\sqrt{(2\pi)^t |K|}} e^{-\frac{1}{2}(D_{1:n} - M(\tilde{\boldsymbol{x}}_{1:n}, \boldsymbol{\phi}_M))^T K^{-1} (D_{1:n} - M(\tilde{\boldsymbol{x}}_{1:n}, \boldsymbol{\phi}_M))} \tag{5.5}$$

This time, even though we assumed that we have E independent GPs (one for each output dimension), all of them need to share the same mean parameters $\boldsymbol{\phi}_M$ (contrary to the kernel parameters, which are typically different for each dimension), because the model of the robot should be consistent in all of the output dimensions. Thus, we have to jointly optimize for the mean parameters and the kernel hyper-parameters of all the GPs. Since most dynamic simulators are not differentiable (or too slow to differentiate by finite differences), we

Algorithm 13 GP-MI Learning process

procedure GP-MI($D_{1:t}$)
  Optimize $\boldsymbol{\phi}_M^*$ according to EvaluateModel($\boldsymbol{\phi}_M$, $D_{1:t}$) using a gradient-free local optimizer
  return $\boldsymbol{\phi}_M^*$
end procedure
procedure EvaluateModel($\boldsymbol{\phi}_M$, $D_{1:t}$)
  Initialize E GPs $f_1, \ldots, f_E$ as $f_i(\tilde{\boldsymbol{x}}) \sim \mathcal{N}(M_i(\tilde{\boldsymbol{x}}, \boldsymbol{\phi}_M), k_i(\tilde{\boldsymbol{x}}, \tilde{\boldsymbol{x}}))$ ▷ M^i queries M and returns the i-th element of the return vector, k_i is the kernel function of the i-th GP
  for i from 1 to E **do** ▷ This can also be done in parallel
    Optimize the kernel hyper-parameters, $\boldsymbol{\phi}_K^i$, of f_i given $D_{1:t}^i$ assuming $\boldsymbol{\phi}_M$ fixed ▷ $D_{1:t}^i$ is the i-th column of $D_{1:t}$
    $\text{lik}_i = p(D_{1:t}^i | \tilde{\boldsymbol{x}}_{1:t}, \boldsymbol{\phi}_K^i, \boldsymbol{\phi}_M)$ ▷ Eq. (5.5)
  end for
  return $\sum_{i=1}^{E} \text{lik}_i$ ▷ Sum of the independent likelihoods
end procedure

cannot resort to gradient-based optimization to optimize Eq. (5.5) jointly for all the GPs. A black-box optimizer like CMA-ES (Hansen and Ostermeier, 2001) could be employed instead, but this optimization was too slow to converge in our preliminary experiments.

To combine the benefits of both gradient-based and gradient-free optimization, we use gradient-based optimization for the kernel hyper-parameters (since we know the analytical gradients) and black-box optimization for the mean parameters. Conceptually, we would like to optimize for the mean parameters, $\boldsymbol{\phi}_M$, given the optimal kernel hyper-parameters for each of them. Since we do not know them before-hand, we use two nested optimization loops: (a) an outer loop where a gradient-free local optimizer searches for the best $\boldsymbol{\phi}_M$ parameters (we use a variant of the Subplex algorithm (Rowan, 1990) provided by NLOpt (Johnson Steven) for continuous spaces and exhaustive search for discrete ones), and (b) an inner optimization loop where given a mean parameter vector $\boldsymbol{\phi}_M$, a gradient-based optimizer searches for the best kernel hyper-parameters (each GP is independently optimized since $\boldsymbol{\phi}_M$ is fixed in the inner loop) and returns a score that corresponds to $\boldsymbol{\phi}_M$ for the optimal $\boldsymbol{\phi}_K$ (Algo. 13).

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

to be numerically more stable than the harmonic mean.

Our model learning approach, which we call *GP-MI* (Gaussian Process Model Identification), that combines non-parametric model learning and parametric model identification is related to the approach in (Nguyen-Tuong and Peters, 2010), but there are some key differences between them. Firstly, the model learning procedure in (Nguyen-Tuong and Peters, 2010) depends on the manipulator equation and cannot easily be used with robots that do not directly comply to the equation (one example would be the hexapod robot in our experiments or a soft robot with complex dynamics), whereas GP-MI imposes no structure on the prior model, other than providing some tunable parameters (continuous or discrete). Furthermore, the approach in (Nguyen-Tuong and Peters, 2010) is tied to inverse dynamics models and cannot be used with forward models in the general case (necessary for long-term forward predictions); on the contrary, GP-MI can be used with inverse or forward dynamics models and in general with any black-box tunable prior model.

5.3.3 Policy Search with the Black-DROPS algorithm

We use the Black-DROPS (see the previous chapter) algorithm for policy search because it allows us to use the type of priors discussed in Section 5.3.2 and to leverage specific policy parameterizations that are suitable for different cases (*e.g.*, we use a neural network policy for the pendubot task and an open-loop periodic policy for the hexapod). We assume *no prior information on the policy parameters* and we begin by optimizing the policy on the prior model. Moreover, we took advantage of multi-core architectures to speed-up our experiments. Contrary to Black-DROPS, PILCO (Deisenroth et al., 2015) cannot take advantage of multiple cores[1] and the need for deriving all the gradients for a different policy/reward makes it difficult (or even impossible) to try new ideas/policies.

PI-REM (Saveriano et al., 2017) is close to our approach as it leverages priors to learn the residual model and then performs policy search on the model. However, PI-REM assumes that the prior information is fixed and cannot be tuned, whereas our approach has the additional flexibility of being able to change the behavior of the prior. In addition, PI-REM utilizes the policy search procedure of PILCO that can be limiting in many cases as already discussed. Nevertheless, as Black-DROPS and PILCO have been shown to perform similarly when PILCO's limitations are not present (Chatzilygeroudis et al., 2017), we include in our experiments a variant of our approach that resembles PI-REM (Black-DROPS with fixed priors).

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

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

Table 5.1: Actual system and priors for the pendubot task.

5.4 Experimental Results

5.4.1 Pendubot swing-up task

We first evaluate our approach in simulation with the pendubot swing-up task. The pendubot is a two-link under-actuated robotic arm (with lenghts l_1, l_2 and masses m_1, m_2) and was introduced by (Spong and Block, 1995) (Fig. 5.2). The inner joint (attached to the ground) exerts a torque $|u| \leq 3.5$, but the outer joint cannot (both of the joints are subject to some friction with coefficients b_1, b_2). The system has four continuous state variables: two joint angles and two joint angular velocities. The angles of the joints, θ_1 and θ_2, are measured anti-clockwise from the upright position. The pendubot starts hanging down and the goal is to find a policy such that the pendubot swings up and then balances in the upright position. Each episode lasts 2.5s and the control rate is 20Hz. We use a distance based reward function as in (Chatzilygeroudis et al., 2017).

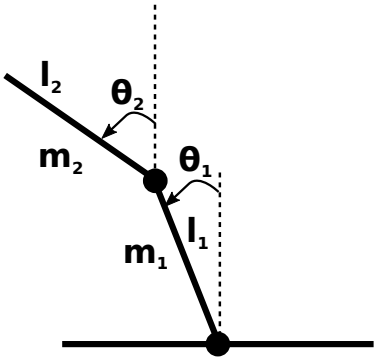


Figure 5.2: The pendubot system

We chose this task because it is a fairly difficult problem and forces slower convergence on model-based techniques without priors, but not too hard (*i.e.*, it can be solved without priors in reasonable interaction time); a fact that allowed us to make a rather extensive evaluation with meaningful comparisons. More precisely, we investigate 4 different prior models, we compare 7 different algorithms, and report 30 replicates of each combination.

We assume that we have 4 priors available; we tried to capture easy and difficult cases and cases where all the altered parameters can be tuned or not (see Table 5.1):

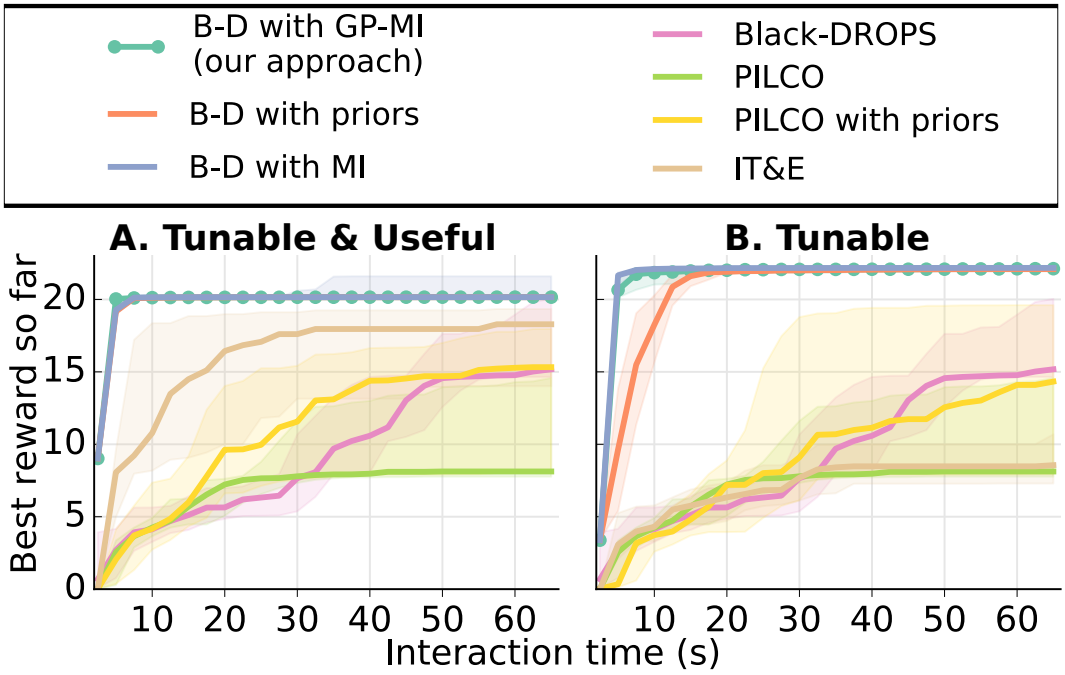


Figure 5.3: Results for the pendubot task (30 replicates of each scenario) — Tunable & Useful and Tunable priors. The lines are median values and the shaded regions the 25^{th} and 75^{th} percentiles. See Table 5.1 for the description of the priors. Black-DROPS with GP-MI always solves the task and achieves high rewards at least as fast as all the other approaches in all the cases that we considered. Black-DROPS with MI achieves good rewards whenever the parameters it can tune are the ones that are wrong (**A,B**) and bad rewards otherwise (Fig. 5.4**D**). Black-DROPS with priors performs very well whenever the prior model is not too far away from the real one (**A,B**) and not so well whenever the prior is misleading (Fig. 5.4**C**). Black-DROPS with priors and MI have very similar performance in **A** and as such are not easily distinguishable. IT&E and PILCO with priors are not able to reliably solve the task across different prior models.

- **Tunable & Useful:** a fully tunable prior that is very close to the actual one;
- **Tunable:** a fully tunable prior that is not very close to the actual;
- **Tunable & Misleading:** a prior that can be fully tuned, but is very far from the actual;
- **Partially tunable:** a prior that cannot be fully tuned, and not very close to the actual.

We compare 7 algorithms:

1. Black-DROPS (see Chapter 4);
2. Black-DROPS with fixed priors, which is close to PI-REM (Saveriano et al., 2017) and GP-ILQG (Lee et al., 2017)[2];

[2]The algorithm in this specific form is first formulated in this paper (*i.e.*, the Black-DROPS policy search procedure with a prior model), but, as discussed above, it is close in spirit with GP-ILQG (Lee et al., 2017) and PI-REM (Saveriano et al., 2017). Therefore, we

3. Black-DROPS with GP-MI (*our approach*);
4. Black-DROPS with MI (Black-DROPS where model learning is replaced by model identification — via mean squared error);
5. PILCO (Deisenroth et al., 2015);
6. PILCO with priors (Cutler and How, 2015);
7. IT&E (Cully et al., 2015).

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

Similarly to IT&E, since PILCO with priors is a deterministic algorithm given the same prior, for each prior we ran PILCO 30 times with different random seeds on the prior model (for 40 episodes in order for PILCO to converge to a good policy and model) and then ran PILCO with priors on the actual system once for each different model. We used priors both in the policy and the dynamics model when learning in the actual system (as advised in (Cutler and How, 2015)). We also used a GP policy with 200 pseudo-observations (Deisenroth et al., 2015)[3].

Black-DROPS with GP-MI always solves the task and achieves high rewards at least as fast as all the other approaches in the cases that we considered (Fig. 5.3,5.4). Black-DROPS with MI performs very well when the parameters it can tune are the ones that are wrong (Fig. 5.3**A,B** and Fig. 5.4**C**), and badly otherwise (Fig. 5.4**D** — *i.e.*, no parameters of the prior model can explain the data). Black-DROPS with fixed priors performs very well whenever the prior model is not far away from the real one (Fig. 5.3**A,B**) and not so well whenever the prior is misleading (Fig. 5.4**C**). Both Black-DROPS and PILCO cannot solve the task in less than $65s$ of interaction time, but Black-DROPS shows a faster learning curve (Fig. 5.3).

Interestingly, PILCO with priors is not able to always achieve better results than Black-DROPS and is always worse than Black-DROPS with fixed priors. This can be explained by the fact that PILCO without priors learns slower than Black-DROPS and is a more local search algorithm and as such needs more interaction time to achieve good results. On the contrary, Black-DROPS

assume that the performance of Black-DROPS with priors is representative of what could be achieved with PI-REM and GP-ILQG, although Black-DROPS with priors should be more flexible to use (Chapter 4).

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

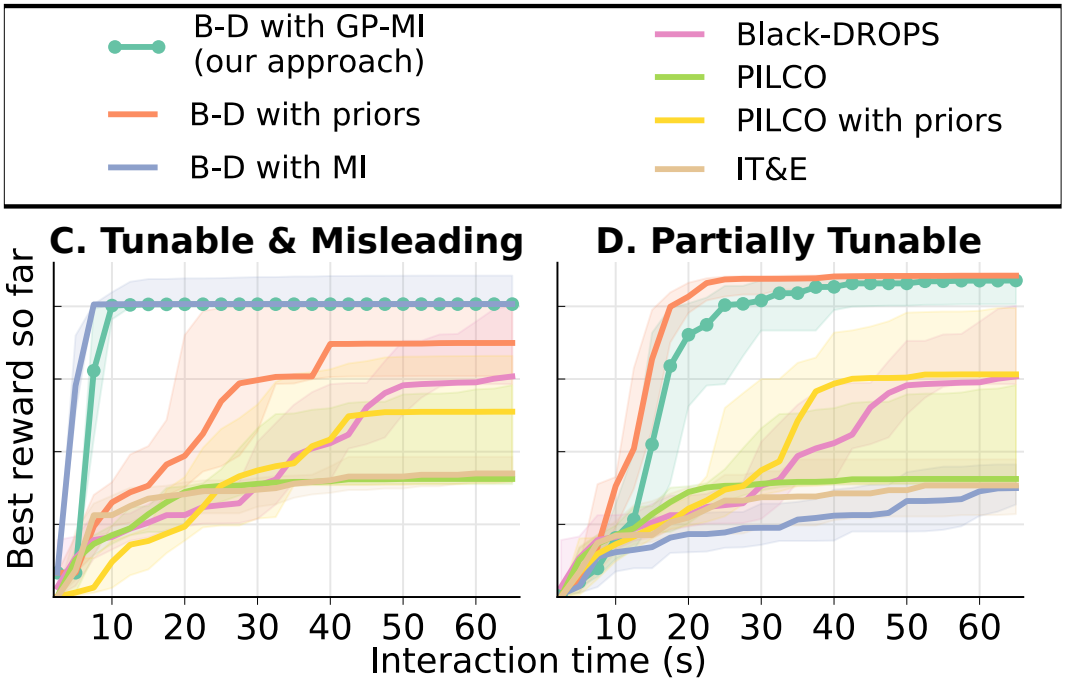


Figure 5.4: Results for the pendubot task (30 replicates of each scenario) — Tunable & Misleading and Partially tunable priors. See Fig. 5.3 for legend and description.

uses a modified version of CMA-ES that can more easily escape local optima. Moreover, the initial prior model for PILCO with priors is an approximated model, whereas Black-DROPS with priors uses the actual prior model to begin with. Lastly, the GP policy, that PILCO is mainly used with[4], creates really high dimensional policy spaces compared to the simple neural network policy that Black-DROPS is using (*i.e.*, 1400 vs 81 parameters) and as such causes the policy search to converge slower.

IT&E is not able to reliably solve the task and achieve high rewards. This is because IT&E assumes that (a) the system is redundant enough so that the task can be solved in many different ways and (b) there is a policy/controller in the pre-computed archive that can solve the task (*i.e.*, IT&E cannot search outside of this archive) (Cully et al., 2015). Obviously, these assumptions are violated in the pendubot scenario: (a) the system is underactuated and thus does not have the required redundancy, and (b) the system is inherently unstable and as such precise policy parameters are needed (it is highly unlikely that one of them exists in the pre-computed archive).

5.4.2 Physical hexapod locomotion

We also evaluate our approach on the hexapod locomotion task as introduced in the IT&E paper (Cully et al., 2015) with a physical robot (Fig. 5.1**A** and Fig. 5.5). This scenario is where IT&E excels and achieves remarkable recovery capabilities (Cully et al., 2015). We assume that a simulator of the intact robot

[4]So far, PILCO can only be used with linear or GP policy types (Deisenroth et al., 2015).

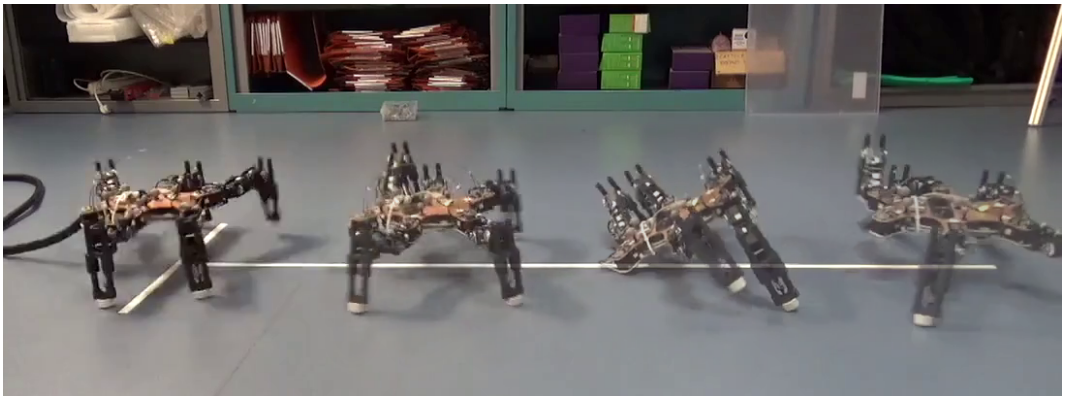


Figure 5.5: Walking gait on a damaged hexapod with Black-DROPS with GP-MI after less than 30 seconds of interaction time. The hexapod is able to walk in a straight line again thanks to our approach.

is available (Fig. 5.1**B**)[5]; for GP-MI we also assume that we can alter this simulator by removing 1 leg of the hexapod (*i.e.*, there are 7 discrete different parameterizations). This simulator is not accurate as we assume perfect velocity actuators and infinite torque. Each leg has 3 DOF leading to a total of 18 DOF.

The state of the robot consists of 18 joint angles, 18 joint velocities, a 6D Center Of Mass (COM) pose (position and orientation) and 6D COM velocities. The policy is an open-loop controller with 36 parameters that outputs 18D joint angles every 0.1s and is similar to the one used in Chapter 3 and in (Cully et al., 2015). Each episode lasts 4s and the robot is tracked with a motion capture system.

The task is to find a policy to walk forward as fast as possible (Fig. 5.5). Due to the complexity of the problem[6], we only compare 2 algorithms (IT&E and our approach) on 2 different conditions: (a) crossing the reality-gap problem; in this case our approach cannot mostly rely on the identification part and the importance of the GP modeling will be highlighted, and (b) one rear leg is removed; the back leg removals are especially difficult as most effective gaits of the intact robot rely on them.

The results show that Black-DROPS with GP-MI is able to learn highly effective walking policies on the physical hexapod robot (Fig. 5.6). In particular, using the dynamics simulator as prior information Black-DROPS with GP-MI is able to achieve better (and with less variance) walking speeds than IT&E (Cully et al., 2015) on the intact physical hexapod (Fig. 5.6**A**). Moreover, in the rear-leg removal damage case Black-DROPS with GP-MI allows the damaged robot to walk effectively after only 16 to 30 seconds of interaction time and finds higher-performing policies than IT&E ($0.21m/s$ vs $0.15m/s$ in the 8^{th} episode, Fig. 5.6**B**).

[5]We use the DART simulator (Lee et al., 2018).

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

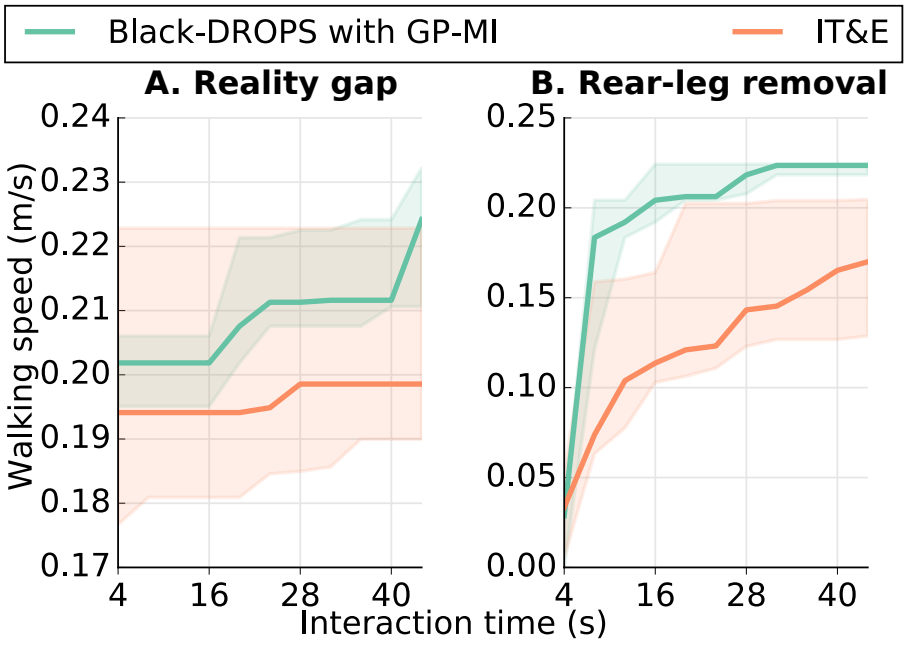


Figure 5.6: Results for the physical hexapod locomotion task (5 replicates of each scenario). The lines are median values and the shaded regions the 25^{th} and 75^{th} percentiles. **A.** Improving a policy for the intact robot (crossing the reality gap): Black-DROPS with GP-MI finds a highly-effective policy (about $0.22m/s$) in less than 30 seconds of interaction time, whereas IT&E is not able to substantially improve the initial policy. **B.** Rear-leg removal damage case: Black-DROPS with GP-MI allows the damaged robot to walk effectively after only 16 to 30 seconds of interaction time and finds higher-performing policies than IT&E ($0.21m/s$ vs $0.15m/s$ in the 8^{th} episode).

Overall, Black-DROPS with GP-MI was able to successfully learn working policies even though the dimensionality of the state and the action space of the hexapod robot is 48D and 18D respectively (see Fig. 5.5 for an example). In addition, in the rear leg damage case, Black-DROPS always tried safer policies than IT&E that too often executed policies that would cause the robot to fall over. A video of our algorithm running on the damaged hexapod is available at https://youtu.be/HFkZkhGGzTo.

5.5 Conclusion and Discussion

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

The main issue of our approach is the quadratic computational complexity of the prediction of the GPs, which makes it practical only for low interaction times.

For example, Black-DROPS with GP-MI required 24 hours on a modern 16-core computer for 26 episodes of the pendubot task. Possible solutions include using scalable GPs (Park and Apley, 2017; Deisenroth and Ng, 2015) (Liu et al., 2018) or to stop using GPs and make use of recent advances in neural networks with uncertain predictions (Gal et al., 2016; Gal and Ghahramani, 2016; Higuera et al., 2018; Chua et al., 2018).

The way we compute the long-term predictions (*i.e.*, by chaining model predictions) requires that predicted states (the output of the GPs) are fed back to the prior simulator. This can cause the simulator to crash because there is no guarantee that the predicted state, that possibly makes sense in the real world, will make sense in the prior model; especially when the two models (prior and real) differ a lot and when there are obstacles and collisions involved. This also holds for most other prior-based methods (Lee et al., 2017; Saveriano et al., 2017; Cutler and How, 2015), but it is not easily seen in simple and unbounded systems. On the contrary, we observed this phenomenon a few times in our hexapod experiments. Using the prior simulator just as a reference and not mixing prior and real data is a direction of future work.

Finally, Black-DROPS with GP-MI brings closer trial-and-error and diagnosis-based approaches for robot damage recovery as it successfully combines: (a) diagnosis (Isermann, 2006) (*i.e.*, identifying the likeliest robot model from data), (b) prior knowledge of possible damages/different conditions that a robot may face and (c) trial-and-error learning.

Chapter 6

Collaborations: Bayesian Optimization for Micro-Data Reinforcement Learning

The results and text of this chapter have been partially published in the following articles.

Articles:

- Papaspyros, V., **Chatzilygeroudis, K.**, Vassiliades, V. and Mouret, J.-B., 2016. *Safety-Aware Robot Damage Recovery Using Constrained Bayesian Optimization and Simulated Priors.* **Workshop "Bayesian Optimization: Black-box Optimization and Beyond"**, NIPS (Papaspyros et al., 2016).
- Paul, S., **Chatzilygeroudis, K.**, Ciosek, K., Mouret, J.-B., Osborne, M.A. and Whiteson, S., 2018. *Alternating Optimisation and Quadrature for Robust Control.* **AAAI Conference on Artificial Intelligence** (Paul et al., 2018).
- Paul, S., **Chatzilygeroudis, K.**, Ciosek, K., Mouret, J.-B., Osborne, M.A. and Whiteson, S., 2018. *Robust Reinforcement Learning with Bayesian Optimisation and Quadrature*, Under review in *Bayesian Optimization JMLR Special Issue.*
- Pautrat, R., **Chatzilygeroudis, K.** and Mouret, J.-B., 2018. *Bayesian Optimization with Automatic Prior Selection for Data-Efficient Direct Policy Search.* **International Conference on Robotics and Automation** (Pautrat et al., 2018).

Other contributors:

- Vassilis Vassiliades (Post-doc)
- Vaios Papaspyros (PhD student at EPFL — he was an intern)
- Rémi Pautrat (Master student)
- Supratik Paul (PhD student at Oxford)
- Kamil Ciosek (Post-doc at Oxford)

- Shimon Whiteson (Associate Prof. at Oxford)
- Michael A. Osborne (Associate Prof. at Oxford)
- Jean-Baptiste Mouret (Thesis supervisor)

Author contributions:

- For the safety-aware paper: VP, **KC** and JBM organized the study. VP performed the experiments. VP and **KC** wrote the code. **KC**, VP, VV and JBM analyzed the results and wrote the paper.
- For the AAAI paper: SP and SW organized the study. SP performed the simulated experiments. **KC** performed the physical robot experiments. SP wrote the code. **KC** wrote the code for the physical robot experiments. SP, **KC**, KaCi, JBM, MO and SW analyzed the results and wrote the paper.
- For the JMLR paper: SP, **KC** and SW organized the study. SP performed the simulated experiments. **KC** performed the physical robot experiments. SP wrote the code. **KC** wrote the code for the physical robot experiments. SP, **KC**, KaCi, JBM, MO and SW analyzed the results and wrote the paper.
- For the ICRA paper: RP, **KC** and JBM organized the study. RP and **KC** wrote the code and performed the experiments. **KC**, RP and JBM analyzed the results and wrote the paper.

6.1 Introduction

In the previous chapters, we saw how learning models of the dynamics or pre-computing priors can speed-up the learning process on the physical robot. Model-based policy search approaches require very low interaction times, but do not scale well with the dimensionality of the state and action space. Although in chapter 5 we saw how we could use parametrized simulators to mitigate this issue, our proposed approach still required considerable computation time. Pre-computing priors, on the other hand, requires expert and task-specific knowledge in order to define the type of the prior.

As we discussed in chapter 2, an alternative way to perform data-efficient policy is to use Bayesian optimization as a direct policy search algorithm. Therefore, here we will focus on Bayesian optimization and how safety constraints, robustness and multiple prior information sources can be incorporated in order to further reduce the interaction time for learning with physical robots. It is important to note that I was not the leading author in any of these works and I am grateful for the excellent collaborators that I had the chance to have.

6.2 Safety-aware Intelligent Trial-and-Error for Robot Damage Recovery

6.2.1 Introduction

As discussed in chapter 3, micro-data RL is an appealing approach to solve the robot damage recovery challenge. We already saw that one of the most promising approaches for data-efficient robot damage recovery is the recently introduced Intelligent Trial-and-Error algorithm (IT&E) (Cully et al., 2015). It combines two ideas: (1) a Bayesian optimization (BO) algorithm (Shahriari et al., 2016) that optimizes a reward function, because it is a generic, data-efficient policy search algorithm (Calandra et al., 2015), and (2) a behavior-performance map generated before the mission with a simulation of the intact robot, which acts both as a prior for the Bayesian optimization algorithm and as a dimension reduction algorithm. This combination allowed a damaged 6-legged robot to find a new gait in about a dozen of trials (less than 2 minutes), and a robotic arm to overcome several blocked joints in a few minutes (Cully et al., 2015).

Unfortunately, trial-and-error approaches, like IT&E, are likely to damage the robot even more because they will often try behaviors that are too extreme for the mechanical design. More generally, learning algorithms will push robots to their limits because they focus solely on maximizing the reward intake[1]. This issue is especially concerning for expensive prototypes like the iCub robot (Nori et al., 2015; Tsagarakis et al., 2007): these robots are too expensive (around 250k euros for the iCub) and too fragile to try risky behaviors.

While recent methods, like SafeOPT (Berkenkamp et al., 2016), tackle this issue successfully, they require an initial safe set of parameters, that is hard to estimate in an unknown damage setting. In this work, we extend the IT&E algorithm (Cully et al., 2015) by adding safety constraints (Gardner et al., 2014) and automatically computing priors over the safety of controller parameters, so that the probability of breaking the robot during the learning process is as low as possible. We evaluate our algorithm using a simulated damaged iCub robot.

6.2.2 Safety-aware Intelligent Trial & Error Algorithm

The first step of IT&E is to create a low-dimensional behavior-performance map with a simulation of the intact robot. This step is achieved with an evolutionary algorithm called MAP-Elites (Mouret and Clune, 2015), which, instead of searching for a single, best solution, like standard optimization algorithms, searches for the highest-performing individual for each point in a user-defined space. This user-defined space is often called the behavior space, because the dimensions of variation (behavior descriptors) usually measure behavioral characteristics. For example, by defining one dimension for each

[1]Interestingly, learning algorithms also push robot simulators to their limits as they often exploit simulation inaccuracies.

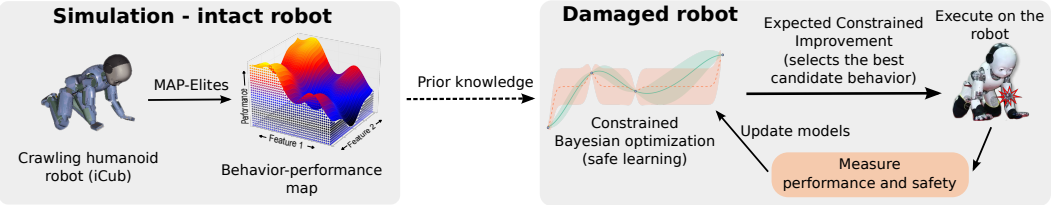


Figure 6.1: Overview of the safety-aware IT&E algorithm. The algorithm first creates a behavior-performance map through MAP-Elites in simulation. This is fed as prior knowledge to a constrained BO procedure. The best candidate behavior is executed on the damaged robot while measuring the contact point forces and crawling speed. Finally, the Gaussian process models are updated.

leg's fraction of time spent on the ground, MAP-Elites produces a wide variety of walking gaits for a hexapod robot (Cully et al., 2015).

To search for the best behavior on the damaged robot, IT&E alters the classical BO scheme (Cully et al., 2015) by (1) searching a behavior in the map, instead of searching the best policy parameters, and (2) modeling the difference between the performance predicted by the map ($M(\cdot)$) and the actual performance, instead of directly modeling the performance function. Thus, at each step of the adaptation algorithm, the BO procedure selects the most promising behavior from the map, executes it on the damaged robot, observes the performance, and updates its predictions accordingly. More precisely, the performance function — a function of the policy parameters, $f(\boldsymbol{\theta})$ — is modeled as a Gaussian process (GP) with priors (see Eq. (2.28),(2.29)).

This approach leads to short adaptation times in several experiments with damaged robots (Cully et al., 2015), but it has a serious limitation that prevents it from being used with expensive robots: it lacks safety constraints, that is, nothing prevents the robot from trying dangerous behaviors. Because we have access to a simulator of the robot, one could think that the safety of a potential behavior could be evaluated using the simulation; however, this is likely to not be sufficient because high-performing controllers for the intact robot might have considerably different behavior and often be harmful on the damaged robot. More often than not, such behaviors are highly dependent on the robot's legs, wheels, etc., the failure of which would result in a radically different outcome in the damaged case. IT&E having no prior knowledge about the damage or any safety limitations, will most likely attempt more than a few of these dangerous behaviors. These behaviors constitute a big risk for further damaging fragile robots. To make matters worse, a damage recovery algorithm would have to compensate for the reality gap (Koos et al., 2013b; Jakobi et al., 1995) as well. In particular, even for intact robots, the behaviors contained in the behavior-performance maps are not likely to be reproduced exactly on the physical robot.

We address these issues by introducing a safety-aware IT&E algorithm (sIT&E; see Fig. 6.1). sIT&E uses a constrained BO procedure (Gardner et al., 2014) in which each user-defined safety constraint, $c_i(\boldsymbol{\theta})$, is modeled as a separate GP. The next sample is selected by optimizing the Expected

Constrained Improvement (ECI) acquisition function (Gardner et al., 2014) (see also Section 2.6):

$$ECI(\hat{\boldsymbol{\theta}}) = EI(\hat{\boldsymbol{\theta}}) \prod_{i=1}^{N} p(c_i(\hat{\boldsymbol{\theta}}) \geq 0) \tag{6.1}$$

where $\hat{\boldsymbol{\theta}}$ is the candidate point, $c_i(\hat{\boldsymbol{\theta}}), i \in \{1, \cdots, N\}$ are the N constraint functions and $EI(\hat{\mathbf{x}})$ is the standard expected improvement (Gardner et al., 2014).

The differences between sIT&E (see Alg.14) and IT&E are as follows: (1) the behavior descriptor in MAP-Elites is augmented with extra dimensions for each safety constraint, so that sIT&E will have a good estimate of the safe regions (i.e., where all inequality safety constraints are fulfilled); for example, these dimensions could be torque or IMU measurements that should not exceed a specific threshold; (2) during the on-line adaptation step, sIT&E optimizes for performance while also guiding the search through the safe regions.

Algorithm 14 Safety-aware Intelligent Trial-and-Error (sIT&E)

procedure sIT&E
  Before mission (intact robot in simulation):
    Create Behavior-Performance Map via MAP-Elites storing safety info
  while in mission **do**
    if significant performance drop **then**
      Adaptation Step (via M-CBO Algorithm)
    end if
  end while
end procedure
procedure Map-Based Constrained Bayesian Optimization (M-CBO)
  $\forall \boldsymbol{\theta} \in map$:
    $p(f(\boldsymbol{\theta})|\boldsymbol{\theta}) = N(m(\boldsymbol{\theta}), k(\boldsymbol{\theta}, \boldsymbol{\theta}))$
    $p(c_i(\boldsymbol{\theta})|\boldsymbol{\theta}) = N(m_{c_i}(\boldsymbol{\theta}), k_{c_i}(\boldsymbol{\theta}, \boldsymbol{\theta}))$, $i \in \{1, \cdots, N\}$
  while stopping criteria not met **do**
    $\boldsymbol{\theta}_{n+1} = \text{argmax}_{\boldsymbol{\theta}}\, ECI(\boldsymbol{\theta}|\boldsymbol{D}_{1:n}, \boldsymbol{C}_{1:n})$
    $\{c_1(\boldsymbol{\theta}_{n+1}), \ \ldots \ , c_N(\boldsymbol{\theta}_{n+1}), f(\boldsymbol{\theta}_{n+1})\} = execute\_behavior(\boldsymbol{\theta}_{n+1})$
    $\boldsymbol{D}_{1:n+1} = \{f(\boldsymbol{\theta}_{n+1}), \boldsymbol{D}_{1:n}\}$
    $\boldsymbol{C}_{1:n+1} = \{\langle c_1(\boldsymbol{\theta}_{n+1}), \ \ldots \ , c_N(\boldsymbol{\theta}_{n+1})\rangle, \boldsymbol{C}_{1:n}\}$
    Update GPs for the objective function/safety constraints
  end while
end procedure

6.2.3 Crawling humanoid robot experiments

To evaluate our algorithm, we use a simulated iCub robot (Nori et al., 2015; Tsagarakis et al., 2007) performing a crawling task. Learning how to crawl could

prove especially useful in humanoid robot damage recovery, where attempting to walk again might constitute a big risk for further damages or be infeasible altogether (*e.g.*, traversing a short or tight tunnel). Furthermore, solving this task serves as a stepping stone towards damage recovery for more advanced tasks (*e.g.*, walking).

To generate a diversity of behaviors with MAP-Elites, we augment an initially 4D behavior descriptor, defined as the fraction of time each arm/leg spent on the ground, with a safety dimension that encodes the sum of contact point forces. sIT&E optimizes for crawling speed and is constrained by a safety threshold for the sum of contact point forces. This threshold is determined after conducting several preliminary experiments in order to understand the correlation between the robot's behavior and the contact point forces at high crawling speeds. To optimize the acquisition function, we iterate over all the points in the behavior-performance map (which contains approximately 1500 behaviors), and select a behavior that is estimated to be the most promising above the safety threshold.

We compare 3 algorithms in terms of the best safe performance observed and unsafe trials attempted: (1) IT&E maximizing crawling speed; (2) a multi-objective (Deb, 2014) IT&E algorithm (MO-IT&E; based on the Expected Hypervolume Improvement (Yang et al., 2015; Hupkens et al., 2015)), that maximizes the crawling speed and minimizes the sum of contact point forces, therefore, building a Pareto front from which the safest behavior can be chosen; and (3) sIT&E maximizing crawling speed within the safe region as described above. We test 4 damage conditions: (1) locked shoulder joint, (2) locked hip joint, (3) locked shoulder joint & angled elbow, and (4) combination of 2 & 3.

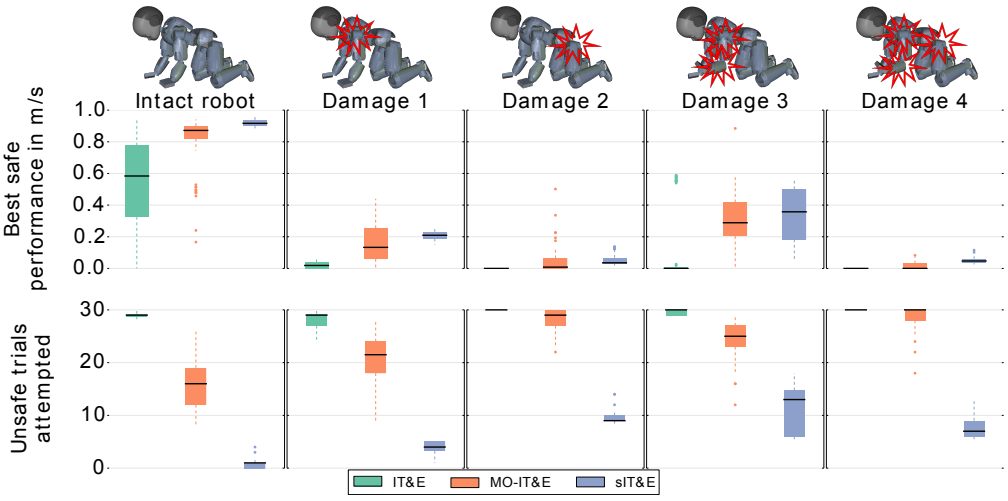


Figure 6.2: Comparison between IT&E, MO-IT&E and sIT&E. Each algorithm is ran 20 times for 4 different behavior-performance maps. We report the best safe performance (i.e., the fastest crawling speed within the safe region observed during the learning process) (upper row) and number of unsafe trials attempted (violating the constraints) (lower row). Horizontal black lines represent medians.

To avoid depending on a single behavior-performance map, we use 4

independently-generated maps and run each algorithm 20 times for 30 trials. We use the squared exponential kernel with $\sigma = 0.1$ and a GP noise value of 0.01. When using IT&E, the median number of dangerous trials is approximately equal to 29 (out of 30) in all damage settings (Fig. 6.2, lower row). For MO-IT&E, this number decreases, but it is still greater than 22. In contrast, sIT&E requires less than 10 unsafe trials for damages 1, 2, and 4, and 14 for damage 3. Pairwise comparisons indicate that the results are highly significant ($p < 0.0001$, Mann-Whitney U test). In terms of safe performance (crawling speed in m/s), sIT&E dominates over both IT&E and MO-IT&E in all damage settings (Fig. 6.2, upper row), with the results being statistically significant ($p < 0.001$) in all cases apart from damage 3.

All experiments were conducted using the limbo framework (Cully et al., 2018). A supplementary video is available at `https://youtu.be/8esrj-7WhsQ`.

6.2.4 Conclusion

Our experiments show that the vanilla IT&E algorithm finds high-performing behaviors in a few trials, but most of the behaviors tested, including the best, final ones, are unsafe for the robot. Since the multi-objective approach searches for a set of Pareto-optimal trade-offs, it can find safe and high-performing behaviors; however, this approach still tests many unsafe behaviors during the learning phase. By contrast, the sIT&E algorithm finds gaits that are both safe and high-performing with only a handful of unsafe trials. Thanks to this property, we are confident that sIT&E is less likely to damage the real iCub than IT&E or BO. Overall, this work shows that safety is a critical component for any robot learning algorithm and that constrained BO can provide a good basis to design algorithms that are both data-efficient and safe.

6.3 Alternating Optimization and Quadrature for Robust Control

6.3.1 Introduction

As it should already be apparent, a key consideration when applying RL to a physical setting is the risk and expense of running trials. Another consideration is the robustness of the learned policies. Since it is typically infeasible to test a policy in all contexts, it is difficult to ensure it works as broadly as intended. Fortunately, policies can often be tested in a simulator that exposes key *environment variables* – state features that are unobserved and randomly determined by the environment in a physical setting but are controllable in the simulator. For example, the state of a hexapod's leg can vary from being fully operational to being broken off due to any damage taken while operating in its environment. This section considers how to use environment variables to help learn robust policies.

Although running trials in a simulator is cheaper and safer than running physical trials, the computational cost of each simulated trial can still be quite high (especially when the simulator is of high precision). The challenge then is to develop algorithms that are sample efficient, *i.e.*, that minimize the number of such trials. We have already seen that *Bayesian optimization* is a sample-efficient approach that has been successfully applied to RL in multiple domains ((Lizotte et al., 2007; Martinez-Cantin et al., 2007, 2009; Cully et al., 2015; Calandra et al., 2015; Pautrat et al., 2018)).

A naïve approach would be to randomly sample the environment variable in each trial, so as to estimate expected performance. However, this approach (1) often requires testing each policy in a prohibitively large number of scenarios, and (2) is not robust to *significant rare events* (SREs), *i.e.*, it fails any time there are rare events that substantially affect expected performance. For example, rare localization errors may mean that a robot is much nearer to an obstacle than expected, increasing the risk of a collision. Since collisions are so catastrophic, avoiding them is key to maximizing expected performance, even though the factors contributing to the collision occur only rarely. In such cases, the naïve approach will not see such rare events often enough to learn an appropriate response.

Instead, we present our approach called *alternating optimisation and quadrature* (ALOQ) (Paul et al., 2018) specifically aimed towards learning policies that are robust to these rare events while being as sample efficient as possible. The main idea is to *actively* select the environment variables (instead of sampling them) in a simulator. We use a *Gaussian process* (GP) (Rasmussen and Williams, 2006) to model returns as a function of *both* the policy and the environment variables and then, at each step, alternately use BO and *Bayesian quadrature* (BQ) (O'Hagan, 1991; Rasmussen and Ghahramani, 2003) to select a policy and environment setting, respectively, to evaluate.

A simulator is only an imperfect representation of reality and policies that are exclusively learnt in simulation can exploit badly modelled aspects of reality and end up performing significantly worse on the physical system. This poses the challenge of bridging the *reality gap* ((Jakobi et al., 1995; Jakobi, 1997; Koos et al., 2013b)), *i.e.*, transferring policies from the simulator to the real system without any significant downgrade in performance. In this section we also present an extension to ALOQ, called *transferable ALOQ* (TALOQ), that automatically trades off the cost and accuracy of running a trial on the simulator against running it on the physical system.

We first apply ALOQ to a number of primarily simulated problems, where the reality gap does not pose any significant challenge. Our results demonstrate that ALOQ learns better and faster than multiple baselines. Next, we demonstrate that TALOQ can use an imperfect simulator to learn policies that bridge the reality gap, while requiring significantly fewer trials than if we were to train with ALOQ solely on the robot.

6.3.2 Problem Setting

Here, we adapt the generic problem formulation of Section 2.2 to our specific case. We assume access to a computationally expensive simulator that takes as input a policy $\pi_{\boldsymbol{\theta}} \in \Theta$ and environment variable $\boldsymbol{\vartheta} \in \mathcal{B}$ and produces as output the return $f(\boldsymbol{\theta}, \boldsymbol{\vartheta}) \in \mathbb{R}$, where both Θ and $\mathcal{B}$ belong to some compact sets in $\mathbb{R}^{d_\pi}$ and $\mathbb{R}^{d_{\boldsymbol{\vartheta}}}$, respectively. We assume that this simulator is highly accurate and poses little to no reality gap and thus policies learnt on this simulator transfer well to the physical system.

We also assume access to $p(\boldsymbol{\vartheta})$, the probability distribution over $\boldsymbol{\vartheta}$. $p(\boldsymbol{\vartheta})$ may be known a priori, or it may be a posterior distribution estimated from whatever physical trials have been conducted. For example, this could be based on the knowledge of some human expert, or as in the case of a mobile robot it could be the robot's belief about its distance from potential obstacles. Our objective is to find the optimal policy parameters $\boldsymbol{\theta}^*$:

$$\pi_{\boldsymbol{\theta}^*} = \operatorname*{argmax}_{\boldsymbol{\theta}} \bar{f}(\boldsymbol{\theta}) = \operatorname*{argmax}_{\boldsymbol{\theta}} \mathbb{E}_{\boldsymbol{\vartheta}}[f(\boldsymbol{\theta}, \boldsymbol{\vartheta})]. \tag{6.2}$$

In Section 6.3.4, we present a method for a setting in which we assume that the simulator is imperfect but still useful enough that running trials on it is informative about the performance on the physical system. We assume that during training $\boldsymbol{\vartheta}$ can be controlled on the physical system, which is not very restrictive in practice. For example, it is easy to run a policy on a mobile robot and deduce from its trajectory whether it would have collided with an obstacle without actually placing the obstacle on its path.

In this case, we define the return $f = f(\boldsymbol{\theta}, \boldsymbol{\vartheta}, \delta)$, where $\delta \in \{0, 1\}$ is an indicator function that denotes whether the return is from a simulated trial or from the physical system. Our objective is to find the optimal policy parameters $\boldsymbol{\theta}^*$:

$$\pi_{\boldsymbol{\theta}^*} = \operatorname*{argmax}_{\boldsymbol{\theta}} \bar{f}(\boldsymbol{\theta}) = \operatorname*{argmax}_{\boldsymbol{\theta}} \mathbb{E}_{\boldsymbol{\vartheta}}[f(\boldsymbol{\theta}, \boldsymbol{\vartheta}, 1)], \tag{6.3}$$

where $\delta = 1$ since we want the policy that maximizes returns on the physical system, not the simulator.

6.3.3 ALOQ

Naïve method A simple approach is to apply BO directly on $\bar{f}(\boldsymbol{\theta}) = \mathbb{E}_{\boldsymbol{\vartheta}}[f(\boldsymbol{\theta}, \boldsymbol{\vartheta})]$ and attempt to estimate $\pi_{\boldsymbol{\theta}^*}$. Formally, this approach models $\bar{f}$ as a GP with a zero mean function and a suitable covariance function $k(\boldsymbol{\theta}, \boldsymbol{\theta}')$. Since f is expensive to evaluate, at the lth BO iteration only one $f(\boldsymbol{\theta}_l, \boldsymbol{\vartheta}_l)$ with $\boldsymbol{\vartheta}_l \sim p(\boldsymbol{\vartheta})$ is evaluated in the simulator and $\bar{f}(\boldsymbol{\theta}_l)$ is approximated by this one sample Monte Carlo estimate. This approach will almost surely fail since the estimates of $\bar{f}(\boldsymbol{\theta})$ are going to be extremely noisy, especially in the presence of SREs.

By contrast, our method ALOQ (see Algorithm 15) models $f(\boldsymbol{\theta}, \boldsymbol{\vartheta})$ as a GP: $f \sim GP(m, k)$, acknowledging both its inputs. Given a dataset $\boldsymbol{D}_{1:n} =$

$\{(\boldsymbol{\theta}_i, \boldsymbol{\vartheta}_i, f(\boldsymbol{\theta}_i, \boldsymbol{\vartheta}_i))\}_{i=1}^n$, the main idea behind ALOQ is to use a BO acquisition function to select $\boldsymbol{\theta}_{n+1}$ for evaluation and then use a BQ acquisition function to select $\boldsymbol{\vartheta}_{n+1}$, conditioning on $\boldsymbol{\theta}_{n+1}$.

Selecting $\boldsymbol{\theta}_{n+1}$ requires maximizing a BO acquisition function (6.7) on $\bar{f}(\boldsymbol{\theta})$, which requires estimating $\bar{f}(\boldsymbol{\theta})$, together with the uncertainty associated with it. Fortunately BQ is well suited for this since it can use the GP to estimate these quantities.

Once $\boldsymbol{\theta}_{n+1}$ is chosen, ALOQ selects $\boldsymbol{\vartheta}_{n+1}$ by minimizing a BQ acquisition function (6.8) quantifying the uncertainty about $\bar{f}(\boldsymbol{\theta}_{n+1})$. After $(\boldsymbol{\theta}_{n+1}, \boldsymbol{\vartheta}_{n+1})$ is selected, ALOQ evaluates it on the simulator and updates the GP with the new datapoint $(\boldsymbol{\theta}_{n+1}, \boldsymbol{\vartheta}_{n+1}, f(\boldsymbol{\theta}_{n+1}, \boldsymbol{\vartheta}_{n+1}))$. Our estimate of $\boldsymbol{\theta}^*$ is thus:

$$\hat{\boldsymbol{\theta}}^* = \operatorname*{argmax}_{\boldsymbol{\theta}} \mathbb{E}[\bar{f}(\boldsymbol{\theta}) | \boldsymbol{D}_{1:n+1}]. \tag{6.4}$$

Although the approach described so far actively selects $\boldsymbol{\theta}$ and $\boldsymbol{\vartheta}$ through BO and BQ, it is unlikely to perform well in practice. A key observation is that the presence of SREs, which we seek to address with ALOQ, implies that the scale of f varies considerably, e.g., returns in case of collision versus no collision. This non-stationarity cannot be modelled with our stationary kernel. Therefore, we must transform the inputs to ensure stationarity of f. In particular, we employ *beta warping*, *i.e.*, we transform the inputs using beta CDFs with parameters (α, β) (Snoek et al., 2014). The CDF of the beta distribution on the support $0 < x < 1$ is given by:

$$\text{BetaCDF}(x, \alpha, \beta) = \int_0^x \frac{u^{\alpha-1}(1-u)^{\beta-1}}{B(\alpha, \beta)} \mathrm{d}u, \tag{6.5}$$

where $B(\alpha, \beta)$ is the beta function. The beta CDF is particularly suitable for our purpose as it can model a variety of warpings based on the settings of only two parameters (α, β). ALOQ transforms each dimension of $\boldsymbol{\theta}$ and $\boldsymbol{\vartheta}$ independently and treats the corresponding (α, β) as hyperparameters. In the rest of this chapter, we assume that we are working with the transformed inputs.

While the resulting algorithm should be able to cope with SREs, the $\hat{\boldsymbol{\theta}}^*$ that it returns at each iteration may still be poor, since our BQ evaluation of $\bar{f}(\boldsymbol{\theta})$ leads to a noisy approximation of the true expected return. This is particularly problematic in high dimensional settings. To address this, *intensification* ((Bartz-Beielstein et al., 2005; Hutter et al., 2009)), *i.e.*, reevaluation of selected policies in the simulator, is essential. Therefore, ALOQ performs two simulator calls at each iteration. In the first evaluation, $(\boldsymbol{\theta}_{n+1}, \boldsymbol{\vartheta}_{n+1})$ is selected via the BO/BQ scheme described above. In the second stage, $(\hat{\boldsymbol{\theta}}^*, \boldsymbol{\vartheta}^*)$ is evaluated, where $\hat{\boldsymbol{\theta}}^* \in \boldsymbol{\theta}_{1:n+1}$ is selected using (6.4) and $\boldsymbol{\vartheta}^* | \hat{\boldsymbol{\theta}}^*$ using the BQ acquisition function (6.8).

In the rest of this section, we provide more details on how ALOQ computes $\bar{f}(\boldsymbol{\theta})$, the acquisition functions it uses, and its convergence and complexity.

Algorithm 15 ALOQ

Input A simulator that outputs $f = f(\boldsymbol{\theta}, \boldsymbol{\vartheta})$, initial dataset $\boldsymbol{D}_{1:n}$, the maximum number of function evaluations L, and a GP prior. Comparison of
for $n = l+1, l+3, ..., L-1$ **do**
    Update the beta warping parameters and transform the inputs.
    Update the GP to condition on the (transformed) dataset $\boldsymbol{D}_{1:n}$
    Use (6.6) to estimate $p(\bar{\boldsymbol{\theta}}|\boldsymbol{D}_{1:n-1})$
    Use the BO acquisition function (6.7) to select $\boldsymbol{\theta}_n = \text{argmax}_{\boldsymbol{\theta}}\, \alpha_{\text{ALOQ}}(\boldsymbol{\theta})$
    Use the BQ acquisition function (6.8) to select $\boldsymbol{\vartheta}_n|\boldsymbol{\theta}_n$
    Perform a simulator call with $(\boldsymbol{\theta}_n, \boldsymbol{\vartheta}_n)$ to obtain $f(\boldsymbol{\theta}_n, \boldsymbol{\vartheta}_n)$ and update $\boldsymbol{D}_{1:n-1}$ to $\boldsymbol{D}_{1:n}$
    Find $\hat{\boldsymbol{\theta}}^* = \text{argmax}_{\boldsymbol{\theta}_i}\, \bar{f}(\boldsymbol{\theta}_i)|\boldsymbol{D}_{1:n}$ and $\boldsymbol{\vartheta}^*|\hat{\boldsymbol{\theta}}^*$ using the BQ acquisition function (6.8).
    Perform a second simulator call with $(\hat{\boldsymbol{\theta}}^*, \boldsymbol{\vartheta}^*)$ to obtain $f(\hat{\boldsymbol{\theta}}^*, \boldsymbol{\vartheta}^*)$ and update $\boldsymbol{D}_{1:n}$ to $\boldsymbol{D}_{1:n+1}$
end for
Output $\boldsymbol{\theta}^* = \text{argmax}_{\boldsymbol{\theta}_i}\, \bar{f}(\boldsymbol{\theta}_i) \mid \boldsymbol{D}_{1:L} \quad i = 1, 2, ..., L$

Computing $\bar{f}(\boldsymbol{\theta})$: For discrete $\boldsymbol{\vartheta}$ with support $\{\boldsymbol{\vartheta}_1, \boldsymbol{\vartheta}_2, \ldots, \boldsymbol{\vartheta}_{N_{\boldsymbol{\vartheta}}}\}$, the estimate of the mean μ and variance σ^2 for $\bar{f}(\boldsymbol{\theta}) \mid \boldsymbol{D}_{1:n}$ is:

$$\mu = \frac{1}{N_{\boldsymbol{\vartheta}}} \sum_{i=1}^{N_{\boldsymbol{\vartheta}}} \mathbb{E}[f(\boldsymbol{\theta}, \boldsymbol{\vartheta}_i)|\boldsymbol{D}_{1:n}] \tag{6.6a}$$

$$\sigma^2 = \frac{1}{N_{\boldsymbol{\vartheta}}^2} \sum_{i=1}^{N_{\boldsymbol{\vartheta}}} \sum_{j=1}^{N_{\boldsymbol{\vartheta}}} \text{Cov}[f(\boldsymbol{\theta}, \boldsymbol{\vartheta}_i)|\boldsymbol{D}_{1:n}, f(\boldsymbol{\theta}, \boldsymbol{\vartheta}_j)|\boldsymbol{D}_{1:n}], \tag{6.6b}$$

where $f(\boldsymbol{\theta}, \boldsymbol{\vartheta})$ is the prediction from the GP. For continuous $\boldsymbol{\vartheta}$, we apply Monte Carlo quadrature. Although this requires sampling a large number of $\boldsymbol{\vartheta}$ and evaluating the corresponding $f(\boldsymbol{\theta}, \boldsymbol{\vartheta}) \mid \boldsymbol{D}_{1:n}$, it is feasible since we evaluate $f(\boldsymbol{\theta}, \boldsymbol{\vartheta}) \mid \boldsymbol{D}_{1:n}$, not from the expensive simulator, but from the computationally cheaper GP.

BO acquisition function for $\boldsymbol{\theta}$: A modified version of the UCB acquisition function is a natural choice since using (6.6) we can compute it easily as

$$\alpha_{\text{ALOQ}}(\boldsymbol{\theta}) = \mu(\bar{f}(\boldsymbol{\theta})|\boldsymbol{D}_{1:n}) + \kappa\sigma(\bar{f}(\boldsymbol{\theta})|\boldsymbol{D}_{1:n}), \tag{6.7}$$

and set $\boldsymbol{\theta}_{n+1} = \text{argmax}_{\boldsymbol{\theta}}\, \alpha_{\text{ALOQ}}(\boldsymbol{\theta})$.

BQ acquisition function for $\boldsymbol{\vartheta}$: BQ can be viewed as performing policy evaluation in our approach, *i.e.*, estimating the expected return of a given $\boldsymbol{\theta}$. Since the presence of SREs leads to high variance in the returns associated with any given policy, it is of critical importance that we minimize the uncertainty

Algorithm 16 TALOQ

Input A simulator that outputs $f = f(\boldsymbol{\theta}, \boldsymbol{\vartheta})$, initial dataset $\boldsymbol{D}_{1:n}$, the maximum number of function evaluations L, and a GP prior.
for $n = l+1, l+3, ..., L-1$ **do**
  Update the beta warping parameters and transform the inputs.
  Update the GP to condition on the (transformed) dataset $\boldsymbol{D}_{1:n}$
  Use (6.6) to estimate $p(\bar{f}|\boldsymbol{D}_{1:n-1})$
  Use the BO acquisition function (6.7) to select $\boldsymbol{\theta}_n = \text{argmax}_{\boldsymbol{\theta}}\, \alpha_{\text{ALOQ}}(\boldsymbol{\theta})$
  Compute γ as per (6.9) and set $\delta_n = 1$ if $\gamma > k$, and $\delta_n = 0$ otherwise.
  Use the BQ acquisition function (6.10) to select $\boldsymbol{\vartheta}_n|\boldsymbol{\theta}_n, \delta_n$
  Perform an evaluation with $(\boldsymbol{\theta}_n, \boldsymbol{\vartheta}_n, \delta_n)$ to obtain $f(\boldsymbol{\theta}_n, \boldsymbol{\vartheta}_n, \delta_n)$ and update $\boldsymbol{D}_{1:n-1}$ to $\boldsymbol{D}_{1:n}$
  Find $\hat{\boldsymbol{\theta}}^* = \text{argmax}_{\boldsymbol{\theta}_i}\, \bar{f}(\boldsymbol{\theta}_i|\boldsymbol{D}_{1:n})$
  Compute γ as per (6.9) and set $\delta^* = 1$ if $\gamma > k$, and $\delta^* = 0$ otherwise.
  Use the BQ acquisition function (6.10) to select $\boldsymbol{\vartheta}^*|(\hat{\boldsymbol{\theta}}^*, \delta^*)$
  Perform a second evaluation with $(\hat{\boldsymbol{\theta}}^*, \boldsymbol{\vartheta}^*, \delta^*)$ to obtain $f(\hat{\boldsymbol{\theta}}^*, \boldsymbol{\vartheta}^*, \delta^*)$ and update $\boldsymbol{D}_{1:n}$ to $\boldsymbol{D}_{1:n+1}$
end for
Output $\boldsymbol{\theta}^* = \text{argmax}_{\boldsymbol{\theta}_i}\, \bar{f}(\boldsymbol{\theta}_i) \mid \boldsymbol{D}_{1:L} \quad i = 1, 2, ..., L$

associated with our estimate of the expected return. We formalize this objective through our BQ acquisition function for $\boldsymbol{\vartheta}$: ALOQ selects $\boldsymbol{\vartheta}_{n+1} \mid \boldsymbol{\theta}_{n+1}$ by minimizing the posterior variance of $\bar{f}(\boldsymbol{\theta}_{n+1})$, yielding:

$$\boldsymbol{\vartheta}_{n+1}|\boldsymbol{\theta}_{n+1} = \underset{\boldsymbol{\vartheta}}{\text{argmin}}\, \mathbb{V}(\bar{f}(\boldsymbol{\theta}_{n+1})|\boldsymbol{D}_{1:n}, \boldsymbol{\theta}_{n+1}, \boldsymbol{\vartheta}). \tag{6.8}$$

Properties of ALOQ Thanks to convergence guarantees for BO using α_{UCB} (Srinivas et al., 2010), ALOQ converges if the BQ scheme on which it relies also converges. Unfortunately, to the best of our knowledge, existing convergence guarantees (Kanagawa et al., 2016; Briol et al., 2015) apply only to BQ methods that do not actively select points, as (6.8) does. Of course, we expect such active selection to only improve the rate of convergence of our algorithms over non-active versions. However, our empirical results in Section 6.3.5 show that in practice ALOQ efficiently optimizes policies in the presence of SREs across a variety of tasks.

ALOQ's computational complexity is dominated by an $\mathcal{O}(n^3)$ matrix inversion, where n is the sample size of the dataset $\boldsymbol{D}$. This cubic scaling is common to all BO methods involving GPs. The BQ integral estimation in each iteration requires only GP predictions, which are $\mathcal{O}(n^2)$.

6.3.4 TALOQ

We now consider the setting where there exists a reality gap between the simulator and the physical system. In this case, at each iteration we face the

additional choice to perform the evaluation on the simulator or on the physical system, which is even more expensive. In this setting, we model the return as a GP with three inputs $(\boldsymbol{\theta}, \boldsymbol{\vartheta}, \delta)$, where $\delta = 0$ corresponds to the simulator and $\delta = 1$ to the physical system. Observing the return of any $(\boldsymbol{\theta}, \boldsymbol{\vartheta})$ in the simulator gives us some information about the corresponding return on the physical system. This is dependent on the lengthscale along the δ dimension, which is long if the reality gap is small and short if the gap is big.

In TALOQ (see Algorithm 16) we follow the same general method as ALOQ: At iteration n, $\boldsymbol{\theta}_n$ is selected with the UCB acquisition function presented in (6.7). Next, we select $\delta_n|\boldsymbol{\theta}_n$. We define the relative reduction in uncertainty:

$$\gamma = \frac{\mathbb{V}[\bar{f}(\boldsymbol{\theta}_n)|\boldsymbol{D}_{1:n-1}] - \operatorname{argmin}_{\boldsymbol{\vartheta}} \mathbb{V}[\bar{f}(\boldsymbol{\theta}_n)|\boldsymbol{D}_{1:n-1}, \boldsymbol{\theta}_n, \boldsymbol{\vartheta}, \delta_n = 0]}{\mathbb{V}[\bar{f}(\boldsymbol{\theta}_n)|\boldsymbol{D}_{1:n-1}] - \operatorname{argmin}_{\boldsymbol{\vartheta}} \mathbb{V}[\bar{f}(\boldsymbol{\theta}_n)|\boldsymbol{D}_{1:n-1}, \boldsymbol{\theta}_n, \boldsymbol{\vartheta}, \delta_n = 1]}, \tag{6.9}$$

and set $\delta_n = 0$ if $\gamma > k$, and $\delta_n = 1$ otherwise. Here $k \in [0, 1]$ is a hyper-parameter that should be set based on the relative cost of running physical trials against simulated trials – a large k encourages more physical evaluations, and vice versa. Since physical trials are likely to be far more expensive than simulated trials, experimentally we have observed that setting k between 0.002 to 0.01 can lead to high sample efficiency with respect to the physical system. This formulation of γ as a ratio of the relative reduction in variance ensures that it is less problem dependent than a formulation with absolute reduction.

Finally, we select $\boldsymbol{\vartheta}_n|(\boldsymbol{\theta}_n, \delta_n)$ using the BQ acquisition function given in (6.8) with the slight modification that the conditioning set now includes δ_n:

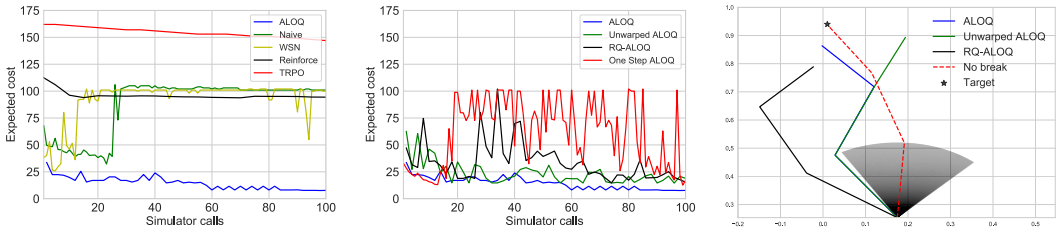
$$\boldsymbol{\vartheta}_n|\boldsymbol{\theta}_n, \delta_n = \operatorname*{argmin}_{\boldsymbol{\vartheta}} \mathbb{V}(\bar{f}(\boldsymbol{\theta}_n)|\boldsymbol{D}_{1:n-1}, \boldsymbol{\theta}_n, \boldsymbol{\vartheta}, \delta_n). \tag{6.10}$$

6.3.5 Experimental Results

6.3.5.1 ALOQ

We use a simulated robot arm control task and a hexapod locomotion task to evaluate ALOQ and we compare to several baselines: 1) the *naïve* method described in Section 6.3.3; 2) the method of (Williams et al., 2000), which we refer to as *WSN*; 3) the simple policy gradient method REINFORCE (Williams, 1992), and 4) the state-of-the-art policy gradient method *trust region policy optiisation* (TRPO) (Schulman et al., 2015). To show the importance of each component of ALOQ, we also perform experiments with ablated versions, namely: 1) *Random Quadrature ALOQ* (RQ-ALOQ), in which $\boldsymbol{\vartheta}$ is sampled randomly from $p(\boldsymbol{\vartheta})$ instead of being chosen actively; 2) *unwarped ALOQ*, which does not perform beta warping of the inputs; and 3) *one-step ALOQ*, which does not use intensification. All plotted results are the median of 20 independent runs.

Robotic Arm Simulator We evaluate ALOQ's performance on a robot control problem implemented in a kinematic simulator. The goal is to configure



(a) Comparison of expected cost of ALOQ with baselines **(b)** Comparison of expected cost of ablations of ALOQ **(c)** Learned arm configurations for ALOQ ablations

Figure 6.3: Performance and learned configurations on the robotic arm joint breakage task.

each of the three controllable joints of a robot arm such that the tip of the arm gets as close as possible to a predefined target point.

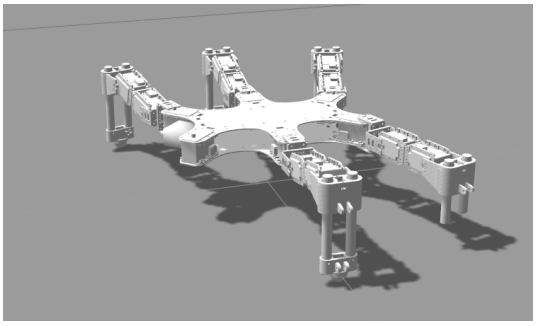
In this setting, we assume that some settings of the first joint carry a 5% probability of it breaking, which consequently incurs a large cost. Minimizing cost thus entails getting as close to the target as possible, while minimizing the probability of the joint breaking.

Results show that ALOQ performs better than all the baselines and the ablations (Fig. 6.3a 6.3b 6.3c). In particular, the Naïve baseline, WSN[2], TRPO and REINFORCE seem to converge to a suboptimal policy since they have not witnessed any SREs.

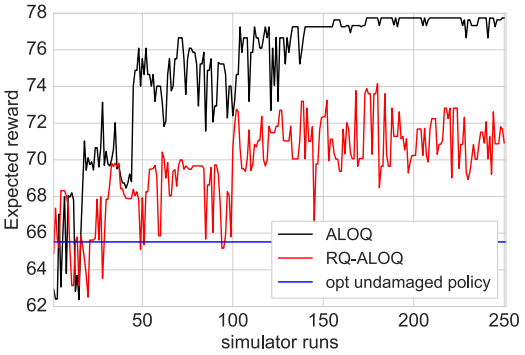
Hexapod Locomotion Task As robots move from fully controlled environments to more complex ones, they have to face the inevitable risk of getting damaged. However, it may be expensive or even impossible to decommission a robot whenever any damage condition prevents it from completing its task. Hence, it is desirable to develop methods that enable robots to recover from failure.

Intelligent trial and error (IT&E) (Cully et al., 2015) has been shown to recover from various damage conditions and thereby prevent catastrophic failure. Before deployment, IT&E uses the simulator to create an archive of diverse and locally high performing policies for the intact robot that are mapped to a lower dimensional *behavior space*. If the robot becomes damaged after deployment, it uses BO to quickly find the policy in the archive that has the highest performance on the damaged robot. However, it can only respond after damage has occurred. Though it learns quickly, performance may still be poor while learning during the initial trials after damage occurs. To mitigate this effect, we propose to use ALOQ to learn in simulation the policy with the highest expected performance across the possible damage conditions. By deploying this policy, instead of the policy that is optimal for the intact robot, we can minimize in expectation the negative effects of damage in the period before IT&E has learned to recover.

[2]Since $\boldsymbol{\vartheta}$ is continuous in this setting, and WSN requires discrete $\boldsymbol{\vartheta}$, it was run on a slightly different version with $\boldsymbol{\vartheta}$ discretised by 100 equidistant points.



(a) Hexapod with a shortened and a missing leg.



(b) Expected reward

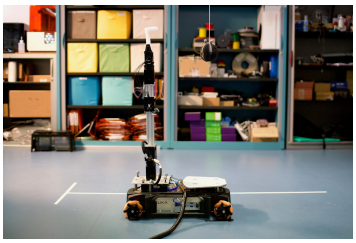
Figure 6.4: Hexapod locomotion problem.

We consider a hexapod locomotion task with a setup similar to that of (Cully et al., 2015) to demonstrate this experimentally. The objective is to cross a finish line a fixed distance from its starting point. Failure to cross the line leads to a large negative reward, while the reward for completing the task is inversely proportional to the time taken. It is possible that a subset of the legs may be damaged or broken when deployed in a physical setting. For our experiments, we assume that, based on prior experience, any of the front two or back two legs can be shortened or removed with probability of 10% and 5% respectively, independent of the other legs, leading to 81 possible configurations. We excluded the middle two legs from our experiment as their failure has relatively little impact on the hexapod's movement. The configuration of the six legs acts as our environment variable. Figure 6.4a shows one such setting.

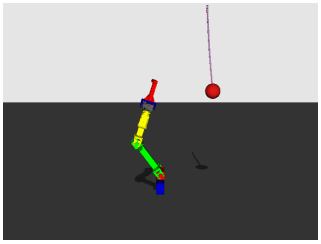
We applied ALOQ to learn the optimal policy given these damage probabilities, but restricted the search to the policies in the archive created by (Cully et al., 2015)[3]. Figure 6.4b shows that ALOQ finds a policy with much higher expected reward than RQ-ALOQ. It also shows the policy that generates the maximum reward when none of the legs are damaged or broken ('opt undamaged policy').

To check if a policy learnt by ALOQ in simulation transfers successfully, we ran an experiment where we used ALOQ to learn a policy entirely in simulation and then deployed the learnt policy on a real hexapod. In order to limit the number of physical trials required to evaluate the ALOQ policy, we limited the possibility of damage to the rear two legs. The learnt policy performed better than the opt undamaged policy (which was also learnt only in simulation) on the physical robot because it optimized performance on the rare configurations that matter most for expected return (e.g., either leg shortened). However, the performance of both policies were worse than in simulation due to the reality gap. For example, in simulation the undamaged hexapod traveled more than $1m$ with both the ALOQ policy and the opt undamaged policy, while in reality it traveled no more than $0.75m$. These results underscore the need for an

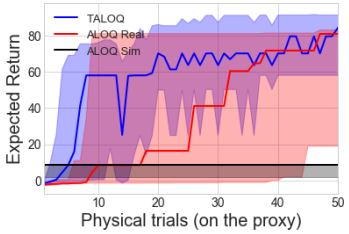
[3]The policies were generated by MAP-Elites (Mouret and Clune, 2015) using a model of the intact robot in simulation using DART (Lee et al., 2018).



(a) Physical robot



(b) Simulated robot



(c) Comparison of performance

Figure 6.5: Experimental setup for TALOQ: The objective is to learn a policy for striking the ball and achieving a particular velocity. (a) The physical setup, (b) the setup in simulation, and (c) comparison of the performance of TALOQ against the baselines on the simulated version of the experiment.

algorithm like TALOQ to bridge the reality gap. In the next section we present an experiment using TALOQ with a robotic arm without using MAP-Elites.

6.3.5.2 TALOQ

To evaluate TALOQ, we applied it to a physical robotic arm. The robot has actuators across 4 joints that control the position of its end effector in 3D space. The objective is to strike a ball hanging from a rope with its end effector and achieve a target velocity. The reward function is the sum of two components: a squared exponential function of the velocity of the ball with a sharp peak at $0.75m/s$, and a cost that increases linearly with the minimum distance of the end effector from the centre of the ball (observed throughout the whole trajectory). The policy space is 5-D consisting of 4 joint angles and a frequency parameter for the arm to oscillate between the initial configuration and the specified joint angles. As an SRE, we assume that the arm is damaged with probability 5%, making the actuator for the third joint unresponsive and stuck in the initial configuration. This can cause policies that perform well on the fully functional arm to perform poorly on the damaged arm. We used DART (Lee et al., 2018) to design a simulator for our experiment [4] (Fig. 6.5b) and dynamixel pro actuators for the real robot (Fig. 6.5a).

Policies learnt by ALOQ only in simulation with a budget of 200 trials achieved a median return of 4.23 on the robot across 20 random replicates, which shows that it transfers very poorly. This is unsurprising since the reality gap can be significant due to the modelling errors and the differences in the controller. Compared to this, across 3 random replicates with a total budget of 200 trials, TALOQ needed 13, 34, and 48 physical trials to find policies with expected returns of 28.18, 89.18, and 93.60 (see Table 6.1). This demonstrates that combining physical and simulated trials during the learning process using TALOQ learns a much better policy. A video of one learnt policy is available at `https://youtu.be/R8Ss-dhDCmo`.

[4] More precisely, we used the robot_dart wrapper.

To evaluate TALOQ more extensively, we devised a simulated version of the experiment. We treated the simulator from the previous set up as a proxy for reality, and developed another version which we treated as the simulator. To create a reality gap between the two, instead of the ball hanging from the rope we modelled it as a pendulum. We also set the mass of the ball to $1kg$ compared to $60g$ in the proxy. Since simulated trials are relatively cheap, we ran 20 random replicates and after each iteration evaluated the expected return of the policy TALOQ specified as optimal.

The results show that the policy learnt only in simulation using ALOQ (ALOQ Sim) does not transfer at all to the proxy for reality, since the reality gap is quite large (Fig. 6.5c; note that the x-axis is the number of physical trials). Learning a policy with ALOQ exclusively in this proxy (ALOQ Real) performs better since there is no reality gap. However it does not do as well as TALOQ since it does not leverage the information provided by the simulator. This shows that TALOQ can effectively leverage simulated trials to improve sample efficiency on the physical system.

Physical trials	Replicate # 1	2	3
1	-2.19	-6.07	-4.98
≤ 10	7.99	-6.07	35.85
≤ 20	28.18	-6.07	35.85
≤ 35	28.18	89.18	56.40
≤ 50	28.18	89.18	93.60

Table 6.1: Evolution of the expected reward on the physical robot using TALOQ

6.3.6 Conclusion

In this section, we presented ALOQ, a novel approach using BO and BQ to perform sample-efficient RL in a way that is robust to the presence of significant rare events. We also presented TALOQ, an extension to ALOQ that addresses the problem of the reality gap by actively selecting when to evaluate on the physical system instead of the imperfect simulator.

We empirically evaluated ALOQ on different simulated tasks involving a robotic arm simulator, and a hexapod locomotion task. Our results demonstrated that ALOQ outperforms multiple baselines, including related methods proposed in the literature. Further, ALOQ is computationally efficient and does not require any restrictive assumptions to be made about the environment variables. We also showed that TALOQ can be used to successfully learn a policy that is robust to the SREs while addressing the challenge posed by the reality gap.

6.4 Bayesian Optimization with Automatic Prior Selection

6.4.1 Introduction

Throughout the manuscript we have seen that RL is a promising approach that can allow robots to adapt to new tasks (*e.g.*, a new tool) and new contexts

(*e.g.*, a damage (Cully et al., 2015; Chatzilygeroudis et al., 2018a)), but only if this adaptation happens in a few minutes: contrary to simulated worlds (*e.g.*, games), where thousands (if not millions) of simulations can be evaluated, the number of trials in robotics hardware is limited by the energetic autonomy of the robot and the need to perform the task as soon as possible to be useful (Mouret, 2016).

As discussed in this chapter and chapter 2, Bayesian Optimization is a promising approach because it can work with continuous action and state spaces, contrary to classic RL algorithms (Deisenroth et al., 2013), and because it scales well with the dimension of the state space, contrary to model-based policy search algorithms (*e.g.*, PILCO (Deisenroth and Rasmussen, 2011) or Black-DROPS (Chatzilygeroudis et al., 2017)). As we already saw, one of the main strategies for micro data reinforcement learning is to leverage prior knowledge; when BO is used for direct policy search, priors on the reward function can be added by using a non-constant mean function in the model, that is, by modeling the difference between the observations and the prior instead of modeling the observations directly (Ko et al., 2007; Cully et al., 2015; Chatzilygeroudis et al., 2018b).

In this section, we are interested in using BO when (1) several priors are available and, (2) we do not know beforehand which prior corresponds to the current context. A typical situation is a robot that knows how to solve a task in context A, B, and C (priors) and needs to learn to solve it in context D, while not knowing whether D is closer to A, B, or C. For some tasks, a perception system might recognize the right context (Plagemann et al., 2008), but in many others only the observations of the reward function can allow the robot to determine what prior is the most plausible. For instance, a walking robot could learn that a surface is slippery by observing that it matches the predictions that correspond to a prior for slippery floors, but it is often difficult to predict the slipperiness of a surface by only looking at it.

Our main insight is that we can compare two priors by computing the likelihood of the combination "prior + model" so that we can select the prior that matches the best the observations. Our second insight is that this prior selection can be elegantly incorporated as an acquisition function of a BO procedure, so that we select the next point to test by balancing between the expected improvement and the likelihood of the model used to compute the expected improvement. We demonstrate our approach on a simulated and physical 6-legged robot that faces different damage conditions and different environments.

6.4.2 Combining Likelihood and Expected Improvement

Like we have already seen in similar cases, we model the objective function, $R(\boldsymbol{\tau}|\boldsymbol{\theta})$[5], by a Gaussian process $f(\boldsymbol{\theta})$ with a mean function $m(\cdot)$ and a covariance

[5] Abusing a bit the notation, we will right it as $R(\boldsymbol{\theta})$ from now on.

function $\kappa(\cdot,\cdot)$:

$$f(\boldsymbol{\theta}) \sim \mathcal{GP}(m(\boldsymbol{\theta}), \kappa(\boldsymbol{\theta}, \boldsymbol{\theta}'))$$

We also assume that some prior knowledge on the objective function is available before starting the optimization. In that case, we can write this information with a prior function $\mathcal{P}$ and update the equations of the GP accordingly as in Eq. (2.28). We use the squared exponential kernel for the GPs as defined in Eq. (2.23).

As discussed in the background section, the next point $\boldsymbol{\theta}$ where the objective function should be evaluated is found by maximizing an *acquisition function*, that is, a function that leverages the model (both the variance and the mean) to predict the most promising point. A function that is often used for this is the Expected Improvement (EI) (Warren B. Powell, 2012; Shahriari et al., 2016) (see also Sec. 2.6.1).

Choosing the best prior can be seen as a problem of model selection (since the prior is part of the model), which is effectively achieved by comparing the likelihood of alternative models (Rasmussen and Williams, 2006):

$$\begin{aligned} &P(\boldsymbol{f}(\boldsymbol{\theta}_{1..n}) \mid \boldsymbol{\theta}_{1..n}, \boldsymbol{\mathcal{P}}(\boldsymbol{\theta}_{1..n})) = \\ &\frac{1}{\sqrt{(2\pi)^n |K|}} \exp\Big(-\frac{1}{2}\big(\boldsymbol{F}(\boldsymbol{\theta}_{1..n}) \cdots \\ &\cdots - \boldsymbol{\mathcal{P}}(\boldsymbol{\theta}_{1..n})\big)^T \boldsymbol{K}^{-1} \big(\boldsymbol{F}(\boldsymbol{\theta}_{1..n}) - \boldsymbol{\mathcal{P}}(\boldsymbol{\theta}_{1..n})\big)\Big) \end{aligned} \tag{6.11}$$

where $\boldsymbol{F}(\boldsymbol{\theta}_{1..n}) = \{R(\boldsymbol{\theta}_1), \ldots, R(\boldsymbol{\theta}_n)\}$ and $\boldsymbol{\mathcal{P}}(\boldsymbol{\theta}_{1..n}) = \{\mathcal{P}(\boldsymbol{\theta}_1), \ldots, \mathcal{P}(\boldsymbol{\theta}_n)\}$.

Intuitively, we could select the prior that corresponds to the best likelihood, then compute the expected improvement for this model. However, we would risk to select an "over-pessimistic" prior at the beginning of the optimization, because the first observations (which are often random points) are likely to be low-performing — if random points were likely to be high-performing, there would be no need for learning. In essence, if we have not yet observed any high-performing solutions, then the likeliest prior is a prior for which every solution is low-performing.

We therefore need to balance between the likelihood of the prior 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 F.

Let us assume that the objective function only takes discrete values, in which case the likelihood is a probability. Considering n observations $R(\boldsymbol{\theta}_1), \ldots, R(\boldsymbol{\theta}_n)$, we introduce the indicator function $\mathbb{1}_{f(\boldsymbol{\theta}_1)=R(\boldsymbol{\theta}_1),\ldots,f(\boldsymbol{\theta}_n)=R(\boldsymbol{\theta}_n)}$ which equals to 1 when the predictions match exactly the observations, and we define the Expected Improvement for a prior $\mathcal{P}$:

$$\begin{aligned} \mathrm{EIP}(\boldsymbol{\theta}, \mathcal{P}) &= \mathbb{E}(\mathrm{I}(\boldsymbol{\theta}) \times \mathbb{1}_{f(\boldsymbol{\theta}_1)=R(\boldsymbol{\theta}_1),\ldots,f(\boldsymbol{\theta}_n)=R(\boldsymbol{\theta}_n)}) \\ &= \mathbb{E}(\max(0, f(\boldsymbol{\theta}) - M_n) \times \mathbb{1}_{f(\boldsymbol{\theta}_1)=R(\boldsymbol{\theta}_1),\ldots,f(\boldsymbol{\theta}_n)=R(\boldsymbol{\theta}_n)}) \end{aligned} \tag{6.12}$$

But as the predicted value $f(\boldsymbol{\theta}) \sim \mathcal{N}(\mu_n(\boldsymbol{\theta}), \sigma_n^2(\boldsymbol{\theta}))$ only depends on the samples $\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_n$, the observations $\boldsymbol{F}(\boldsymbol{\theta}_{1..n})$ and the deterministic function $\mathcal{P}$, it is independent of the original distribution $\boldsymbol{f}(\boldsymbol{\theta}_{1..n}) \sim \mathcal{N}(\boldsymbol{\mathcal{P}}(\boldsymbol{\theta}_{1..n}), \boldsymbol{K})$. Thus the two factors inside the expectation are two independent variables and can be split:

$$\begin{aligned} \text{EIP}(\boldsymbol{\theta}, \mathcal{P}) &= \mathbb{E}(\max(0, f(\boldsymbol{\theta}) - M_n)) \\ &\quad \times \mathbb{E}(\mathbb{1}_{f(\boldsymbol{\theta}_1)=R(\boldsymbol{\theta}_1),\ldots,f(\boldsymbol{\theta}_n)=R(\boldsymbol{\theta}_n)}) \\ &= \text{EI}(\boldsymbol{\theta}) \times P(\ \boldsymbol{f}(\boldsymbol{\theta}_{1..n})=\boldsymbol{F}(\boldsymbol{\theta}_{1..n}) \mid \boldsymbol{\theta}_{1..n}, \mathcal{P}) \end{aligned} \tag{6.13}$$

This new function can be extended afterwards to the case where R takes continuous values; the likelihood becomes a density probability function, but the EIP can still be defined as the product of the expected improvement with the likelihood:

$$\text{EIP}(\boldsymbol{\theta}, \mathcal{P}) = \text{EI}(\boldsymbol{\theta}) \times P(\boldsymbol{f}(\boldsymbol{\theta}_{1..n}) \mid \boldsymbol{\theta}_{1..n}, \boldsymbol{\mathcal{P}}(\boldsymbol{\theta}_{1..n})) \tag{6.14}$$

When we have m priors $\mathcal{P}_1, \cdots, \mathcal{P}_m$, the Most Likely Expected Improvement (MLEI) acquisition function can then be defined as:

$$\text{MLEI}(\boldsymbol{\theta}, \mathcal{P}_1, \cdots, \mathcal{P}_m) = \max_{p \in \mathcal{P}_1, \cdots, \mathcal{P}_m} \text{EIP}(\boldsymbol{\theta}, p) \tag{6.15}$$

The MLEI acquisition function can be used like any other acquisition function in the BO algorithm. Please note that the likelihood has to be evaluated only once for each model (that is, once for each prior), and not for every point $\boldsymbol{\theta}$ (see Algo. 17). We use the C++-11 Limbo library for the BO implementation (Cully et al., 2018).

Algorithm 17 Bayesian Optimization with MLEI

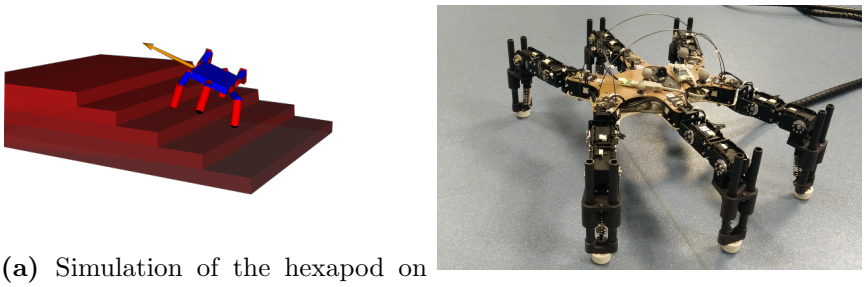
procedure BOMULTIPLEPRIORS
Input: m priors $\mathcal{P}_1, \dots, \mathcal{P}_m$, an objective function F
Output: An approximation of the maximum of F
    Initialize m Gaussian processes $f_1, \dots, f_m$ with the m priors and the kernel function κ:
    $\forall i \in \{1, \dots, m\}, f_i(\boldsymbol{\theta}) \sim \mathcal{N}(\mathcal{P}_i(\boldsymbol{\theta}), \kappa(\boldsymbol{\theta}, \boldsymbol{\theta}'))$
    $n \leftarrow 1$
    while $n \leq maxIterations$ **do**
        for $i = 1..m$ **do**
            $l \leftarrow computeLikelihood(f_i, \boldsymbol{\theta}_1, \dots, \boldsymbol{\theta}_{n-1})$
            $\boldsymbol{s}_i \leftarrow \text{argmax}_{\boldsymbol{\theta} \in \Theta} \text{EI}(\boldsymbol{\theta})$
                $= \text{argmax}_{\boldsymbol{\theta} \in \Theta}(\mathbb{E}(\max(0, f_i(\boldsymbol{\theta}) - M_n)))$
            $\text{EIP}(\boldsymbol{s}_i, \mathcal{P}_i) \leftarrow l \times \text{EI}(\boldsymbol{s}_i)$
        end for
        $\boldsymbol{\theta}_n, p_n \leftarrow \text{argmax}_{i=1..m} \text{EIP}(\boldsymbol{s}_i, \mathcal{P}_i)$
        Evaluate $R(\boldsymbol{\theta}_n)$ on the robot
        Update the m Gaussian processes with the new observation $R(\boldsymbol{\theta}_n)$
        $n \leftarrow n + 1$
    end whilereturn $\max_{n=1..maxIterations} R(\boldsymbol{\theta}_n)$
end procedure

6.4.3 Experimental Results

We evaluate the MLEI acquisition function in a similar context as in Cully *et al.* (Cully et al., 2015): a 6-legged robot is either damaged in an unknown way or introduced to an unknown environment and BO is used to find an alternative walking gait that works in spite of the unforeseen situation. However, while Cully *et al.* used a single prior (walking on a flat surface with an intact robot), we introduce many other priors that correspond either to potential damages (*e.g.*, a missing leg) or to different terrains (*e.g.*, stairs). We test the learning algorithm with priors corresponding to the actual situation, but also in situations that are not fully covered by any prior.

Robot and policy The robot has identical legs with 3 DOFs per leg (Fig. 6.6). One DOF (θ_1) controls the horizontal movements of the leg (from back to front) whereas the two others (θ_2 and θ_3) control the elevation of the leg. Each one of these DOFs is controlled by an open-loop oscillator defined with 3 parameters (Cully et al., 2015): an amplitude, a phase, and a duty cycle (proportion of time in which the angle is in an extreme position). The second vertical angle θ_3 is constrained to take values between $-\theta_2$ and $-\theta_2 + \frac{\pi}{4}$, so that the inferior member (the "tibia") remains vertical or at most at an angle of $\frac{\pi}{4}$ with the vertical line. Thus, the whole gait of the robot can be defined with $6 \times 3 \times 3 = 54$ parameters. All simulations[6] of the robot are performed with

[6] https://github.com/resibots/robot_dart



(a) Simulation of the hexapod on stairs

(b) Real hexapod robot

Figure 6.6: The 6-legged robot used in the experiments.

the Dynamic Animation and Robotics Toolkit (DART) (Lee et al., 2018) in a world with gravity were the simulated robot is similar to the intact, physical hexapod.

Reward function In all the experiments, the reward function is the distance covered by the 6-legged robot in a virtual corridor with a width of $1m$ (the width of the robot is about $40\,cm$). As soon as the robot gets out of the corridor, the evaluation is stopped; it is also stopped after $10\,s$ if the robot stays in the corridor. Compared to more traditional reward functions, for instance the distance covered in 10 seconds, our reward function encourages more the robot to follow a straight line, even if it means that the gait is slower. Similar results were however obtained with the average walking speed as a reward.

Prior generation All the priors are 6-dimensional behavior-reward maps computed for a simulated 6-legged robot in different environments or with the damaged robot (*e.g.*, with a missing leg). These behavior-reward maps are created beforehand using the MAP-Elites algorithm (Cully et al., 2015; Mouret and Clune, 2015), which is a recent evolutionary algorithm designed to generate thousands of different high-performing control policies (see also Chapter 3).

We use one of the behavior descriptors proposed in Cully *et al.* (Cully et al., 2015): the body orientation, which captures how often the body of the robot is tilted in each direction[7]. More formally, simulating each policy leads to a 6-dimensional vector that contains the proportion of time that the body of the

[7]Similar results were obtained with other behavioral descriptors.

robot has a positive and negative pitch, yaw and roll:

$$\boldsymbol{BOF} = \begin{bmatrix} \frac{1}{K}\sum_{k=1}^{K} \mathbb{1}_{\Xi(k)>0.005\pi} \\ \frac{1}{K}\sum_{k=1}^{K} \mathbb{1}_{\Xi(k)<-0.005\pi} \\ \frac{1}{K}\sum_{k=1}^{K} \mathbb{1}_{\Psi(k)>0.005\pi} \\ \frac{1}{K}\sum_{k=1}^{K} \mathbb{1}_{\Psi(k)<-0.005\pi} \\ \frac{1}{K}\sum_{k=1}^{K} \mathbb{1}_{\Phi(k)>0.005\pi} \\ \frac{1}{K}\sum_{k=1}^{K} \mathbb{1}_{\Phi(k)<-0.005\pi} \end{bmatrix} \tag{6.16}$$

where the duration of the gait of the robot is divided in K intervals of 15 ms, Ξ, Ψ and Φ are the pitch, roll and yaw of the torso of the robot, respectively, $\mathbb{1}$ is the indicator function which returns 1 if and only if its argument is true, and angles between $\pm 0.005\pi$ are ignored.

This quantity is rounded so that it can only take values in $\{0, 0.2, 0.4, 0.6, 0.8\}$ and so the set of all the body orientation factors is finite and contains $5^6 = 15625$ elements that can be organized in a map.

For the purpose of the experiments, 15 behavior-performance maps have been created for each of the possible environments (priors). Each one of these maps was created with a run of the MAP-Elites algorithm for 24 hours on a 16-core Xeon computer. We used the Sferes C++ library (Mouret and Doncieux, 2010).

Experiment 1 — Adaptation to stairs in simulation In our first set of experiments, the intact robot needs to adapt to unknown environments. We generated 15 behavior-performance maps (*i.e.*, 15 priors for the GP) for each of the four following environments:

- flat ground;
- easy stairs (steps with height: $4cm$, width: $1.2m$, depth: $50cm$);
- medium stairs (steps with height: $5cm$, width: $1.2m$, depth: $20cm$);
- hard stairs (steps with height: $7.5cm$, width: $1.2m$, depth: $25cm$).

We compare the following acquisition functions for BO:

- EI with a single prior coming from a simulated robot on flat ground – this corresponds to the original IT&E experiments (Cully et al., 2015);
- EI with a single prior, randomly chosen among the available priors at each iteration.
- EI with a single prior coming from a simulated robot on the actual stairs (when available) – this corresponds to the ideal case, in which we know the right prior;
- MLEI with a prior selected at each iteration among the available priors (flat ground, easy, medium and hard stairs).

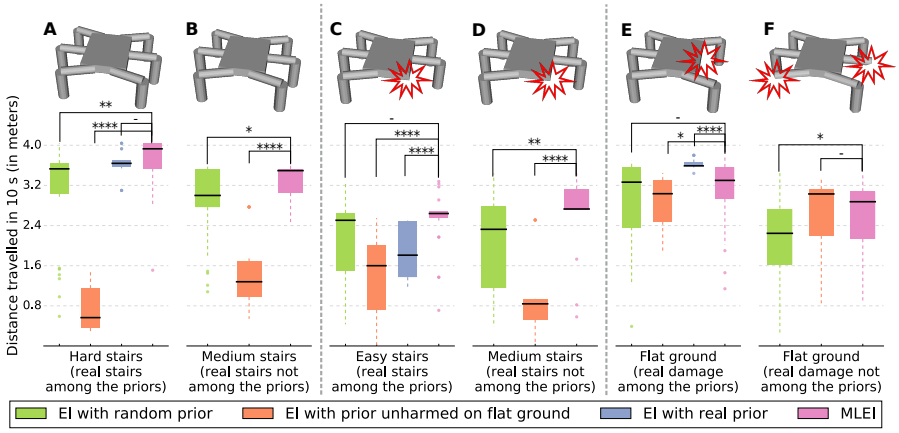


Figure 6.7: Comparison in simulation of MLEI with other acquisition functions and choices of prior (EI with a randomly selected prior, EI with a prior generated on an unharmed robot on flat ground and EI with the prior corresponding to the real stairs or damage) on the 6-legged robot learning to climb stairs and/or to recover from damages after 5 iterations of BO and with 30 replicates of each experiment. (A) and (B): the robot is on unknown stairs with no damage and the real stairs can be among the priors (A) or not (B). (C) and (D): the robot is on unknown stairs with unknown damages and the priors are only on stairs not on damages (the actual stairs can be among the priors (C) or not (D)). (E) and (F): the robot is on flat ground with unknown damages and the real damage can be among the priors (E) or not (F). The number of stars indicates that the p-value, obtained using the Mann-Whitney-Wilcoxon test, is below 0.0001, 0.001, 0.01 and 0.05 respectively.

For the MLEI and EI with random priors experiments, we randomly choose 5 priors (*i.e.*, 5 maps) for each possible environment, leading to a unique set of priors for each MLEI experiment and for each experiment with randomly chosen priors. Please note that several priors correspond to the same situation, which is interesting because some maps might be of higher-quality than others, even if they have been generated with the same environment.
We test two situations:

1. adaptation to hard stairs when the hard stairs are part of the priors given to MLEI (and to random selection) — $5 \times 4 = 20$ priors to select from;

2. adaptation to medium stairs, with the medium stairs removed from the priors given to MLEI (and to random selection) — $5 \times 3 = 15$ priors to select from.

In these two situations, the robot is the same in the prior and in the adaptation experiment (there is no "reality gap").

The results (Fig. 6.7A-B) show that MLEI allows the robot to learn high-performing gaits for the stairs, even when the stairs used for the learning experiments are not present in the set of priors (Fig. 6.7B): when the right prior is accessible, MLEI finds it; when it is not accessible, it can still leverage other priors and use BO to find a good behavior while using other priors. In the two tested cases, MLEI clearly outperforms the random selection of priors and the method using the flat ground prior, which means that MLEI selects priors

correctly and that these priors help the learning process. Surprisingly, MLEI also outperforms the EI with a "perfect" prior (Fig. 6.7A): this is because MLEI has access to 5 priors for the hard stairs (in addition to the 15 other priors) and therefore can select the best of them, whereas each EI experiment has access to a single prior for the considered stairs (and the best controller for each map is different). The relatively good performance of the random selection of priors is likely to stem from the fact that this algorithm has access to a much higher diversity of behaviors than EI with flat ground as a prior (that is, to the original IT&E), which makes it more likely to find a behavior that works in the tested situation.

Experiment 2 — Adaptation to stairs and damages in simulation In this second experiment, we evaluate if the robot can adapt to unforeseen damage conditions, with and without stairs, with and without priors about the damage conditions. For each of the 6 legged removed, we generated 15 priors with MAP-Elites (with a robot on flat ground), leading to $(6 + 1) \times 15 = 105$ priors (6 damage conditions + intact robot). Like in the previous experiments, the set of available priors is made of 5 random maps (out of the 15 generated priors) for each situation.

We compare the same methods as before in four situations that cover different combinations of environmental and body-related priors:

1.a adaptation to damage with priors about stairs (no prior about damage), and when the actual stairs are among the priors — 20 priors to select from;

1.b adaptation to damage with priors about stairs (no prior about damage), and when the actual stairs are not among the priors — 15 priors to select from;

2.a adaptation to damage with priors about the damage conditions, on flat ground, when the actual damage (left middle leg removed) is among the priors — $7 \times 5 = 35$ priors to select from;

2.b adaptation to damage with priors about the damage conditions, on flat ground, when the actual damage (front right leg and middle left leg shortened) is not among the priors — $7 \times 5 = 35$ priors to select from.

The results (Fig. 6.7C-F and supplementary video[8]) show that MLEI can find compensatory gaits on stairs while using priors computed with the intact robot. When the real stairs are among the priors (Fig. 6.7C), MLEI outperforms the EI with the right stairs because (1) since the robot is damaged, the most helpful prior is not always the prior that corresponds to the correct stairs (*e.g.*, the prior that corresponds to the hard stairs might be more conservative and be more helpful when the robot is damaged); (2) like before, MLEI has access

[8]Available at: https://youtu.be/xo8mUIZTvNE

to more priors, which makes it more likely to have a policy in one of the map that can compensate for the damage.

When the actual stairs are not in the priors, MLEI still outperforms the two other approaches (Fig. 6.7D). MLEI can also take advantage of priors about the damage condition whether the damage is included in the priors or not (Fig. 6.7E-F): when the actual damage conditions is among the priors, MLEI leads to higher-performing solutions than EI with the intact robot as a prior; when the damage condition is not among the prior, MLEI performs the same as EI with the intact robot as a prior. These results are consistent with (Cully et al., 2015), which shows that an intact robot can be an effective prior to adapt to damage.

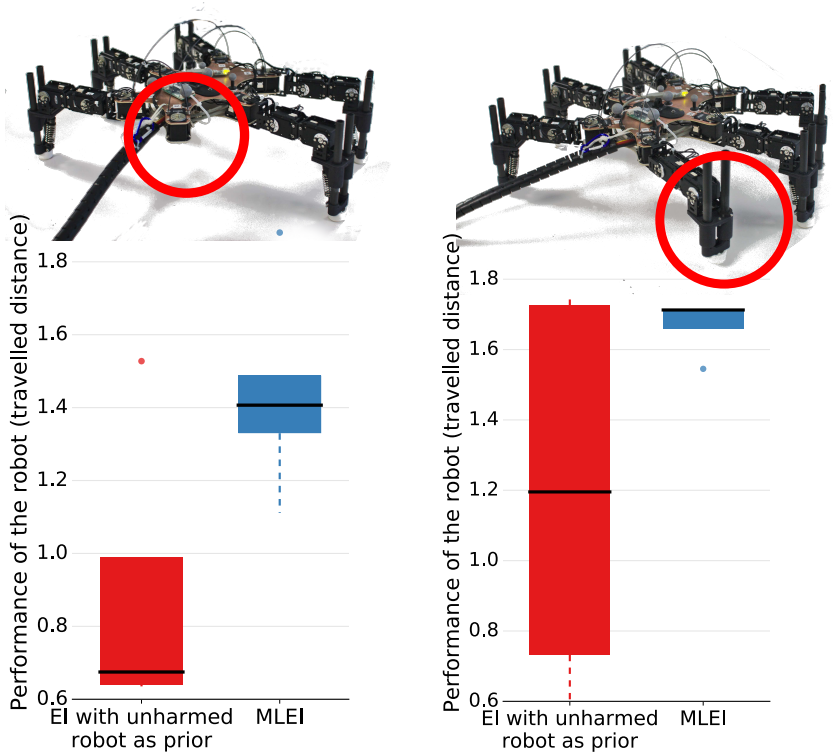
Experiment 3 — Adaptation to damage with a physical robot In this experiment, we use $(6+1) \times 15 = 105$ priors for damage conditions to allow a physical 6-legged robot to adapt. As the simulation is not perfect, the learning algorithm has to compensate for both the "reality gap" and the damage. The robot is tracked with an external motion capture system (Optitrack) and we use 10 episodes of 10 seconds. Like before, we consider two situations: when the damage is among the priors, and when it is not. We replicate each experiment 5 times.

Like in simulation, MLEI takes advantage of the priors to find higher-performing gaits than when a single prior is used (Fig. 6.8). When one of the priors correspond to the actual damage condition (Fig. 6.8(a)), MLEI clearly outperforms EI with a single prior and finds high-performing gaits in less than 10 episodes; MLEI also finds better gaits than EI when the actual damage condition is not among the priors (Fig. 6.8(b)), which is likely to come from the fact that MLEI can "take inspiration" from other priors to compensate for the damage (like in the previous task, this corresponds to a form of transfer learning).

6.4.4 Conclusion and Discussion

Well-chosen priors can guide BO to find a high-performing solution (Lizotte et al., 2007; Cully et al., 2015) while not constraining the search to a few pre-designed solutions. However, learning algorithms are most useful when the robot or the environment are partially known, therefore it is often challenging to design a single prior that would help BO in all the possible situations. The Most Likely Expected Improvement (MLEI) allows us to relax this assumption by making BO capable of selecting the most useful prior and ignore all the others. It therefore makes it possible for BO to benefit from the faster convergence speed given by the priors, while not assuming much about the robot or the environment.

In this section, we demonstrated that our new acquisition function leads to a powerful adaptation algorithm in two systems, a planar manipulator and a 6-legged robot. In the latter case, the robot was capable of discovering compensatory behaviors in a dozen of trials when damaged — even with priors



(a) Results with removed rear leg **(b)** Results with shortened rear leg

Figure 6.8: Comparison of MLEI with the standard EI with a single prior coming from a simulated undamaged robot. This real experiment was carried out on the physical damaged 6-legged robot walking on flat ground after 10 iterations of BO and with 5 replicates of the experiment. Damage used: missing rear leg (damage present among the available priors), a shortened rear leg (damage not present among the available priors).

.

that correspond to the intact robot — and when it faced unknown stairs – even without any prior for the actual stairs. Overall, MLEI substantially increases the potential uses of priors in BO because we can often "guess" what could be useful to the robot, but we cannot be sure in advance if a given prior will be useful in the future.

Even the best classification system based on perception (which context is recognized by the robot?) (Plagemann et al., 2008) is prone to errors in real situations (*e.g.*, steps hidden by high grass). By contrast, the automatic selection of priors that we introduced here is based on the direct observation of the rewards: the robot discovers what works and what does not, it does not attempt to know *why* some behaviors work and some do not. This approach fits well the theory of "embodied cognition" (Brooks, 1991; Pfeifer and Bongard, 2006) which suggests that robots do not need an explicit representation of the world to act. A classic "sense-plan-act" architecture would assume the existence of an accurate model of the world to act; at the other end of the spectrum, most learning algorithms aim at assuming as little knowledge as possible about the robot or the environment. BO with automatic selection of prior can be an

effective middle ground in which prior knowledge or a perception system can guide a direct policy search that can, if needed, ignore all previous knowledge and still find an effective way to act.

Chapter 7

Discussion

Throughout the manuscript, we have introduced algorithms that actively try to *minimize the interaction time between the robot and the environment* and *provide adaptive capabilities to robots*. To do this we relied on three main concepts:

- *Learning surrogate models*: using the gathered data to learn models either of the expected return or the transition dynamics is a powerful way towards reducing the interaction time.
- *Using simulators to improve learning*: utilizing dynamics simulators to create repertoires or use them as priors proves to be an effective way to reduce the required interaction time to complete a task.
- *Exploiting structured knowledge to improve learning*: taking advantage of years of research in robotics, we can propose algorithms that insert learning in the right place and not "waste" time to learn, for example, the basic laws of nature.

In the following sections, we will discuss the limitations of each of these concepts and provide intuitions and examples of situations in which our approaches cannot be applied or will struggle to work. We will also discuss potential improvements to our methods that could alleviate their limitations. Finally, we will discuss about the interplay between planning, model-predictive control and policy search as well as express a few thoughts on the computation time that micro-data algorithms need.

7.1 Learning surrogate models

7.1.1 Dealing with complex robots

The results of our experiments show that learning surrogate models is a powerful method for reducing the interaction time required to solve a robot control task. If we have a model in our possession, we can use it to reason about possible next decisions or to learn a policy by interacting with the model and not the

real system; if handled properly (as we have seen in the previous chapters), this approach can lead to substantial reductions of the interaction time. Moreover, model-based approaches that learn the transition dynamics can generalize well to new tasks, since the dynamics model usually does not depend on the task. Nevertheless, model-based approaches do not scale very well with the dimensionality of the function being approximated (the state and action space when modeling the dynamics and the policy space when modeling the expected return): in the general case, the quantity of data to learn a good approximation of a surrogate model scales exponentially with the dimensionality of the learned function (this is the curse of dimensionality, see (Bellman, 1957)).

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 (Jonschkowski and Brock, 2015; Lesort et al., 2018, 2017; Lesort and Filliat, 2017). In their approach, Jonschkowski and Brock (2015) learn in an unsupervised way a compact state representation using prior knowledge; they define 4 *robotic priors* that are inspired from the laws of nature and the nature of the robots:

- **Temporal coherence Prior:** Two states close to each other in time are also close to each other in the state representation space.
- **Proportionality Prior:** Two identical actions should result in two proportional magnitude state variations.
- **Repeatability Prior:** Two identical actions applied at similar states should provide similar state variations, not only in magnitude but also in direction.
- **Causality Prior:** If two states on which the same action is applied give two different rewards, they should not be close to each other in the state representation space.

The last prior is the only one giving information about the task and helps discovering the underlying states which lead to good rewards. They then showcase that this compact state representation can be used in RL algorithms without hurting much the performance (Jonschkowski and Brock, 2015). Nevertheless, there is still a lot of work before we have algorithms that can provide generic state representations that can be used to control complex robots (Lesort et al., 2018), like humanoids.

7.1.2 Dealing with high-dimensional observations

The dimensionality of the sensory observations is also an important challenge for model-based micro-data reinforcement learning since it affects indirectly the size of the policy in approaches where the expected return is modeled, and directly the dimensionality of the model learned in model-based policy search

approaches. To our knowledge, no micro-data approach (including the ones we presented in this manuscript) can perform "end-to-end learning", that is, learning with a raw data stream like a camera. Deep RL has recently made possible to learn policies from raw pixel input (Mnih et al., 2015), largely thanks to the prior 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 (Mnih et al., 2015)), which is not compatible with most robotics experiments and applications.

Fortunately, it is conceivable to create forward models in image space with reasonable number of samples, 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 (Oh et al., 2015; Assael et al., 2015; Ha and Schmidhuber, 2018; Wahlström et al., 2015). In their approach, Wahlström et al. (2015) propose a deep neural network architecture composed of an auto-encoder and a predictor in feature space. More precisely, the image data is first transformed in a low-dimensional feature space and the predictor layer predicts in feature space what would the next state look like; lastly, the decoder layer takes the predicted feature vector and transforms it back to the image space. Their results show that their approach can learn several different dynamics models from pixels, like a planar single or double pendulum.

Overall, it is still unclear how to learn with high-dimensional observation spaces in a handful of trials. Transferring the knowledge (Taylor and Stone, 2009) from a simulator to the real world could be one direction towards tackling this challenge: we can use the simulator to perform a lot of trials and then transfer what we have learned on a few trials on the real robot. Using the simulator as a means to learn low-dimensional features can also be a promising direction: the idea would be to first learn in an unsupervised fashion the low-dimensional features (Lesort et al., 2018) and then use these low dimensional features on the real system to achieve faster learning.

7.1.3 Dealing with sparse reward scenarios

Our experiments in chapters 4 and 5 showcase that in model-based policy search using the variance in the optimization is one of the key components to learn with as little interaction time as possible. However, the learned dynamics models are only confident in areas of the state space previously visited and thus could drive the optimization into local optima when multiple and diverse solutions exist (Deisenroth et al., 2015; Chatzilygeroudis et al., 2017). None of our approaches is able to systematically perform well in sparse reward scenarios. In other words, the robot, in order to improve its reward, needs to not only exploit good actions or states that have already been visited, but also try different actions that may lead to novel and possibly more effective behaviors. This is, of course, a specific instantiation of the general *exploration-exploitation dilemma* that arises in many fields as well as in RL (Sutton and Barto, 1998).

When there is no reward available, the robot should be "curious" and search for "interesting stepping stones". This question is central in developmental robotics (Oudeyer et al., 2007; Gottlieb et al., 2013), and evolutionary robotics (Lehman and Stanley, 2011; Mouret and Clune, 2015; Mouret, 2011; Doncieux and Mouret, 2014). Intrinsic motivation (Forestier et al., 2017) or curiosity-driven (Laversanne-Finot et al., 2018; Oudeyer, 2018) techniques as based on the fact that humans and animals are performing activities for their inherent satisfaction rather than for some separable consequence (Ryan and Deci, 2000). A few noteworthy successes of this type of algorithms were produced within the "Playground Experiment" (Oudeyer et al., 2005) and "Ergo-Robots" project (Oudeyer, 2011). In particular, in the "Ergo-Robots" project, some robotic manipulators (equipped with heads) explore their sensorimotor world, that is their body and what they can do with it, through "artificial curiosity". In essence, they execute some actions, observe the outcome in the environment (*e.g.*, an object was moved), and they build two models: one modeling the relations between actions and outcomes, and one that tells them how good they are at predicting. Finally, they are using these models to select actions that will produce novel or "surprising" outcomes. Interestingly, this unsupervised learning procedure gave rise to elementary communication skills between the robots themselves and between robots and humans (by making gestures and sounds).

Following similar ideas, in chapter 4, we saw that one way of addressing the exploration-exploitation dilemma in robot control is to combine model-based policy search approaches with novelty-based ideas (Lehman and Stanley, 2008, 2011). More specifically, we introduced a new algorithm, Multi-DEX, that frames the policy search problem as a multi-objective, model-based policy optimization problem with three objectives: (1) generate maximally novel state trajectories, (2) maximize the cumulative reward and (3) keep the system in state-space regions for which the model is as accurate as possible. It then optimizes these objectives using a Pareto-based multi-objective optimization algorithm. Multi-DEX was able to solve sparse reward scenarios (with a simulated robotic arm) in much lower interaction time than state-of-the-art model-free (*i.e.*, GEP-PG (Colas et al., 2018), TRPO (Schulman et al., 2015)), and model-based approaches (*i.e.*, Black-DROPS (Chatzilygeroudis et al., 2017), TRPO-VIME (Houthooft et al., 2016)). Nevertheless, Multi-DEX is not able to handle noisy systems and will struggle in high dimensional systems. Designing algorithms that are able to consistently find high performing policies in sparse reward scenarios is still an open problem.

7.1.4 Safe learning

Exploring too much on a physical robotic system might be dangerous or damage the robot. In Section 6.2, we combined Bayesian optimization with prior knowledge about safety criteria in order to reduce the probability of breaking the robot when searching for a policy. We were able to learn on a damaged simulated iCub humanoid robot that had to crawl again while

minimizing the unsafe trials. We still, however, relied on expert knowledge about what would be crucial for damaging the robot and performed several unsafe trials. While recent methods, like SafeOPT (Berkenkamp et al., 2016), safely learn without any unsafe trial, they require an initial safe set of policy parameters, which is hard to estimate in every possible scenario and still requires expert knowledge. Combining generic prior knowledge and safety criteria remains an open question and safe learning is of crucial importance in robotics applications.

7.2 Using simulators to improve learning

7.2.1 Behavioral or action repertoires

Simulation has been a very useful tool for many scientific fields, most notably physics, mechanical engineering, robotics and evolutionary robotics (Mouret and Chatzilygeroudis, 2017), as it allows, for example, researchers to extensively test their ideas "cost-free", or to exploit big clusters to speed-up their research experiments. In chapter 3, we saw how we can use simulation in order to create repertoires of actions and speed-up adaptation on the physical robot. In our case, we create a repertoire of controllers with the goal of covering the task space as well as possible, but also providing a mapping from the task space to the controller space. Moreover, using this repertoire to search for a controller also reduces the dimensionality of the search space since the task space is usually much lower dimensional than the controller space.

Although our goal was to create a repertoire that covers the task space and provides a mapping from the task space to the controller space, Cully et al. (2015) proposed using the same ideas to create a repertoire of controllers that maps a behavior descriptor to the task performance (they call them behavior-performance maps); more specifically, they were experimenting with a hexapod locomotion task and they defined the behavioral descriptor as the proportion of time that each leg touches the ground, and in this case, the resulting repertoires contained behaviors that walk in a straight line. Even though approaches based on repertoires can be robust to the choice of the description of the task space (referred also as *behavior descriptor*, see (Cully et al., 2015), Chapter 3 and Sec. 6.2), selecting the most appropriate behavior descriptor (and the appropriate performance function) remains an open question or requires knowledge from an expert.

One potential approach to handle this challenge could be the recently introduced hierarchical behavioral repertoires (HBR) that stacks several behavioral repertoires with the goal to generate sophisticated behaviors (Cully and Demiris, 2018). The main idea of the approach is to create several layers of behavioral repertoires each one selecting a behavior of the repertoire in the lower level; the repertoire in the last level selects either motor commands or low-level policies. In their paper (Cully and Demiris, 2018), a NAO robot was able to learn how to draw lines, arcs and digits by combining a 4-layer HBR with the concept of

innovation engines (Nguyen et al., 2015b, 2016) (*i.e.*, unsupervised learning of behavioral descriptors). The authors also showcase that by changing only the repertoire of the last layer, we can transfer behaviors from a simple robotic arm to NAO.

7.2.2 Learning robust policies

In chapter 6, we saw how dynamics simulators can help towards discovering robust and safe policies in low interaction time. In traditional trajectory optimization (or planning) in robotics (Siciliano and Khatib, 2016), the trajectory is given by an expert and a controller is designed to follow it in a robust way. In learning algorithms, the goal is to discover the trajectory and the robust controller simultaneously. A potential remedy is to use policies that are intrinsically robust to some perturbations, that is, designing the policy space such that a small change in the parameter space keeps the policy robust. This is one of the ideas behind dynamic movement primitives (see Sec. 2.5), which act like "attractors" towards a trajectory of a fixed point. Similarly, it is possible to learn waypoints (Lober et al., 2016) or "repulsors" (Spitz et al., 2017) to mix learning with advanced, closed-loop "whole-body" controllers. Lastly, one can learn distinct *soft* policies for simpler tasks and then compose them in order to achieve a more complicated task (Haarnoja et al., 2018).

7.3 Exploiting structured knowledge to improve learning

7.3.1 Nature versus nurture

Evolution has endowed animals and humans with substantial prior knowledge. For instance, hatchling turtles are prewired to run towards the sea (Musick and Limpus, 1997); or marine iguanas are able to run and jump within moments of their birth in order to avoid being eaten by snakes[1]. 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 2.5, 2.6.2, 2.7.2 and Chapters 3 and 5. In other words, priors play a catalytic role in reducing the interaction time of reinforcement learning methods. Thus, the following questions naturally arise: 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 learning, 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)

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

could relax these assumptions while still providing a learning speedup. Physical simulations can also be used to automatically generate priors while being a very generic tool (Cully et al., 2015; Pautrat et al., 2018; Antonova et al., 2017, 2016). In 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.

In Chapter 5, we saw how to combine model identification with data-driven model learning to improve learning on physical systems. We were able to learn on a physical hexapod in less than one minute of interaction time by exploiting specific properties of Gaussian processes and dynamics simulators. In short, we effectively combined (a) dynamics simulators that are the product of many years of research in dynamical systems, (b) prior knowledge of possible damages/different conditions and diagnosis (*i.e.*, identifying the likeliest robot model from data), and (c) Gaussian processes that can generalize well and improve with experience. Similarly, in concept, Jonschkowski et al. (2018) recently proposed to combine the well-known and studied particle filters with deep learning in order to improve the learning speed of localization. The main idea is that we can substitute the measurement and prediction models of the particle filter algorithm with deep neural networks that can be improved with more data. In essence, they combine the structure of *Bayes filtering* with the generalization capabilities of neural networks; in other words, they combine structured knowledge that comes from years and years of research in robotics and state estimation with data-driven algorithms that can improve with experience.

7.3.2 Automatic design of priors

Meta-learning (Feurer et al., 2015; Finn et al., 2017; Clavera et al., 2019; Sæmundsson et al., 2018), that is, "learning to learn", is a related line of work that can provide a principled and potentially automatic way of designing priors. Clavera et al. (2019), for example, use meta-learning to train a global neural network dynamics model such that, when combined with recent data, the model can be rapidly adapted to the local context. Their approach was, for example, able to efficiently learn to control a crippled in an unexpected way ant agent (quadruped) or a half-cheetah with disabled joints. Nevertheless, meta-learning approaches based on neural networks are still data-hungry: for example, the above results were obtained after 1.5-3 hours of interaction time. One way to mitigate this is to use Gaussian processes. For example, Sæmundsson et al. (2018) use GPs and they frame meta-learning as a hierarchical latent variable model and infer the relationship between tasks automatically from data. They were able to generalize to several instances of the cart-pole swing-up and the double pendulum swing-up tasks while using few examples in the meta-learning step. However, Gaussian processes have cubic computational complexity and thus practical approaches involve mainly simple or low dimensional systems.

7.3.3 Misleading priors

While priors can bootstrap learning, 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 to the current task (Chatzilygeroudis and Mouret, 2018). In Chapter 5, we saw how we can use parameterized black-box priors, and more specifically simulators, in order to properly select the parameterization that is the most likely, given the experience that the robot has so far. Nevertheless, this procedure can make the simulator to crash (*i.e.*, states getting out of bounds or objects in-collision) because there is no guarantee that the predicted state, which possibly makes sense in the real world, will make sense in the prior model; this is especially the case when the two models (prior and real) differ a lot and when there are obstacles and collisions involved. Recently, Ajay et al. (2018) proposed to combine a simulator with variational recurrent neural networks and were able to learn model for systems that involve collisions without feeding the model predictions back to the simulator; in particular a planar bouncing ball and a planar pushing task. Nevertheless, the systems they used were relatively low dimensional and involved only planar motions. Using the prior simulator just as a reference and not mixing prior and real data for complex robots remains an open problem.

Following the same line of thought, in Section 6.4 we proposed an acquisition function for Bayesian optimization that actively tries to select the most promising prior among a variety of them. Using our approach, a hexapod robot was able to learn in a few trials: (a) to walk again despite being damaged in an unknown way, and (b) to walk on unknown terrains (*e.g.*, new kinds of stairs). Nevertheless, choosing the available priors still required an expert (*i.e.*, we ad-hocly chose the types of prior information) and we can only select from a discrete set of priors. We believe that designing algorithms that take as prior a parameterized dynamics simulator and effectively exploit it in order to speed-up learning or transferring knowledge will produce some interesting results in the future.

7.3.4 Domain randomization and adaptation

Domain randomization (James et al., 2017) techniques (previously referred as "envelope-of-noise" approaches (Jakobi, 1997)) aim at answering to how we can effectively exploit simulation for speeding-up learning. In particular, they use a parameterized simulator and try to find policies that are robust to a vast number of conditions. Their ultimate goal is to discover control policies that will also work in the real world without any online refinement. James et al. (2017) use a rather simple controller, sample different goal targets and environmental conditions (*e.g.*, lighting, textures, position of the camera, etc.) and collect 1 million state-action trajectories of completing different goals. Once this big dataset is collected, a convolutional neural network (CNN), that will later serve as the policy, is trained in a supervised manner to find a

mapping between image observations and the appropriate (or optimal) actions to take. Finally, they deploy this policy both in new simulation environments and 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). However, their approach does 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.

Similar approaches randomize dynamic parameters of the system (*e.g.*, masses of the links, friction coefficients, etc.) instead of environmental conditions (Peng et al., 2017; Yu et al., 2017; Jakobi, 1997) in order to find a "robust" policy that is likely to work on the real system.

We can draw a parallel here and argue that model-based policy search with probabilistic models is performing something similar to dynamics randomization (*i.e.*, domain randomization over dynamics). If we think about it a bit more, 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. In particular, the policy returned by a policy search procedure under uncertain dynamics is not performing well with only some specific dynamics parameters, but with a set of them.

To go even further, the modeling procedure that we introduced in Chapter 5, GP-MI, should be superior to a pure dynamics randomization algorithm, since it both actively searches for dynamics parameters that are more likely to be true and correct whatever cannot be captured. Moreover, the probabilistic nature of GPs additionally provides a "robust" policy search procedure, as the policy found is not likely to overfit some specific dynamics parameters or model predictions. The computation complexity of the GPs added to the simulator's delays (remember that we are calling the simulator at each GP query) makes our model learning procedure practical only up to a few hundreds of time steps. Discovering a model learning procedure that can scale to high dimensional systems in a few minutes of interaction time is an exciting research avenue.

7.4 Interplay between model-predictive control, planning 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 2.7.1 and Chapters 4 and 5). 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) (Kalman, 1960), 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 (Mayne, 1966; Jacobson and Mayne, 1970) and its simpler variant, the iterative linear-quadratic Gaussian algorithm (Todorov and Li, 2005). 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.

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) (Garcia et al., 1989). 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 (Koenemann et al., 2015).

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 when not enough data have been collected), therefore, there needs to be a mechanism that corrects the mismatch. A potential solution could be to combine iterative learning control (Moore et al., 1992; Bristow et al., 2006) with MPC (*e.g.*, see (Lee et al., 1999, 2000; Wang et al., 2008; Assael et al., 2015)). MPC additionally has the disadvantage of requiring full knowledge of the system state. A way to mitigate this problem is to combine MPC with a policy search algorithm, such as guided policy search. This can be realized by using MPC with full state information in some training phase, to learn neural network policies that take the raw observations as input, thus, not requiring full state information at test time (Zhang et al., 2016). The execution of the policy can be parallelized and can thus run faster than MPC online.

Should we then learn a big neural network 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 the policy space. Even if we do not consider the memory requirements for such policies, 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 (Karaman and Frazzoli, 2011; Browne et al., 2012) 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 (Duarte et al., 2017)) or a planning algorithm (*e.g.*, see (Clever et al., 2017; Chatzilygeroudis et al., 2018a)).

7.5 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 2.7.1) is a very successful and data-efficient algorithm, but can be very computationally expensive when the state-action or policy space dimensionality increases (Chatzilygeroudis et al., 2017; Wilson et al., 2014) (*e.g.*, Wilson et al. (2014) 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 2.7.2 and Chapters 4 and 5) 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; see (Chatzilygeroudis and Mouret, 2018) and Chapter 5). Both approaches use GP models which have a complexity that is quadratic to the number of samples when queried; this is clearly inefficient when millions of such GP queries (*e.g.*, Black-DROPS performs around 64M (Chatzilygeroudis et al., 2017)) are performed in each episode.

As we have already discussed, there exist a few model-based RL methods that utilize neural networks that scale better than GPs with the number of samples (Chua et al., 2018; Higuera et al., 2018; Depeweg et al., 2017, 2018). Nevertheless, it is still not clear how to insert priors in neural networks like we do with GPs. Recently, Hafner et al. (2018) showed that deep neural networks with noise contrastive priors can provide reliable uncertainty information even for out-of-distribution query points. This seems a promising direction of research towards having neural networks with priors that could allow for more interesting micro-data RL algorithms.

On the other hand, IT&E (Cully et al., 2015) and "robust policies" (*e.g.*, see (Peng et al., 2017; Yu et al., 2017; Paul et al., 2018; Rajeswaran et al., 2017)) can practically run in real-time because the prior is pre-computed offline. This "recipe" is shared by recent meta-learning methodologies, such as (Finn et al., 2017), that aim to learn an expressive policy (*i.e.*, a deep neural network) 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 (Peng et al., 2017) use a form of directed exploration to create such a prior[2]. If, for example, random

[2]IT&E uses an evolutionary illumination algorithm (MAP-Elites (Mouret and Clune, 2015; Vassiliades et al., 2017)) to discover solutions to thousands of problems in a single run; the work by (Peng et al., 2017) uses a related approach (hindsight experience replay (Andrychowicz et al., 2017)).

search was used, it would probably need orders of magnitude more computation time to create a prior of the same quality.

Chapter 8

Conclusion

In this thesis, we first defined the challenge of learning by trial-and-error in a few minutes as "micro-data reinforcement learning" (Mouret, 2016). This challenge is closely related to the concept of "data-efficient reinforcement learning", but refers to slightly different cases or scenarios. The main difference is that the terminology "micro-data reinforcement learning" represents an absolute value of interaction time unlike the term efficiency that represents a relative one. For example, a micro-data algorithm might reduce the interaction time 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. We then proposed and evaluated several techniques for tackling this challenge by leveraging prior knowledge or building surrogate models. The goal of the proposed approaches was to *minimize the interaction time between the robot and the environment required to solve the task at hand* and made the case that this is the most appropriate metric to compare trial-and-error algorithms.

In our literature review (Chapter 2), we considered three main strategies for micro-data learning: (1) using prior information in the policy, (2) learning models of the expected return, and (3) learning models of the dynamics of the system. First, we made the observation that when designing the policy, the key design choices are what the space of the policy parameters is, and how it maps states to actions. This design is guided by a trade-off between having a representation that is expressive enough, and one that provides a parameter space that is efficiently searchable. As such, we can insert prior information either in the parameters or the structure of policy. In the first case, the most successful approaches are based on learning from demonstrations (Billard et al., 2008): starting with policy parameters that are close to the optimal ones greatly reduces the interaction time needed to find the optimal policy. In the second case, hand-designed policies (like Dynamical Movement Primitives, see Sec. 2.5.3) have provided impressive results within the robot learning community. Nevertheless, it is still not obvious how to automatically find a policy space that is both expressive and efficiently searchable and expert task-specific knowledge is required. Later in our review, we saw that Bayesian optimization is one of the

most promising approaches for micro-data reinforcement learning (Sec. 2.6.1). One of the most interesting features of BO is that it is straight-forward to insert prior information either in the policy space or in the value of the expected return (*e.g.*, by using a simulator). However, BO does not scale to big search spaces, which prevents it from being used with generic policies like neural networks. In the last part of our review, we examined how learning models of the dynamics and performing policy search on the learned models can greatly reduce the interaction time (Sec. 2.7.1). The methods that fall in this category have provided algorithms that require much fewer trials than algorithms that fit in either of the previous categories. Nevertheless, most of these approaches are based on Gaussian processes and thus practical implementations are only available for simple and low dimensional systems.

In this thesis, inspired by the application of reinforcement learning to robot damage recovery, we proposed algorithms that exploit prior information or model learning in order to substantially reduce the interaction time for learning robot controllers, especially with complex or high dimensional robots. With our approaches, robots can now learn and adapt to unforeseen situations in front of our eyes within a few seconds or minutes of interaction time.

We first proposed the usage of evolutionary algorithms for producing "creative priors" that can be effective when searching for a compensating behavior (Chapter 3). More precisely, we considered a robot damage recovery scenario, where a waypoint-controlled robot is damaged in an unknown way and, in order to get out of the building, the robot must recover its locomotion abilities so that it can reach the waypoints fixed by its operator. By combining probabilistic modeling (GPs) and approximate probabilistic planning (MCTS), we were able to propose the RTE algorithm that: (1) breaks the complexity by pre-generating hundreds of possible behaviors with a dynamics simulator of the intact robot, and (2) allows complex robots to quickly recover from damage while completing their tasks and taking the environment into account. Our experiments showcased that a damaged physical hexapod robot can recover most of its locomotion abilities in an environment with obstacles, and without any human intervention.

We then explored an alternative way of performing model-based policy search with probabilistic models (Chapter 4). We showed that we can combine the policy evaluation step with the optimization procedure in order to get a more flexible, faster and modern implementation of model-based policy search algorithms. In particular, we introduced Black-DROPS, a novel policy search algorithm that takes advantage of multi-core architectures and lifts several constraints (*e.g.*, specific reward type) while maintaining the data-efficiency of analytical algorithms. The main insight is that Monte-carlo approaches and population-based black-box optimizers like CMA-ES (Hansen and Ostermeier, 2001) (1) do not put any constraint on the reward functions and policies, and (2) are straightforward to parallelize, which can make them competitive with analytical approaches when several cores are available. Black-DROPS was able to learn to solve the cartpole swing-up task and to control a physical 4-DOF manipulator in just 20 seconds of interaction time.

We additionally explored an extension to Black-DROPS with the goal of improving its computation time and quality of results in difficult scenarios (like very noisy systems). In particular, we used the empirical bootstrap method (Efron and Tibshirani, 1994) and the idea of racing (Heidrich-Meisner and Igel, 2009) aiming at allocating as efficiently as possible the available budget of function evaluations, that is, to spend more function evaluations where it is actually needed and not blindly, like pure Monte-carlo methods. Our preliminary results showcased that we can get similar (or better) quality of results with vanilla Black-DROPS while improving the computation time by avoiding needless evaluations. However, there is still space for improvement and more investigation is needed since we only tested this extension on a simplified system.

The vanilla Black-DROPS algorithm is essentially greedy and will exploit the most promising states and/or actions; this might lead to sub-optimal behaviors or even completely prevent convergence. This is especially the case when the reward is sparse, that is, the reward is zero for the most part of the state space. To handle these cases, we proposed Multi-DEX, a model-based policy search approach that takes inspiration from novelty-based ideas. In particular, apart from the cumulative reward, we also maximize the novelty of state trajectories in order to explore new and interesting policies. We demonstrated the performance of Multi-DEX in several sparse rewards scenarios and showcased that it outperforms several state-of-the-art approaches.

While Black-DROPS scales well with the number of processors, the main challenge of model-based policy search is scaling up to complex robots: as the algorithm models the transition function between full state/action spaces (joint positions, environment, joint velocities, *etc.*), the complexity of the model increases substantially with each new degree of freedom; unfortunately, the quantity of data required to learn a good model scales most of the time exponentially with the dimension of the state space (Keogh and Mueen, 2011). To cope with this problem, we introduced a new model learning procedure, called GP-MI, that combines model identification, black-box parameterized priors and Gaussian process regression (Chapter 5). Our results revealed that by combining Black-DROPS with GP-MI, we can learn to control a physical hexapod robot (48D state space, 18D action space) in less than one minute of interaction time.

In the last part of the thesis (Chapter 6), we discussed how safety constraints, robustness and multiple priors can be incorporated in a Bayesian optimization procedure in order to solve the micro-data reinforcement learning challenge. More precisely, we first introduced sIT&E (Safety-aware Intelligent Trial & Error Algorithm) an extension of the Intelligent Trial & Error algorithm that includes safety criteria in the learning process. With sIT&E we were able to safely learn crawling behaviors for a simulated damaged iCub humanoid robot (Nori et al., 2015; Tsagarakis et al., 2007) in less than 20 trials. We, also, proposed ALOQ (ALternating Optimization and Quadrature) and TALOQ (Transferable ALOQ) with the goal of finding policies that are robust even in significant rare events. Using TALOQ we were able to learn policies for a damaged physical arm that

had to hit a ball and produce a specific velocity. Lastly, we introduced MLEI (Most Likely Expected Improvement), a new acquisition function for Bayesian optimization, that effectively combines multiple sources of prior information in order to minimize the interaction time. Our results showed that using the MLEI acquisition function we were able to learn effective policies for a physical hexapod robot that had to climb different types of stairs and adapt to unforeseen damages.

With our proposed algorithms, robots can now learn and find adaptive behaviors in a handful of trials in order to complete their tasks despite possible unforeseen situations. All of our methods relied on either leveraging prior knowledge or building surrogate models; in many cases, these two strategies were combined. Interestingly, the Black-DROPS algorithm combines somehow the reinforcement learning community with the optimization and evolutionary robotics literature and we believe that it will give rise to many interesting hybrid algorithms. One such example is our Multi-DEX algorithm that merges the big novelty-based literature from evolutionary robotics (Lehman and Stanley, 2008, 2011) with model-based RL (Deisenroth et al., 2013) to produce a sample-efficient RL algorithm that is able to outperform many state-of-the-art pure RL approaches. We can imagine many more variants just by changing the type of models (*e.g.*, probabilistic neural networks), reward functions (*e.g.*, coming from an inverse reinforcement learning procedure) and optimizers (*e.g.*, quality-diversity algorithms) inspired by other fields.

Bibliography

P. Abbeel, M. Quigley, and A. Y. Ng. Using inaccurate models in reinforcement learning. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2006.

A. Ajay, J. Wu, N. Fazeli, M. Bauza, L. P. Kaelbling, J. B. Tenenbaum, and A. Rodriguez. Augmenting Physical Simulators with Stochastic Neural Networks: Case Study of Planar Pushing and Bouncing. In *Proceedings of the International Conference on Intelligent Robots (IROS)*, 2018.

B. D. Anderson and J. B. Moore. Optimal filtering. *Englewood Cliffs*, 21:22–95, 1979.

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 *Advances in Neural Information Processing Systems (NIPS)*, 2017.

R. Antonova, A. Rai, and C. G. Atkeson. Sample efficient optimization for learning controllers for bipedal locomotion. In *Proceedings of the International Conference on Humanoid Robots (Humanoids)*. IEEE, 2016.

R. Antonova, A. Rai, and C. G. Atkeson. Deep Kernels for Optimizing Locomotion Controllers. In *Proceedings of Conference on Robot Learning (CoRL)*, 2017.

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 Reinforcement Learning Workshop*, 2015.

K. J. Åström. *Introduction to stochastic control theory*. Courier Corporation, 2012.

C. G. Atkeson, B. P. W. Babu, N. Banerjee, D. Berenson, C. P. Bove, X. Cui, M. DeDonato, R. Du, S. Feng, P. Franklin, et al. No falls, no resets: Reliable humanoid behavior in the DARPA robotics challenge. In *Proceedings of the International Conference on Humanoid Robots (Humanoids)*, 2015.

C. G. Atkeson, B. Babu, N. Banerjee, D. Berenson, C. Bove, X. Cui, M. DeDonato, R. Du, S. Feng, P. Franklin, et al. What happened at the DARPA robotics challenge, and why? *DRC Finals Special Issue of the Journal of Field Robotics*, 2016.

A. Auger and N. Hansen. A restart cma evolution strategy with increasing population size. In *Proceedings of IEEE Congress on Evolutionary Computation*, 2005.

T. Bartz-Beielstein, C. W. G. Lasarczyk, and M. Preuss. Sequential parameter optimization. In *Proceedings of IEEE Congress on Evolutionary Computation*, 2005.

R. E. Bellman. *Dynamic Programming*. Princeton University Press, Princeton, NJ, 1957.

F. Berkenkamp, A. P. Schoellig, and A. Krause. Safe Controller Optimization for Quadrotors with Gaussian Processes. In *Proceedings of the International Conference on Robotics and Automation (ICRA)*, 2016.

A. Billard, S. Calinon, R. Dillmann, and S. Schaal. Robot programming by demonstration. In *Springer handbook of robotics*, pages 1371–1394. Springer, 2008.

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 *Proceedings of the International Conference on Robotics and Automation (ICRA)*, 2014.

R. Bischoff, U. Huggenberger, and E. Prassler. Kuka youbot: a mobile manipulator for research and education. In *Proceedings of the International Conference on Robotics and Automation (ICRA)*, 2011.

J. Bongard, V. Zykov, and H. Lipson. Resilient machines through continuous self-modeling. *Science*, 314(5802):1118–1121, 2006.

J. C. Bongard and R. Pfeifer. Evolving complete agents using artificial ontogeny. In F. Hara and R. Pfeifer, editors, *Morpho-functional Machines: The New Species*, pages 237–258, Tokyo, 2003. Springer Japan. ISBN 978-4-431-67869-4.

F.-X. Briol, C. J. Oates, M. Girolami, M. A. Osborne, and D. Sejdinovic. Probabilistic Integration: A Role for Statisticians in Numerical Analysis? *arXiv*, 2015.

D. A. Bristow, M. Tharayil, and A. G. Alleyne. A survey of iterative learning control. *IEEE Control Systems*, 26(3):96–114, 2006.

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.

R. A. Brooks. Elephants don't play chess. *Robotics and autonomous systems*, 6(1-2):3–15, 1990.

R. A. Brooks. Intelligence without representation. *Artificial intelligence*, 47 (1-3):139–159, 1991.

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*, 4(1):1–43, 2012.

J. Buchli, F. Stulp, E. Theodorou, and S. Schaal. Learning variable impedance control. *International Journal of Robotics Research*, 30(7):820–833, 2011.

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

S. Calinon. A tutorial on task-parameterized movement learning and retrieval. *Intelligent Service Robotics*, 9(1):1–29, January 2016. ISSN 1861-2776. doi: 10.1007/s11370-015-0187-9.

S. Calinon, F. Guenter, and A. Billard. On Learning, Representing and Generalizing a Task in a Humanoid Robot. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 37(2):286–298, 2007.

S. Calinon, P. Kormushev, and D. G. Caldwell. Compliant skills acquisition and multi-optima policy search with EM-based reinforcement learning. *Robotics and Autonomous Systems*, 61(4):369–379, April 2013.

S. Calinon, D. Bruno, and D. G. Caldwell. A task-parameterized probabilistic model with minimal intervention control. In *Proceedings of the International Conference on Robotics and Automation (ICRA)*, 2014.

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

R. Camoriano, S. Traversaro, L. Rosasco, G. Metta, and F. Nori. Incremental semiparametric inverse dynamics learning. In *Proceedings of the International Conference on Robotics and Automation (ICRA)*, 2016.

J. Carlson and R. R. Murphy. How UGVs physically fail in the field. *IEEE Transactions on Robotics*, 21(3):423–437, 2005.

T. Cazenave and N. Jouandeau. On the parallelization of UCT. In *Proceedings of the Computer Games Workshop*, 2007.

G. Chaslot, S. Bakkes, I. Szita, and P. Spronck. Monte-carlo tree search: A new framework for game AI. In *Proceedings of Artificial Intelligence and Interactive Digital Entertainment (AIIDE)*, 2008.

K. Chatzilygeroudis and J.-B. Mouret. Using Parameterized Black-Box Priors to Scale Up Model-Based Policy Search for Robotics. In *Proceedings of the International Conference on Robotics and Automation (ICRA)*, 2018.

K. Chatzilygeroudis, A. Cully, and J.-B. Mouret. Towards semi-episodic learning for robot damage recovery. In *AILTA '16: Proceedings of the International Workshop "AI for Long-term Autonomy" at ICRA*, 2016.

K. Chatzilygeroudis, R. Rama, R. Kaushik, D. Goepp, V. Vassiliades, and J.-B. Mouret. Black-Box Data-efficient Policy Search for Robotics. In *Proceedings of the International Conference on Intelligent Robots and Systems (IROS)*, 2017.

K. Chatzilygeroudis, V. Vassiliades, and J.-B. Mouret. Reset-free trial-and-error learning for robot damage recovery. *Robotics and Autonomous Systems*, 100: 236–250, 2018a.

K. Chatzilygeroudis, V. Vassiliades, F. Stulp, S. Calinon, and J.-B. Mouret. A survey on policy search algorithms for learning robot controllers in a handful of trials. *arXiv preprint arXiv:1807.02303*, 2018b.

K. Chua, R. Calandra, R. McAllister, and S. Levine. Deep reinforcement learning in a handful of trials using probabilistic dynamics models. In *Advances in Neural Information Processing Systems (NIPS)*, 2018.

K. Ciosek and S. Whiteson. Expected policy gradients. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence (AAAI)*, 2018a.

K. Ciosek and S. Whiteson. Expected Policy Gradients for Reinforcement Learning. *arXiv preprint arXiv:1801.03326*, 2018b.

D. CireşAn, U. Meier, J. Masci, and J. Schmidhuber. Multi-column deep neural network for traffic sign classification. *Neural Networks*, 32:333–338, 2012.

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 *Proceedings of the International Conference on Learning Representations (ICLR)*, 2019.

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*, 2(2):977–984, 2017.

C. Colas, O. Sigaud, and P.-Y. Oudeyer. GEP-PG: Decoupling Exploration and Exploitation in Deep Reinforcement Learning Algorithms. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2018.

F. Corbato. On Building Systems That Will Fail. *ACM Turing award lectures*, 34(9):72–81, 2007.

A. Couëtoux, J.-B. Hoock, N. Sokolovska, O. Teytaud, and N. Bonnard. Continuous upper confidence trees. In *Proceedings of Learning and Intelligent Optimization (LION)*, 2011.

A. Couetoux, M. Milone, M. Brendel, H. Doghmen, M. Sebag, and O. Teytaud. Continuous rapid action value estimates. In *Proceedings of Asian Conference on Machine Learning*, 2011.

A. Cully and Y. Demiris. Quality and Diversity Optimization: A Unifying Modular Framework. *IEEE Transactions on Evolutionary Computation*, 2017.

A. Cully and Y. Demiris. Hierarchical behavioral repertoires with unsupervised descriptors. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, 2018. ISBN 978-1-4503-5618-3.

A. Cully, J. Clune, D. Tarapore, and J.-B. Mouret. Robots that can adapt like animals. *Nature*, 521(7553):503–507, 2015.

A. Cully, K. Chatzilygeroudis, F. Allocati, and J.-B. Mouret. Limbo: A Flexible High-performance Library for Gaussian Processes modeling and Data-Efficient Optimization. *The Journal of Open Source Software*, 3(26): 545, 2018. doi: 10.21105/joss.00545.

M. Cutler and J. P. How. Efficient reinforcement learning for robots using informative simulated priors. In *Proceedings of the International Conference on Robotics and Automation (ICRA)*, 2015.

C. Daniel, G. Neumann, O. Kroemer, and J. Peters. Hierarchical relative entropy policy search. *Journal of Machine Learning Research (JMLR)*, pages 1–50, 2016.

DARPA. DARPA's ATLAS Robot Unveiled, 2013. URL https://www.darpa.mil/news-events/2013-07-11.

K. Deb. Multi-objective optimization. In *Search methodologies*, pages 403–449. Springer, 2014.

K. Deb and D. Kalyanmoy. *Multi-Objective Optimization Using Evolutionary Algorithms*. John Wiley & Sons, Inc., New York, NY, USA, 2001.

K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast and elitist multi-objective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197, 2002.

M. DeDonato, F. Polido, K. Knoedler, B. P. Babu, N. Banerjee, C. P. Bove, X. Cui, R. Du, P. Franklin, J. P. Graff, et al. Team WPI-CMU: Achieving Reliable Humanoid Behavior in the DARPA Robotics Challenge. *Journal of Field Robotics*, 34(2):381–399, 2017.

T. Degris, M. White, and R. S. Sutton. Linear off-policy actor-critic. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2012.

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

M. P. Deisenroth and C. E. Rasmussen. PILCO: A model-based and data-efficient approach to policy search. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2011.

M. P. Deisenroth, C. E. Rasmussen, and D. Fox. Learning to control a low-cost manipulator using data-efficient reinforcement learning. In *Proceedings of Robotics: Science & Systems (RSS)*, 2011.

M. P. Deisenroth, R. Calandra, A. Seyfarth, and J. Peters. Toward fast policy search for learning legged locomotion. In *Proceedings of the International Conference on Intelligent Robots and Systems (IROS)*, 2012.

M. P. Deisenroth, G. Neumann, and J. Peters. A Survey on Policy Search for Robotics. *Foundations and Trends in Robotics*, 2(1):1–142, 2013.

M. P. Deisenroth, P. Englert, J. Peters, and D. Fox. Multi-task policy search for robotics. In *Proceedings of the International Conference on Robotics and Automation (ICRA)*, 2014.

M. P. Deisenroth, D. Fox, and C. E. Rasmussen. Gaussian processes for data-efficient learning in robotics and control. *IEEE Transactions Pattern Analysis and Machine Intelligence*, 37(2):408–423, 2015.

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 *Proceedings of the International Conference on Learning Representations (ICLR)*, 2017.

S. Depeweg, J. M. Hernández-Lobato, F. Doshi-Velez, and S. Udluft. Decomposition of uncertainty in bayesian deep learning for efficient and risk-sensitive learning. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2018.

A. Doerr, C. Daniel, D. Nguyen-Tuong, A. Marco, S. Schaal, T. Marc, and S. Trimpe. Optimizing long-term predictions for model-based policy search. In *Proceedings of Conference on Robot Learning (CoRL)*, 2017.

S. Doncieux and J.-B. Mouret. Beyond black-box optimization: a review of selective pressures for evolutionary robotics. *Evolutionary Intelligence*, 7(2): 71–93, 2014.

A. Droniou, S. Ivaldi, P. Stalph, M. Butz, and O. Sigaud. Learning velocity kinematics: Experimental comparison of on-line regression algorithms. In *Robotica*, pages 15–20, 2012.

M. Duarte, J. Gomes, S. M. Oliveira, and A. L. Christensen. EvoRBC: evolutionary repertoire-based control for robots with arbitrary locomotion complexity. In *Proceedings of The Genetic and Evolutionary Computation Conference (GECCO)*, 2016.

M. Duarte, J. Gomes, S. M. Oliveira, and A. L. Christensen. Evolution of repertoire-based control for robots with complex locomotor systems. *IEEE Transactions on Evolutionary Computation*, 2017.

H. Durrant-Whyte and T. Bailey. Simultaneous localization and mapping: part I. *IEEE Robotics & Automation Magazine*, 13(2):99–110, 2006.

B. Efron and R. J. Tibshirani. *An introduction to the bootstrap*. CRC press, 1994.

Y. Engel, S. Mannor, and R. Meir. Reinforcement learning with Gaussian processes. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2005.

H. Eskandari and C. D. Geiger. Evolutionary multiobjective optimization in noisy problem environments. *Journal of Heuristics*, 15(6):559, 2009.

C. Fernando, D. Banarse, C. Blundell, Y. Zwols, D. Ha, A. A. Rusu, A. Pritzel, and D. Wierstra. Pathnet: Evolution channels gradient descent in super neural networks. *arXiv preprint arXiv:1701.08734*, 2017.

M. Feurer, J. T. Springenberg, and F. Hutter. Initializing bayesian hyperparameter optimization via meta-learning. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, 2015.

P. Fidelman and P. Stone. Learning ball acquisition on a physical robot. In *Proceedings of the International Symposium on Robotics and Automation (ISRA)*, 2004.

C. Finn, P. Abbeel, and S. Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2017.

S. Forestier, Y. Mollard, and P.-Y. Oudeyer. Intrinsically motivated goal exploration processes with automatic curriculum learning. *arXiv preprint arXiv:1708.02190*, 2017.

A. Gaier, A. Asteroth, and J.-B. Mouret. Feature space modeling through surrogate illumination. In *Proceedings of The Genetic and Evolutionary Computation Conference (GECCO)*, 2017.

Y. Gal and Z. Ghahramani. Dropout as a Bayesian approximation: Representing model uncertainty in deep learning. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2016.

Y. Gal, R. T. McAllister, and C. E. Rasmussen. Improving PILCO with Bayesian neural network dynamics models. In *Data-Efficient Machine Learning Workshop*, 2016.

C. Garcia and M. Delakis. Convolutional face finder: A neural architecture for fast and robust face detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(11):1408–1423, 2004.

C. E. Garcia, D. M. Prett, and M. Morari. Model predictive control: theory and practice—a survey. *Automatica*, 25(3):335–348, 1989.

J. R. Gardner, M. J. Kusner, Z. E. Xu, K. Q. Weinberger, and J. P. Cunningham. Bayesian Optimization with Inequality Constraints. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2014.

J. Gottlieb, P.-Y. Oudeyer, et al. Information-seeking, curiosity, and attention: computational and neural mechanisms. *Trends in Cognitive Sciences*, 17(11): 585–593, 2013.

F. Guenter, M. Hersch, S. Calinon, and A. Billard. Reinforcement learning for imitating constrained reaching movements. *Advanced Robotics, Special Issue on Imitative Robots*, 21(13):1521–1544, 2007.

E. Guizzo. Fukushima robot operator writes tell-all blog. In *IEEE Spectrum*, 2011. URL: http://spectrum.ieee.org/automaton/robotics/industrial-robots/fukushima-robot-operator-diaries.

D. Ha and J. Schmidhuber. Recurrent World Models Facilitate Policy Evolution. In *Advances in Neural Information Processing Systems (NIPS)*, 2018.

T. Haarnoja, V. Pong, A. Zhou, M. Dalal, P. Abbeel, and S. Levine. Composable Deep Reinforcement Learning for Robotic Manipulation. In *Proceedings of the International Conference on Robotics and Automation (ICRA)*, 2018.

D. Hafner, D. Tran, A. Irpan, T. Lillicrap, and J. Davidson. Reliable Uncertainty Estimates in Deep Neural Networks using Noise Contrastive Priors. *arXiv preprint arXiv:1807.09289*, 2018.

N. Hansen. *The CMA Evolution Strategy: A Comparing Review*. Springer, 2006. ISBN 978-3-540-32494-2. doi: 10.1007/3-540-32494-1_4.

N. Hansen. Benchmarking a BI-population CMA-ES on the BBOB-2009 noisy testbed. In *Proceedings of The Genetic and Evolutionary Computation Conference (GECCO)*, 2009.

N. Hansen and A. Ostermeier. Completely derandomized self-adaptation in evolution strategies. *Evolutionary Computation*, 9(2):159–195, 2001.

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 Transactions on Evolutionary Computation*, 13(1):180–197, 2009.

H. v. Hasselt and M. A. Wiering. Reinforcement learning in continuous action spaces. 2007.

N. Heess, S. Sriram, J. Lemmon, J. Merel, G. Wayne, Y. Tassa, T. Erez, Z. Wang, A. Eslami, M. Riedmiller, et al. Emergence of locomotion behaviours in rich environments. *arXiv preprint arXiv:1707.02286*, 2017.

V. Heidrich-Meisner and C. Igel. Hoeffding and bernstein races for selecting policies in evolutionary direct policy search. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2009.

P. Hennig and C. J. Schuler. Entropy search for information-efficient global optimization. *Journal of Machine Learning Research*, 13(Jun):1809–1837, 2012.

T. Hester and P. Stone. TEXPLORE: real-time sample-efficient reinforcement learning for robots. *Machine Learning*, 90(3):385–429, 2013.

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.

R. Houthooft, X. Chen, Y. Duan, J. Schulman, F. De Turck, and P. Abbeel. VIME: Variational information maximizing exploration. In *Advances in Neural Information Processing Systems (NIPS)*, 2016.

I. Hupkens, A. Deutz, K. Yang, and M. Emmerich. Faster exact algorithms for computing expected hypervolume improvement. In *Proceedings of the International Conference on Evolutionary Multi-Criterion Optimization*, 2015.

F. Hutter, H. H. Hoos, K. Leyton-Brown, and K. P. Murphy. An Experimental Investigation of Model-based Parameter Optimisation: SPO and Beyond. In *Proceedings of The Genetic and Evolutionary Computation Conference (GECCO)*, 2009.

A. Ijspeert, J. Nakanishi, P. Pastor, H. Hoffmann, and S. Schaal. Dynamical movement primitives: Learning attractor models for motor behaviors. *Neural Computation*, 25(2):328–373, 2013.

A. J. Ijspeert, J. Nakanishi, and S. Schaal. Movement imitation with nonlinear dynamical systems in humanoid robots. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2002.

A. J. Ijspeert, J. Nakanishi, and S. Schaal. Learning attractor landscapes for learning motor primitives. In *Advances in Neural Information Processing Systems (NIPS)*, 2003.

V. T. Inman, H. D. Eberhart, et al. The major determinants in normal and pathological gait. *JBJS*, 35(3):543–558, 1953.

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

D. H. Jacobson and D. Q. Mayne. Differential dynamic programming. 1970.

N. Jakobi. Evolutionary robotics and the radical envelope-of-noise hypothesis. *Adaptive behavior*, 6(2):325–368, 1997.

N. Jakobi, P. Husbands, and I. Harvey. Noise and the reality gap: The use of simulation in evolutionary robotics. In *Proceedings of the European Conference on Artificial Life*, 1995.

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 *Proceedings of the Conference on Robot Learning (CoRL)*, 2017.

Y. Jin and J. Branke. Evolutionary optimization in uncertain environments-a survey. *IEEE Transactions on Evolutionary Computation*, 9(3):303–317, 2005.

G. Johnson Steven. The NLopt nonlinear-optimization package.

R. Jonschkowski and O. Brock. Learning state representations with robotic priors. *Autonomous Robots*, 39(3):407–428, 2015.

R. Jonschkowski, D. Rastogi, and O. Brock. Differentiable Particle Filters: End-to-End Learning with Algorithmic Priors. In *Proceedings of Robotics: Science and Systems (RSS)*, 2018.

S. J. Julier and J. K. Uhlmann. Unscented filtering and nonlinear estimation. *Proceedings of the IEEE*, 92(3):401–422, 2004.

L. P. Kaelbling, M. L. Littman, and A. W. Moore. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4:237–285, 1996.

S. M. Kakade. A natural policy gradient. In *Advances in Neural Information Processing Systems (NIPS)*, 2002.

M. Kalakrishnan, L. Righetti, P. Pastor, and S. Schaal. Learning force control policies for compliant manipulation. In *Proceedings of the International Conference on Intelligent Robots and Systems (IROS)*, 2011.

R. E. Kalman. A new approach to linear filtering and prediction problems. *Journal of basic Engineering*, 82(1):35–45, 1960.

M. Kanagawa, B. K. Sriperumbudur, and K. Fukumizu. Convergence guarantees for kernel-based quadrature rules in misspecified settings. In *Advances in Neural Information Processing Systems (NIPS)*, 2016.

K. Kandasamy, J. Schneider, and B. Póczos. High dimensional bayesian optimisation and bandits via additive models. In *International Conference on Machine Learning (ICML)*, 2015.

S. Karaman and E. Frazzoli. Sampling-based algorithms for optimal motion planning. *The International Journal of Robotics Research*, 30(7):846–894, 2011.

R. Kaushik, K. Chatzilygeroudis, and J.-B. Mouret. Multi-objective model-based policy search for data-efficient learning with sparse rewards. In *Proceedings of the Conference on Robot Learning (CoRL)*, 2018.

E. Keogh and A. Mueen. Curse of dimensionality. In *Encyclopedia of Machine Learning*, pages 257–258. Springer, 2011.

J. Kirkpatrick, R. Pascanu, N. Rabinowitz, J. Veness, G. Desjardins, A. A. Rusu, K. Milan, J. Quan, T. Ramalho, A. Grabska-Barwinska, et al. Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences*, 2017.

J. Ko, D. J. Klein, D. Fox, and D. Haehnel. Gaussian processes and reinforcement learning for identification and control of an autonomous blimp. In *Proceedings of the International Conference on Robotics and Automation (ICRA)*, 2007.

J. Kober and J. Peters. Learning motor primitives for robotics. In *Proceedings of the International Conference on Robotics and Automation (ICRA)*, 2009.

J. Kober, J. A. Bagnell, and J. Peters. Reinforcement learning in robotics: A survey. *International Journal of Robotics Research*, 32(11):1238–1274, 2013.

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 *Proceedings of the International Conference on Intelligent Robots and Systems (IROS)*, 2015.

N. Kohl and P. Stone. Policy gradient reinforcement learning for fast quadrupedal locomotion. In *Proceedings of the International Conference on Robotics and Automation (ICRA)*, 2004.

V. R. Konda and J. N. Tsitsiklis. Actor-critic algorithms. In *Advances in Neural Information Processing Systems (NIPS)*, 2000.

S. Koos and J.-B. Mouret. Online discovery of locomotion modes for wheel-legged hybrid robots: A transferability-based approach. In *Proceedings of the International Conference on Climbing and Walking Robots and Support Technologies for Mobile Machines*. 2012.

S. Koos, A. Cully, and J.-B. Mouret. Fast damage recovery in robotics with the t-resilience algorithm. *The International Journal of Robotics Research*, 32(14):1700–1723, 2013a.

S. Koos, J.-B. Mouret, and S. Doncieux. The transferability approach: Crossing the reality gap in evolutionary robotics. *IEEE Transactions on Evolutionary Computation*, 17(1):122–145, 2013b.

M. Krstic, I. Kanellakopoulos, P. V. Kokotovic, et al. *Nonlinear and adaptive control design*, volume 222. Wiley New York, 1995.

V. Kumar, E. Todorov, and S. Levine. Optimal control with learned local models: Application to dexterous manipulation. In *Proceedings of the International Conference on Robotics and Automation (ICRA)*, 2016.

A. Kupcsik, M. P. Deisenroth, J. Peters, A. P. Loh, P. Vadakkepat, and G. Neumann. Model-based contextual policy search for data-efficient generalization of robot skills. *Artificial Intelligence*, 247:415–439, 2017.

H. J. Kushner. A new method of locating the maximum point of an arbitrary multipeak curve in the presence of noise. *Journal of Basic Engineering*, 86 (1):97–106, 1964.

K. J. Kyriakopoulos and G. N. Saridis. Minimum jerk path generation. In *Proceedings of the International Conference on Robotics and Automation (ICRA)*, 1988.

A. Laversanne-Finot, A. Péré, and P.-Y. Oudeyer. Curiosity Driven Exploration of Learned Disentangled Goal Spaces. In *Proceedings of the Conference on Robot Learning (CoRL)*, 2018.

S. Lawrence, C. L. Giles, A. C. Tsoi, and A. D. Back. Face recognition: A convolutional neural-network approach. *IEEE Transactions on Neural Networks*, 8(1):98–113, 1997.

Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *Nature*, 521(7553): 436–444, 2015.

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.

J. Lee et al. DART: Dynamic Animation and Robotics Toolkit. *The Journal of Open Source Software*, 3(22), 2018. doi: 10.21105/joss.00500.

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*, 36 (5):641–657, 2000.

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*, 45(10):2175–2187, 1999.

S. Legg and M. Hutter. Universal intelligence: A definition of machine intelligence. *Minds and Machines*, 17(4):391–444, 2007.

J. Lehman and K. O. Stanley. Exploiting open-endedness to solve problems through the search for novelty. In *Proceedings of the Conference on Artificial Life (ALIFE)*, 2008.

J. Lehman and K. O. Stanley. Abandoning objectives: Evolution through the search for novelty alone. *Evolutionary Computation*, 19(2):189–223, 2011.

J. Lehman, S. Risi, and J. Clune. Creative generation of 3D objects with deep learning and innovation engines. In *Proceedings of the International Conference on Computational Creativity*, 2016.

J. Lehman, J. Chen, J. Clune, and K. O. Stanley. ES Is More Than Just a Traditional Finite-Difference Approximator. *arXiv preprint arXiv:1712.06568*, 2017.

S. Lengagne, J. Vaillant, E. Yoshida, and A. Kheddar. Generation of whole-body optimal dynamic multi-contact motions. *International Journal of Robotics Research*, 32(9-10):1104–1119, 2013.

T. Lesort and D. Filliat. Unsupervised deep learning of state representation using robotic priors. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2017.

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.

T. Lesort, N. Díaz-Rodríguez, J.-F. Goudou, and D. Filliat. State representation learning for control: An overview. *arXiv preprint arXiv:1802.04181*, 2018.

S. Levine and P. Abbeel. Learning neural network policies with guided policy search under unknown dynamics. In *Advances in Neural Information Processing Systems (NIPS)*, 2014.

S. Levine and V. Koltun. Guided policy search. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2013.

S. Levine, C. Finn, T. Darrell, and P. Abbeel. End-to-end training of deep visuomotor policies. *Journal of Machine Learning Research*, 17(39):1–40, 2016.

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 *Proceedings of the International Conference on Learning Representations (ICLR)*, 2016.

H. Liu, Y.-S. Ong, X. Shen, and J. Cai. When Gaussian Process Meets Big Data: A Review of Scalable GPs. *arXiv preprint arXiv:1807.01065*, 2018.

D. J. Lizotte, T. Wang, M. H. Bowling, and D. Schuurmans. Automatic gait optimization with gaussian process regression. In *Proceedings of the International Joint Conferences on Artificial Intelligence (IJCAI)*, 2007.

R. Lober, V. Padois, and O. Sigaud. Efficient reinforcement learning for humanoid whole-body control. In *Proceedings of the International Conference on Humanoid Robots (Humanoids)*, 2016.

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 *Proceedings of the International Conference on Robotics and Automation (ICRA)*, 2017.

K. M. Lynch and F. C. Park. *Modern Robotics*. Cambridge University Press, 2017.

N. Mansard, O. Stasse, P. Evrard, and A. Kheddar. A versatile generalized inverted kinematics implementation for collaborative working humanoid robots: The stack of tasks. In *Proceedings of the International Conference on Advanced Robotics*, 2009.

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 *Proceedings of the International Conference on Robotics and Automation (ICRA)*, 2017.

R. Martinez-Cantin, N. de Freitas, A. Doucet, and J. A. Castellanos. Active Policy Learning for Robot Planning and Exploration under Uncertainty. In *Proceedings of Robotics: Science & Systems (RSS)*, 2007.

R. Martinez-Cantin, N. de Freitas, E. Brochu, J. Castellanos, and A. Doucet. A bayesian exploration-exploitation approach for optimal online sensing and planning with a visually guided mobile robot. *Autonomous Robots*, 27(2), 2009.

T. Matsubara, S. Hyon, and J. Morimoto. Learning parametric dynamic movement primitives from multiple demonstrations. *Neural Networks*, 24(5): 493–500, 2011. doi: 10.1016/j.neunet.2011.02.004.

D. Mayne. A second-order gradient method for determining optimal trajectories of non-linear discrete-time systems. *International Journal of Control*, 3(1): 85–95, 1966.

R. S. Michalski, J. G. Carbonell, and T. M. Mitchell. *Machine learning: An artificial intelligence approach*. Springer Science & Business Media, 2013.

B. L. Miller and D. E. Goldberg. Genetic algorithms, selection schemes, and the varying effects of noise. *Evolutionary Computation*, 4(2):113–131, 1996.

V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015.

V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2016.

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 *Proceedings of the International Conference on Robotics and Automation (ICRA)*, 2017.

K. L. Moore, M. Dahleh, and S. Bhattacharyya. Iterative learning control: A survey and new results. *Journal of Field Robotics*, 9(5):563–594, 1992.

C. Moulin-Frier and P.-Y. Oudeyer. Exploration strategies in developmental robotics: a unified probabilistic framework. In *Proceedings of the Joint International Conference on Development and Learning and Epigenetic Robotics (ICDL)*, 2013.

J.-B. Mouret. Novelty-based Multiobjectivization. In *New Horizons in Evolutionary Robotics*, pages 139–154. Springer Berlin/Heidelberg, 2011.

J.-B. Mouret. Micro-data learning: The other end of the spectrum. *ERCIM News*, (107):2, 2016.

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

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

J.-B. Mouret and S. Doncieux. Sferes$_{v2}$: Evolvin'in the multi-core world. In *Proceedings of Congress on Evolutionary Computation (CEC)*, 2010.

J.-B. Mouret and S. Doncieux. Encouraging Behavioral Diversity in Evolutionary Robotics: an Empirical Study. *Evolutionary Computation*, 20(1):91–133, 2012.

R. M. Murray. *A mathematical introduction to robotic manipulation*. CRC press, 2017.

J. A. Musick and C. J. Limpus. Habitat utilization and migration in juvenile sea turtles. *The biology of sea turtles*, 1:137–163, 1997.

G. Nelson, A. Saunders, N. Neville, B. Swilling, J. Bondaryk, D. Billings, C. Lee, R. Playter, and M. Raibert. Petman: A humanoid robot for testing chemical protective clothing. *Journal of the Robotics Society of Japan*, 30(4): 372–377, 2012.

A. Y. Ng and M. Jordan. PEGASUS: a policy search method for large MDPs and POMDPs. In *Proceedings of Uncertainty in Artificial Intelligence*, 2000.

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

A. Nguyen, J. Yosinski, and J. Clune. Deep neural networks are easily fooled: High confidence predictions for unrecognizable images. In *Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015a.

A. Nguyen, J. Yosinski, and J. Clune. Understanding Innovation Engines: Automated Creativity and Improved Stochastic Optimization via Deep Learning. *Evolutionary Computation*, 24:545–572, 2016.

A. M. Nguyen, J. Yosinski, and J. Clune. Innovation engines: Automated creativity and improved stochastic optimization via deep learning. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, 2015b.

D. Nguyen-Tuong and J. Peters. Using model knowledge for learning inverse dynamics. In *Proceedings of the International Conference on Robotics and Automation (ICRA)*, 2010.

D. Nguyen-Tuong and J. Peters. Model learning for robot control: a survey. *Cognitive Processing*, 12(4):319–340, 2011.

N. J. Nilsson. Shakey the robot. Technical Report 323, 1984.

F. Nori, S. Traversaro, J. Eljaik, F. Romano, A. Del Prete, and D. Pucci. iCub whole-body control through force regulation on rigid non-coplanar contacts. *Frontiers in Robotics and AI*, 2:6, 2015.

C. Null and B. Caulfield. Fade To Black The 1980s vision of "lights-out" manufacturing, where robots do all the work, is a dream no more., 2003.

J. Oh, X. Guo, H. Lee, R. L. Lewis, and S. Singh. Action-conditional video prediction using deep networks in atari games. In *Advances in Neural Information Processing Systems (NIPS)*, pages 2863–2871, 2015.

A. O'Hagan. Bayes-hermite quadrature. *Journal of Statistical Planning and Inference*, 1991.

P.-Y. Oudeyer. Curiosity and languages. *Catalogue of the Exhibition" Mathematics: A Beautiful Elsewhere", Fondation Cartier pour l'Art Contemporain*, 2011.

P.-Y. Oudeyer. Computational Theories of Curiosity-Driven Learning. *arXiv preprint arXiv:1802.10546*, 2018.

P.-Y. Oudeyer, F. Kaplan, V. V. Hafner, and A. Whyte. The playground experiment: Task-independent development of a curious robot. In *Proceedings of the AAAI Spring Symposium on Developmental Robotics*, 2005.

P.-Y. Oudeyer, F. Kaplan, and V. V. Hafner. Intrinsic Motivation Systems for Autonomous Mental Development. *IEEE transactions on Evolutionary Computation*, 11(2):265–286, 2007.

V. Padois, S. Ivaldi, J. Babič, M. Mistry, J. Peters, and F. Nori. Whole-body multi-contact motion in humans and humanoids: Advances of the CoDyCo European project. *Robotics and Autonomous Systems*, 90:97–117, 2017.

V. Papaspyros, K. Chatzilygeroudis, V. Vassiliades, and J.-B. Mouret. Safety-aware robot damage recovery using constrained bayesian optimization and simulated priors. In *BayesOpt'16: Proceedings of the International Workshop "Bayesian Optimization: Black-box Optimization and Beyond" at NIPS*, 2016.

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

S. Paul, K. Chatzilygeroudis, K. Ciosek, J.-B. Mouret, M. A. Osborne, and S. Whiteson. Alternating Optimisation and Quadrature for Robust Control. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence (AAAI)*, 2018.

R. Pautrat, K. Chatzilygeroudis, and J.-B. Mouret. Bayesian optimization with automatic prior selection for data-efficient direct policy search. In *Proceedings of the International Conference on Robotics and Automation (ICRA)*, 2018.

X. B. Peng, M. Andrychowicz, W. Zaremba, and P. Abbeel. Sim-to-real transfer of robotic control with dynamics randomization. *arXiv preprint arXiv:1710.06537*, 2017.

R. Pfeifer and J. Bongard. *How the body shapes the way we think: a new view of intelligence*. MIT press, 2006.

C. Plagemann, S. Mischke, S. Prentice, K. Kersting, N. Roy, and W. Burgard. Learning predictive terrain models for legged robot locomotion. In *Proceedings of the International Conference on Intelligent Robots (IROS)*, 2008.

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

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

J. Queisser and J. Steil. Bootstrapping of Parameterized Skills Through Hybrid Optimization in Task and Policy Spaces. *Frontiers in Robotics and AI*, 2018. doi: 10.3389/frobt.2018.00049.

J. Quiñonero-Candela and C. E. Rasmussen. A unifying view of sparse approximate Gaussian process regression. *Journal of Machine Learning Research*, 6:1939–1959, Dec 2005.

M. Raibert, K. Blankespoor, G. Nelson, and R. Playter. Bigdog, the rough-terrain quadruped robot. In *Proceedings of IFAC*, pages 10822–10825. Elsevier, 2008.

A. Rajeswaran, S. Ghotra, B. Ravindran, and S. Levine. Epopt: Learning robust neural network policies using model ensembles. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2017.

C. E. Rasmussen and Z. Ghahramani. Bayesian monte carlo. In *Advances in Neural Information Processing Systems (NIPS)*, 2003.

C. E. Rasmussen and C. K. Williams. *Gaussian processes for machine learning*, volume 1. MIT press Cambridge, 2006.

J. B. Rawlings and D. Q. Mayne. Model predictive control: Theory and design. 2009.

J. Rieffel and J.-B. Mouret. Soft tensegrity robots. *Soft Robotics*, 2018.

P. Rolet, M. Sebag, and O. Teytaud. Boosting active learning to optimality: A tractable monte-carlo, billiard-based algorithm. In *Proceedings of the European Conference on Machine Learning (ECML)*, 2009.

P. Rolland, J. Scarlett, I. Bogunovic, and V. Cevher. High-Dimensional Bayesian Optimization via Additive Models with Overlapping Groups. In *Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2018.

T. H. Rowan. Functional stability analysis of numerical algorithms. 1990.

G. A. Rummery and M. Niranjan. *On-line Q-learning using connectionist systems*, volume 37. University of Cambridge, Department of Engineering Cambridge, England, 1994.

S. J. Russell and P. Norvig. *Artificial intelligence: a modern approach*. Malaysia; Pearson Education Limited,, 2016.

R. M. Ryan and E. L. Deci. Intrinsic and extrinsic motivations: Classic definitions and new directions. *Contemporary educational psychology*, 25(1): 54–67, 2000.

S. Sæmundsson, K. Hofmann, and M. P. Deisenroth. Meta reinforcement learning with latent variable gaussian processes. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence (UAI)*, 2018.

J. Salini, V. Padois, and P. Bidaud. Synthesis of complex humanoid whole-body behavior: a focus on sequencing and tasks transitions. In *Proceedings of the International Conference on Robotics and Automation (ICRA)*, 2011.

M. Saveriano, Y. Yin, P. Falco, and D. Lee. Data-efficient control policy search using residual dynamics learning. In *Proc. of IROS*, 2017.

J. Schmidhuber. Developmental robotics, optimal artificial curiosity, creativity, music, and the fine arts. *Connection Science*, 18(2):173–187, 2006.

J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz. Trust region policy optimization. In *International Conference on Machine Learning (ICML)*, pages 1889–1897, 2015.

R. Sellaouti, O. Stasse, S. Kajita, K. Yokoi, and A. Kheddar. Faster and smoother walking of humanoid hrp-2 with passive toe joints. In *Proceedings of the International Conference on Intelligent Robots and Systems (IROS)*, 2006.

R. J. Serfling. Probability inequalities for the sum in sampling without replacement. *The Annals of Statistics*, pages 39–48, 1974.

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*, 104(1):148–175, 2016.

B. Siciliano and O. Khatib. *Springer handbook of robotics*. Springer, 2 edition, 2016.

O. Sigaud and F. Stulp. Policy search in continuous action domains: an overview. *arXiv preprint arXiv:1803.04706*, 2018.

D. Silver and J. Veness. Monte-carlo planning in large POMDPs. In *Advances in Neural Information Processing Systems (NIPS)*, 2010.

D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller. Deterministic policy gradient algorithms. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2014.

D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, et al. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, et al. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *arXiv preprint arXiv:1712.01815*, 2017a.

D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, et al. Mastering the game of go without human knowledge. *Nature*, 550(7676):354, 2017b.

K. Sims. Evolving virtual creatures. In *Proceedings of the Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH)*, 1994.

J. Snoek, K. Swersky, R. Zemel, and R. Adams. Input warping for bayesian optimization of non-stationary functions. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2014.

J. Spitz, K. Bouyarmane, S. Ivaldi, and J.-B. Mouret. Trial-and-error learning of repulsors for humanoid qp-based whole-body control. In *Proceedings of the International Conference on Humanoid Robots (Humanoids)*, 2017.

M. W. Spong and D. J. Block. The pendubot: A mechatronic system for control research and education. In *Proceedings of Decision and Control*, 1995.

N. Srinivas, A. Krause, S. M. Kakade, and M. Seeger. Gaussian process optimization in the bandit setting: no regret and experimental design. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2010.

K. O. Stanley and R. Miikkulainen. Evolving neural networks through augmenting topologies. *Evolutionary Computation*, 2002.

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*, 2013a.

F. Stulp and O. Sigaud. Robot skill learning: From reinforcement learning to evolution strategies. *Paladyn. Journal of Behavioral Robotics*, 4(1):49–61, September 2013b.

F. Stulp, E. Theodorou, and S. Schaal. Reinforcement learning with sequences of motion primitives for robust manipulation. *IEEE Transactions on Robotics*, 28(6):1360–1370, 2012.

F. Stulp, G. Raiola, A. Hoarau, S. Ivaldi, and O. Sigaud. Learning compact parameterized skills with a single regression. In *Proceedings of the International Conference on Humanoid Robots (Humanoids)*, 2013.

M. Sugiyama, I. Takeuchi, T. Suzuki, T. Kanamori, H. Hachiya, and D. Okanohara. Least-squares conditional density estimation. *IEICE Transactions on Information and Systems*, 93(3):583–594, 2010.

R. S. Sutton. Dyna, an integrated architecture for learning, planning, and reacting. *ACM SIGART Bulletin*, 2(4):160–163, 1991.

R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge, 1998.

R. S. Sutton, D. A. McAllester, S. P. Singh, and Y. Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Advances in Neural Information Processing Systems (NIPS)*, pages 1057–1063, 2000.

Y. Taigman, M. Yang, M. Ranzato, and L. Wolf. Deepface: Closing the gap to human-level performance in face verification. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2014.

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*, 57:128–140, 2014.

M. E. Taylor and P. Stone. Transfer learning for reinforcement learning domains: A survey. *Journal of Machine Learning Research*, 10(Jul):1633–1685, 2009.

E. Todorov and W. Li. A generalized iterative lqg method for locally-optimal feedback control of constrained nonlinear stochastic systems. In *Proceedings of the American Control Conference*, 2005.

N. G. Tsagarakis, G. Metta, G. Sandini, D. Vernon, R. Beira, F. Becchi, L. Righetti, J. Santos-Victor, A. J. Ijspeert, M. C. Carrozza, et al. icub: the design and realization of an open humanoid platform for cognitive and neuroscience research. *Advanced Robotics*, 21(10):1151–1175, 2007.

S. Tsutsui and A. Ghosh. Genetic algorithms with a robust solution searching scheme. *IEEE transactions on Evolutionary Computation*, 1(3):201–208, 1997.

S. C. Turaga, J. F. Murray, V. Jain, F. Roth, M. Helmstaedter, K. Briggman, W. Denk, and H. S. Seung. Convolutional networks can learn to generate affinity graphs for image segmentation. *Neural Computation*, 22(2):511–538, 2010.

R. Vaillant, C. Monrocq, and Y. Le Cun. Original approach for the localisation of objects in images. *IEE Proceedings - Vision, Image and Signal Processing*, 141(4):245–250, 1994.

H. Van Seijen, H. Van Hasselt, S. Whiteson, and M. Wiering. A theoretical and empirical analysis of Expected Sarsa. In *Proceedings of the Symposium on Adaptive Dynamic Programming and Reinforcement Learning (ADPRL)*, 2009.

V. Vassiliades, K. Chatzilygeroudis, and J.-B. Mouret. Using centroidal Voronoi tessellations to scale up the multi-dimensional archive of phenotypic elites algorithm. *IEEE Transactions on Evolutionary Computation*, 2017.

V. Verma, G. Gordon, R. Simmons, and S. Thrun. Real-time fault diagnosis. *IEEE Robotics & Automation Magazine*, 11(2):56–66, 2004.

N. Wahlström, T. B. Schön, and M. P. Desienroth. Learning deep dynamical models from image pixels. In *Proceedings of the 17th IFAC Symposium on System Identification (SYSID)*, 2015.

Y. Wang, D. Zhou, and F. Gao. Iterative learning model predictive control for multi-phase batch processes. *Journal of Process Control*, 18(6):543–557, 2008.

Z. Wang, F. Hutter, M. Zoghi, D. Matheson, and N. de Feitas. Bayesian optimization in a billion dimensions via random embeddings. *Journal of Artificial Intelligence Research*, 55:361–387, 2016.

I. O. R. Warren B. Powell. *Optimal Learning*. Wiley Series in Probability and Statistics, 2012.

C. J. Watkins and P. Dayan. Q-learning. *Machine learning*, 8(3-4):279–292, 1992.

P. Wawrzynski. Learning to control a 6-degree-of-freedom walking robot. In *Proceedings of the International Conference on "Computer as a Tool" (EUROCON)*, 2007.

P. Werbos. Approximate dynamic programming for realtime control and neural modelling. *Handbook of intelligent control: neural, fuzzy and adaptive approaches*, pages 493–525, 1992.

B. J. Williams, T. J. Santner, and W. I. Notz. Sequential design of computer experiments to minimize integrated response functions. *Statistica Sinica*, 2000.

R. J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 1992.

A. Wilson, A. Fern, and P. Tadepalli. Using trajectory data to improve bayesian optimization for reinforcement learning. *Journal of Machine Learning Research*, 15(1):253–282, 2014.

T. Wu and J. Movellan. Semi-parametric Gaussian process for robot system identification. In *Proceedings of the International Conference on Intelligent Robots and Systems (IROS)*, 2012.

K. Yang, D. Gaida, T. Bäck, and M. Emmerich. Expected hypervolume improvement algorithm for PID controller tuning and the multiobjective dynamical control of a biogas plant. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC)*, 2015.

W. Yu, C. K. Liu, and G. Turk. Preparing for the unknown: Learning a universal policy with online system identification. In *Proceedings of Robotics: Science and Systems (RSS)*, 2017.

T. Zhang, G. Kahn, S. Levine, and P. Abbeel. Learning deep control policies for autonomous aerial vehicles with mpc-guided policy search. In *Proceedings of the International Conference on Robotics and Automation (ICRA)*, 2016.

S. Zhu, A. Kimmel, K. E. Bekris, and A. Boularias. Fast Model Identification via Physics Engines for Data-Efficient Policy Search. In *Proceedings of the International Joint Conferences on Artificial Intelligence (IJCAI)*, 2018.

M. Zimmer, Y. Boniface, and A. Dutech. Neural Fitted Actor-Critic. In *Proceedings of the European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning (ESANN)*, 2016.

Appendix A

Racing bootstrap pseudo-code

Algorithm 18 Evaluate Candidates with bootstrap and racing

procedure RacingBootstrap($\{\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_\lambda\}$, μ, n_{step}, $n_{\max}$, N, τ_{boot}, τ_{race})
  $\mathbb{S} = \emptyset$ ▷ Set of selected candidates for racing
  $\mathbb{U} = \{\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_\lambda\}$ ▷ Set of undecided candidates for racing
  $\mathbb{A} = \{\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_\lambda\}$ ▷ Set of alive candidates for self-accuray
  $n \leftarrow n_{\text{step}}$
  for all $\boldsymbol{\theta}_i \in \mathbb{A}$ **do**
    for $k = 1, \ldots, n$ **do**
      $G_i^k \leftarrow G(\boldsymbol{\theta}_i)$
    end for
  end for
  while $n < n_{\max}$ **and** $\mathbb{A} > 0$ **and** $|\mathbb{S}| < \mu$ **do**
    $n \leftarrow n + n_{\text{step}}$
    for all $\boldsymbol{\theta}_i \in \mathbb{A}$ **do**
      values $= \{G_i^1, \ldots, G_i^n\}$
      for $j = 1, \ldots, N$ **do**
        bvalues $=$ bootstrap(values)
        $\delta_i^j = \overline{\text{bvalues}} - \overline{\text{values}}$ ▷ $\overline{\text{val}}$ is the mean of the values
      end for
      $\Delta_i = \overline{|\delta_i|}$ ▷ Average absolute error
      if $\Delta_i < \tau_{\text{boot}} * \overline{\text{values}}$ **then** ▷ Self-accuracy check
        $\mathbb{A} \leftarrow \mathbb{A} \setminus \{\boldsymbol{\theta}_i\}$
        break
      end if
      svalues $=$ sort(δ_i) ▷ Sort the bootstrapped deltas
      max $= \tau_{\text{race}} * N$ ▷ Find index of upper bound
      min $= (1 - \tau_{\text{race}}) * N$ ▷ Find index of lower bound
      $\text{LB}_i = \overline{\text{values}} - \text{svalues}_{\text{min}}$ ▷ Lower bound
      $\text{UB}_i = \overline{\text{values}} - \text{svalues}_{\text{max}}$ ▷ Upper bound
    end for
    for $\boldsymbol{\theta}_i \in \mathbb{U}$ **do**
      if $|\boldsymbol{\theta}_j \in \mathbb{U} | \text{LB}_i < \text{UB}_j| \geq |\mathbb{S}| + |\mathbb{U}| - \mu$ **then**
        // probably among the best μ
        $\mathbb{S} \leftarrow \mathbb{S} \cup \{\boldsymbol{\theta}_i\}$ // select it
        $\mathbb{A} \leftarrow \mathbb{A} \setminus \{\boldsymbol{\theta}_i\}$
        $\mathbb{U} \leftarrow \mathbb{U} \setminus \{\boldsymbol{\theta}_i\}$
      else if $|\boldsymbol{\theta}_j \in \mathbb{U} | \text{UB}_i < \text{LB}_j| \geq \mu - |\mathbb{S}|$ **then**
        // probably not among the best μ
        $\mathbb{U} \leftarrow \mathbb{U} \setminus \{\boldsymbol{\theta}_i\}$ // discard it
        $\mathbb{A} \leftarrow \mathbb{A} \setminus \{\boldsymbol{\theta}_i\}$
      end if
    end for
    for all $\boldsymbol{\theta}_i \in \mathbb{A}$ **do** ▷ Evaluate remaining active candidates
      for $k = n - n_{\text{step}}, \ldots, n$ **do**
        $G_i^k \leftarrow G(\boldsymbol{\theta}_i)$
      end for
    end for
  end while
end procedure