Bayesian Optimization in Robot Learning

Automatic Controller Tuning and Sample-Efficient Methods

Dissertation

der Mathematisch-Naturwissenschaftlichen Fakultät
der Eberhard Karls Universität Tübingen
zur Erlangung des Grades eines
Doktors der Naturwissenschaften
(Dr. rer. nat.)

vorgelegt von
M.Sc. Alonso Marco-Valle
aus Villanueva de los Infantes,
Castilla La-Mancha, Spanien

Tübingen
2020

Gedruckt mit Genehmigung der Mathematisch-Naturwissenschaftlichen Fakultät der Eberhard Karls Universität Tübingen.

Tag der mündlichen Qualifikation:	13.07.2020
Dekan:	Prof. Dr. Wolfgang Rosenstiel
1. Berichterstatter:	Prof. Dr. Sebastian Trimpe
2. Berichterstatter:	Prof. Dr. Philipp Hennig

Abstract

The problem of designing controllers to regulate dynamical systems has been studied by engineers during the past millennia. Ever since, suboptimal performance lingers in many closed loops as an unavoidable side effect of manually tuning the parameters of the controllers. Nowadays, industrial settings remain skeptic about data-driven methods that allow one to automatically learn controller parameters. In the context of robotics, machine learning (ML) keeps growing its influence on increasing autonomy and adaptability, for example to aid automating controller tuning. However, data-hungry ML methods, such as standard reinforcement learning, require a large number of experimental samples, prohibitive in robotics, as hardware can deteriorate and break. This brings about the following question:

> *Can manual controller tuning, in robotics, be automated by using data-efficient machine learning techniques?*

In this thesis, we tackle the question above by exploring Bayesian optimization (BO), a data-efficient ML framework, to buffer the human effort and side effects of manual controller tuning, while retaining a low number of experimental samples. We focus this work in the context of robotic systems, providing thorough theoretical results that aim to increase data-efficiency, as well as demonstrations in real robots. Specifically, we present four main contributions.

We first consider using BO to replace manual tuning in robotic platforms. To this end, we parametrize the design weights of a linear quadratic regulator (LQR) and learn its parameters using an information-efficient BO algorithm. Such algorithm uses Gaussian processes (GPs) to model the unknown performance objective. The GP model is used by BO to suggest controller parameters that are expected to increment the information about the optimal parameters, measured as a gain in entropy. The resulting "automatic LQR tuning" framework is demonstrated on two robotic platforms: A robot arm balancing an inverted pole and a humanoid robot performing a squatting task. In both cases, an existing controller is automatically improved in a handful of experiments without human intervention.

BO compensates for data scarcity by means of the GP, which is a probabilistic model that encodes prior assumptions about the unkown performance objective. Usually, incorrect or non-informed assumptions have negative consequences, such as higher number of robot experiments, poor tuning performance or reduced sample-

efficiency. The second to fourth contributions presented herein attempt to alleviate this issue.

The second contribution proposes to include the robot simulator into the learning loop as an additional information source for automatic controller tuning. While doing a real robot experiment generally entails high associated costs (e.g., require preparation and take time), simulations are cheaper to obtain (e.g., they can be computed faster). However, because the simulator is an imperfect model of the robot, its information is biased and could have negative repercussions in the learning performance. To address this problem, we propose "simu-vs-real", a principled multi-fidelity BO algorithm that trades off cheap, but inaccurate information from simulations with expensive and accurate physical experiments in a cost-effective manner. The resulting algorithm is demonstrated on a cart-pole system, where simulations and real experiments are alternated, thus sparing many real evaluations.

The third contribution explores how to adequate the expressiveness of the probabilistic prior to the control problem at hand. To this end, the mathematical structure of LQR controllers is leveraged and embedded into the GP, by means of the *kernel* function. Specifically, we propose two different "LQR kernel" designs that retain the flexibility of Bayesian nonparametric learning. Simulated results indicate that the LQR kernel yields superior performance than non-informed kernel choices when used for controller learning with BO.

Finally, the fourth contribution specifically addresses the problem of handling controller failures, which are typically unavoidable in practice while learning from data, specially if non-conservative solutions are expected. Although controller failures are generally problematic (e.g., the robot has to be emergency-stopped), they are also a rich information source about what should be avoided. We propose "failures-aware excursion search", a novel algorithm for Bayesian optimization under black-box constraints, where failures are limited in number. Our results in numerical benchmarks indicate that by allowing a confined number of failures, better optima are revealed as compared with state-of-the-art methods.

The first contribution of this thesis, "automatic LQR tuning", lies among the first on applying BO to real robots. While it demonstrated automatic controller learning from few experimental samples, it also revealed several important challenges, such as the need of higher sample-efficiency, which opened relevant research directions that we addressed through several methodological contributions. Summarizing, we proposed "simu-vs-real", a novel BO algorithm that includes the simulator as an additional information source, an "LQR kernel" design that learns faster than standard choices and "failures-aware excursion search", a new BO algorithm for constrained black-box optimization problems, where the number of failures is limited.

Zusammenfassung

Das Problem des Reglerentwurfs für dynamische Systeme wurde von Ingenieuren in den letzten Jahrtausenden untersucht. Seit diesen Tagen ist suboptimales Verhalten ein unvermeidlicher Nebeneffekt der manuellen Einstellung von Reglerparametern. Heutzutage steht man in industriellen Anwendungen datengestriebenen Methoden, die das automatische Lernen von Reglerparametern ermöglichen, nach wie vor skeptisch gegenüber. Im Bereich der Robotik gewinnt das maschinelle Lernen (ML) immer mehr an Einfluss und ermöglicht einen erhöhten Grad der Autonomie und Anpassungsfähigkeit, z.B. indem es dabei unterstützt, den Prozess der Reglereinstellung zu automatisieren. Datenintensive Methoden, wie z.B. Methoden des Reinforcement Learning, erfordern jedoch eine große Anzahl experimenteller Versuche, was in der Robotik nicht möglich ist, da die Hardware sich abnutzt und kaputt gehen kann. Das wirft folgende Frage auf:

> *Kann die manuelle Reglereinstellung in der Robotik durch den Einsatz dateneffizienter Techniken des maschinellen Lernens ersetzt werden?*

In dieser Arbeit gehen wir die obige Frage an, indem wir den Einsatz von Bayes'scher Optimierung (BO), ein dateneffizientes ML-Framework, als Ersatz für manuelles Einstellen unter Beibehaltung einer geringen Anzahl von experimentellen Versuchen untersuchen. Der Fokus dieser Arbeit liegt auf Robotersystemen. Dabei präsentieren wir Demonstrationen mit realen Robotern, sowie fundierte theoretische Ergebnisse zur Steigerung der Dateneffizienz. Im Einzelnen stellen wir vier Hauptbeiträge vor.

Zunächst betrachten wir die Verwendung von BO als Ersatz für das manuelle Einstellen auf einer Roboterplattform. Zu diesem Zweck parametrisieren wir die Einstellgewichtungen eines linear-quadratischen Reglers (LQR) und lernen diese Parameter mit einem informationseffizienten BO-Algorithmus. Dieser Algorithmus nutzt Gauß-Prozesse (GPs), um das unbekannte Zielfunktion zu modellieren. Das GP-Modell wird vom BO-Algorithmus genutzt, um Reglerparameter vorzuschlagen von denen erwartet wird, dass sie die Informationen über die optimalen Parameter erhöhen, gemessen als eine Zunahme der Entropie. Das resultierende Framework zur *automatischen LQR-Einstellung* wird auf zwei Roboterplattformen demonstriert: Ein Robterarm, der einen umgekehrten Stab ausbalanciert und ein humanoider Roboter, der Kniebeugen ausführt. In beiden Fällen wird ein vorhandener Regler in einer handvoll Experimenten automatisch verbessert, ohne dass ein Mensch eingreifen muss.

BO kompensiert Datenknappheit durch den GP, ein probabilistisches Modell, das a priori Annahmen über das unbekannte Zielfunktion enthält. Normalerweise haben falsche oder uninformierte Annahmen negative Folgen, wie z.B. eine höhere Anzahl von Roboterexperimenten, ein schlechteres Reglerverhalten oder eine verringerte Dateneffizienz. Die hier vorgestellten Beiträge Zwei bis Vier beschäftigen sich mit diesem Problem.

Der zweite Beitrag schlägt vor, den Robotersimulator als zusätzliche Informationsquelle für die automatische Reglereinstellung in die Lernschleife miteinzubeziehen. Während reale Roboterexperimente im Allgemeinen hohe Kosten mit sich bringen, sind Simulationen günstiger (sie können z.B. schneller berechnet werden). Da der Simulator aber ein unvollkommenes Modell des Roboters ist, sind seine Informationen einseitig verfälscht und können negative Auswirkungen auf das Lernverhalten haben. Um dieses Problem anzugehen, schlagen wir "sim-vs-real" vor, einen auf grundlegenden Prinzipien beruhenden BO-Algorithmus, der Daten aus Simulationen und Experimenten nutzt. Der Algorithmus wägt dabei die günstigen, aber ungenauen Informationen des Simulators gegen die teuren und exakten physikalischen Experimente in einer kostengünstigen Weise ab. Der daraus resultierende Algorithmus wird an einem inversen Pendels auf einem Wagen demonstriert, bei dem sich Simulationen und reale Experimente abwechseln, wodurch viele reale Experimente eingespart werden.

Der dritte Beitrag untersucht, wie die Aussagekraft der probabilistischen Annahmen des vorliegenden Regelungsproblem adäquat behandelt werden kann. Zu diesem Zweck wird die mathematische Struktur des LQR-Reglers genutzt und durch die *Kernel*-Funktion in den GP eingebaut. Insbesondere schlagen wir zwei verschiedene "LQR-Kernel"-Entwürfe vor, die die Flexibilität des Bayes'schen, nichtparametrischen Lernens beibehalten. Simulierte Ergebnisse deuten darauf hin, dass die LQR-Kernel bessere Ergebnisse erzielen als uninformierte Kernel, wenn sie zum Lernen von Reglern mit BO verwendet werden.

Der vierte Beitrag schließlich befasst sich speziell mit dem Problem, wie ein Versagen des Reglers behandelt werden soll. Fehlschläge von Reglern sind beim Lernen aus Daten typischerweise unvermeidbar, insbesondere wenn nichtkonservative Lösungen erwartet werden. Obwohl ein Versagen des Reglers im Allgemeinen problematisch ist (z.B. muss der Roboter mit einem Not-Aus angehalten werden), ist es gleichzeitig eine reichhaltige Informationsquelle darüber, was vermieden werden sollte. Wir schlagen "failures-aware excursion search" vor, einen neuen Algorithmus für Bayes'sche Optimierung mit unbekannten Beschränkungen, bei dem die Anzahl an Fehlern begrenzt ist. Unsere Ergebnisse in numerischen Vergleichsstudien deuten darauf hin, dass, verglichen mit dem aktuellen Stand der Technik, durch das Zu-

lassen einer begrenzten Anzahl von Fehlschlägen bessere Optima aufgedeckt werden.

Der erste Beitrag dieser Dissertation ist unter den ersten die BO an realen Robotern anwenden. Diese Arbeit diente dazu, mehrere Probleme zu identifizieren, wie zum Beispiel den Bedarf nach einer höheren Dateneffizienz, was mehrere neue Forschungsrichtungen aufzeigte, die wir durch verschiedene methodische Beiträge addressiert haben. Zusammengefasst haben wir "sim-vs-real", einen neuen BO-Algorithmus der den Simulator as zusätzliche Informationsquelle miteinbezieht, einen "LQR-Kernel"-Entwurf, der schneller lernt als Standardkernel und "failures-aware excursion search", einen neuen BO-Algorithmus für beschränkte Black-Box-Optimierungsprobleme, bei denen die Anzahl der Fehler begrenzt ist, vorgeschlagen.

Acknowledgements

The path of a PhD student is never steady. On the contrary, it's a huge roller coaster, where hills and valleys appear and disappear far beyond your prediction horizon. That's what makes it the opposite of boring. But that's also what makes it the opposite of easy. Despite this bittersweet sensation, if I could condition my past choices on the current observations, I definitely would do a PhD again.
However, if I were to walk over my steps, I would (i) never let the pass of time be measured by my progress in work, (ii) dedicate a bit more time to myself, and (iii) put higher weight into the positive feedback.

I thank my PhD advisor, Prof. Dr. Sebastian Trimpe, for his constant support during this long marathon; for his useful advices, both in research, but also in life; for his patience, his trust in my research ideas and his willingness to always make it easy for me. Together, we have steered our research along the right course, which has resulted in many successful outcomes, which I will always thank him for. I also thank him for attending one of my theatre plays, for our nice conversations in Berkeley, for many Friday beers we spent together, and for those late evenings of work in the old AMD lab when Apollo was still not balancing the pole. All of that, together with his support in the present time, constitute one of the greatest outcomes of my PhD.

I thank Prof. Dr. Philipp Hennig for always having his door open to me, for so many interesting discussions, for being an unending source of interesting ideas and for all the great moments we spent in the Friday beers during the old AMD times.

I thank Dr. Francisco Ramos De la Flor, because he never stops believing in me. Good luck in life presents itself to those who work hard to deserve it. He has been my good luck, and thanks to him, I am where I am.

I thank Dominik Baumann, who has been and *is* my friend, colleague, intellectual companion, office mate, beer passionate, and inconditional support during this long adventure, both in the joyful and sad moments.

I give special thanks to Alexander von Rohr for his never ending support in our research projects, our intellectually stimulating conversations, for being so patient

and for having this great sense of humor that transforms work in fun.

I thank Friedrich Solowjow for his great support, intellectual discussions, for being the other “night owl” in the lab, and for always keeping it cool.

I thank Andreas René Geist, Mona Buisson-Fenet, Steve Heim, Christian Fiedler, and all the Intelligent Control Systems members for having contributed to build a cheerful, pleasant, safe and stimulating environment in the lab, that has fostered great research. Also for being so cool people.

I thank Lidia Pavel for always receiving me in her office with a smile, and being so helpful.

I thank Kara Loehr for always bringing joy and good mood to everyone around.

I thank Vincent Berenz for always willing to help me when I got stuck, for helping me to develop part of my research code and for all the fun conversations over lunch and coffee breaks.

I thank Daniel Kappler for all the nice times in the flat and for his support in many key moments.

I thank Felix Grimminger for his unconditional support from the first day I joined the AMD lab. He literally created many pieces of hardware that I used for my research, and hence was a pillar that made it possible. He is the *Sine qua non* that makes robotics possible in Tübingen.

I thank Jeannete Bohg for our long walk-breaks in the AMD times, for being a source of understanding in life and work matters and for all the time she dedicated to help me when I first joined the AMD lab.

I thank Cristina García Cifuentes, Alexander Herzog, Jim Mainprice, Franziska Meier, Stefan Schaal and all the members of the old AMD lab. It was so much fun to work with them.

I thank Alina Kloss for all the nice moments in the lab, over lunch, and in sushi dinners in Tübingen. She has been a great companion during my PhD and she made my day in uncountable occasions.

I thank Manuel Wüthrich for always willing to have a whiteboard discussion, but also for having this wild party spirit.

I thank Julian Viereck for never saying no to a concert, for asking me to join the Tübingen choir, and for being a great source of support.

I thank Bilal Hammoud, Maximilien Naveau, Brahayam Ponton, Ahmad Gazar, Majid Khadiv, Miroslav Bogdanovic, Ludovic Righetti and all the NYU robotics team for being such a cool crowd and always be willing to take a coffee break.

I thank Cristina Pinneri for listening and supporting me when I most needed it and for being such a cool person.

I thank Sarah Bechtle for being a friend, a colleague, a support, and for showing me the best pesto recipe. It feels nicer when she is around.

I thank Sean Mason and Nick Rotella for the time they dedicated to our research project in USC, for always willing to grab street tacos for lunch, and for showing me the best places to eat burgers in Los Angeles.

I thank Giovanni Sutanto for the great and fun moments we spent in USC and in Facebook, both outside and inside the lab.

I thank Austin Wang and Yixin lin for super cool moments in the music room in Facebook, and for always be willing to eat in the BBQ place.

I thank Roberto Calandra for making my Facebook internship a real pleasure, and for all the interesting research discussions and fun dinner.

I thank Neha Das for the cool moments we spent in San Francisco.

I thank my flatmates from my PhD stay in the University of Cambridge for many fun evenings, and specially to Carline Kunst, for letting me play her piano, and letting me hear her wonderful music.

I thank Robert Pinsler, Erik Daxberger and Gregor Simm for many fun discussions and experiences during my time in Cambridge.

I thank Yevgen Chebotar, Bharath Sankaran, and Artem Molchanov for fun moments in USC.

I thank Andrea Bajcsy for the great time we spent together when she visited Tübingen.

I thank Cristóbal Araya Altamirano for allowing me to create music by his side, for being a companion in my artistic development in Tübingen, and for having made possible to counterpoint my research with playing in a band.

I thank my friend Damián Pérez Escribano for being my best and unconditional friend.

I thank my sister, for her constant support, her surprise parties, her willingness to always make our relationship better, and for her warmth.

I thank my parents for supporting and encouraging my decision of living abroad, for telling me to always look forward, and for teaching me to draw a smile in the face of the storm.

I thank all the members of my family, who represent *the joy*.

I thank all my friends, who are happily uncountable.

This work was supported by the Max Planck Society, the Max Planck ETH Center for Learning Systems, the International Max Planck Research School for Intelligent Systems (IMPRS-IS), the National Science Foundation grants IIS-1205249, IIS-1017134, EECS-0926052, the Office of Naval Research, and the Okawa Foundation.

Contents

«To my uncle Alonso»

Chapter 1

Introduction

Despite the current inertia toward the era of automation, controller parameters of closed-loop dynamical systems are still tuned manually. This is a tedious and laborious task, which usually requires human effort and entails associated costs. Machine learning has potential to alleviate the need of manual controller tuning. However, many challenges arise when executing this idea in practice.

This introductory chapter discusses those challenges and proposes several ways to overcome them. It is structured as follows. First, in Sec. 1.1, we motivate the need of automatic controller tuning frameworks in the industrial setting. Barriers and challenges that complicate the deployment of automatic controller tuning in robotic platforms are presented in Sec. 1.2. The problem setting, which poses Bayesian optimization as an approach to automate controller tuning, is discussed in Sec. 1.3. Then, in Sec. 1.4, we enumerate the four main contributions of this thesis and outline its structure. Finally, in Sec. 1.5, we discuss existing work in the context of the aforementioned contributions.

1.1 Motivation

Designing controllers to stabilize real systems has been studied by engineers for more than 2000 years (Bennett, 1996). In the 18^{th} century, mostly during the industrial revolution, a prolific period of developments in control theory took place. Therein, many analog controllers (back then named "governors") were implemented in machinery, such as wind and water mills to control their speed. In 1868, J. C. Maxwell described several governor mechanisms using linear differential equations, and proposed conditions to determine their stability (Maxwell, 1868). After 1930, the first PID controller, which was proposed to automate the steering system of U.S. Navy ships (Minorsky, 1922), started gaining popularity. However, its applicability was

limited by the lack of amplification devices, needed to amplify the low power signals obtained from measuring instruments (Bennett, 1984). Further developments in long-distance telephony fostered the creation of signal amplifiers. Then, since 1970, analog controllers started to be replaced by computer-based micro controllers (Ender, 1993). They extended the applicability of control loops to many devices, which allowed for more precise controllers in many industrial sectors, such as petrochemical and paper plants. In the 90's, the PID-type controllers constituted 90% of the controllers used in the industry (Jelali, 2006). Nowadays, they remain present in an uncountable set of applications, such as production lines, large scale construction machinery (e.g., cranes, excavators), power plants, water supply networks and aircrafts. In addition, modern control theory (Ogata and Yang, 2010) keeps studying techniques to "close the loop" in dynamical systems, which has allowed the control of sophisticated machinery, like walking robots (Winkler et al., 2017).

Despite the disruptive technological progress entailed by advances in controller design, many control loops implemented today in the industry exhibit poor performance. Specifically, from thirty to fifteen years ago, controller performance has been reported to decay over time because the initially tuned parameters are never re-tuned, hence performing worse as the properties of the plant change over time (e.g., due to hardware wearing off) (Jelali, 2006). To mitigate this, *robust control* (Zhou et al., 1996) is known for keeping closed-loop stability in the presence of small variations in the dynamics of the plant, at the expense of performance. However, finding good robust controllers also requires manual tuning. Additionally, technicians are usually instructed by the company management to tune controllers until they are "good enough", rather than worrying about reaching optimality (Ender, 1993). These facts heavily contribute to poor plant performance overall and additional costs in the long term.

There exists an important reason for which manual tuning in the industry leads to poor performance: Manually "tweaking" controller parameters is time-consuming, which incurs in high associated costs. However, companies see them as unavoidable fixed costs (Ender, 1993). In general, manual tuning does not only lead to poor performance, but it is also a laborious task, solely based on a trial-and-error process, mainly driven by the engineer's intuition, rather than expertise. It generally entails a waste of valuable assets, such as engineering skills, which implies associated costs. This becomes specially relevant when designing controllers for highly sophisticated dynamical systems, like industrial robots. Therefore, methodologies that enable to switch from "manual" to "automatic" tuning are desirable to mitigate manufacturing costs.

In the research community, machine learning (ML) has been used during the

past decades to increase autonomy and adaptability to new environments. Applications are extensive and range from personalized online recommender systems (Vanchinathan, 2015) to learning control policies in underwater robotics (El-Fakdi and Carreras, 2013). In particular, *robot learning* and *learning control* are two emerging areas that aim at the same goal: Increasing adaptability in controlled dynamical systems. For example, Schaal and Atkeson (2010) review early work that used model-free reinforcement learning (RL) algorithms on real robots: A walking quadruped and a robot arm swinging up an inverted pendulum. Later on, (Mülling et al., 2013) leveraged RL to learn cooperative table tennis from physical interaction with a human and generalize to new situations. In recent work, deep RL is used for learning legged locomotion policies, represented as neural networks, on a real quadruped with little human intervention (Ha et al., 2020).

Generally speaking, ML has potential to help mitigate the efforts associated with manual tuning. However, this raises additional challenges, which are discussed next.

1.2 Challenges

Automatic controller tuning with minimal or no human supervision is a complicated problem (Jelali, 2006). In a nutshell, it requires dispensing with the human judgment out of the manual tuning loop, and completely leaving the tuning process to an algorithm. This raises many concerns, such as safety considerations, need of resilient programs with appropriate user interfaces, renovated engineering skills to handle them, reliability, and noticeable improvement upon manual tuning.

Although machine learning (ML) methods seem appealing for automating the manual tuning process, deploying resilient and reliable learning algorithms that can function without any human supervision is quite challenging. First, the algorithm must be resilient to alterations in the tuning process. For example, the communication between the ML algorithm and the machine could break due to unforeseen signal values, or unmodeled disturbances. Second, the algorithm must deliver a good performing controller in a reduced number of iterations; otherwise, manual tuning would still be preferred. Third, the controller parameters to be learned depend on the specific controller structure we decide to use. Finding a parametrization expressive enough, yet keeping low dimensionality, is difficult.

Furthermore, for automating controller tuning we need data-efficient approaches. The main reason is that we need to conduct experiments on the real system, as these will give the ultimate high-fidelity performance measure that the learning algorithm needs to steer the search toward the optimal parameters. Hence, classical reinforcement learning (RL) and neural network-based methods are usually too data-

hungry to be deployed in real systems, as many robot experiments would need to be collected. This could cause the robot to deteriorate faster and to break more often, which carries high associated costs. Because of this, an open research question is how to leverage scarce experimental data as much as possible in order to make better informed decisions about how to steer the search toward the optimal parameters. In other words, deploying sample-efficient algorithms when learning robot controllers is important to mitigate the number of real experiments.

There exist several possibilities to spare experiments in the real system. For example, realizing simulations using a robot model to acquire rough understanding of the behavior of the controllers. However, combining the simulated data with the real data during the learning process is not straightforward. Another possibility is to inform the learning algorithms with more principled probabilistic models that leverage directly the mathematical structure of the controller problem. Nonetheless, embedding the controller structure into the construction of the probabilistic prior is challenging. Finally, one could attempt to mitigate controller failures while learning, as these can delay the search. Yet, optimizing controller parameters while being constrained to avoid failures also requires investigation.

In the following section, we propose a ML framework that has potential to solve the aforementioned issues.

1.3 Problem statement

A promising framework emerging as a possible solution to the challenges stated in Sec. 1.1 and 1.2 is Bayesian optimization (BO), an area of increasing interest in the ML community. BO comprises a collection of strategies that aim at finding the minimum of an unknown objective, when queries of the objective are limited or scarce. Because BO automatically steers the search toward the optimum, and it respects the lack of abundant data, it can do both, serve as a replacement for manual tuning while retaining sample efficiency.

BO perfectly exemplifies the *no-free-lunch theorem* (Wolpert and Macready, 1997): because it attempts to find the optimal controller with scarce data, it also needs to be supported on prior assumptions. Specifically, BO assumes the performance objective to be expressed with a Bayesian model. Such a model carries a probabilistic prior that expresses our knowledge about the objective in probabilistic terms. If the Bayesian model carries perfect assumptions about the shape of the objective, theoretical guarantees of convergence to the global optimum can be provided. In other words, we are guaranteed to find the optimal controller parameters.

BO is a theoretically founded framework, with promising properties for learning

on real systems. However, questioning whether it is deployable and actually useful on real systems opens up a variety of research directions, which we discuss next.

First, we wonder whether it is possible to automate controller tuning of robots using BO. As an entry-level problem, we propose to automatically improve an existing controller that performs poorly. To this end, we need to accommodate the trial-and-error process on the real robot to the learning loop that BO proposes. Specifically, we need to (i) deploy a robust communication framework between BO and the robot, (ii) propose a performance criterion to quantify the quality of each tried controller parameters, and (iii) ensure that the learned optimal controller improves upon the initial one. We dedicate Chap. 3 to address this problem.

When using BO to automatically tune robot controllers, the objective performance is modeled using a Gaussian process (GP), which is a Bayesian model used to aid the optimization. However, prior to any data collection, the shape of the objective function is unknown and thus, we cannot be certain about what would be the best probabilistic prior to construct our GP. Having a poorly chosen prior can lead to poor learning performance. Thus, another interesting question is how to use informed priors to better leverage the acquired experimental data to be more *sample-efficient*.

Thereby, one way to increase sample-efficiency when using BO to learn robot controllers is to aid the controller learning using a dynamics model, e.g., running robot simulations inside a computer. Because simulations are faster than real experiments, they can be seen as a cheap source of information. Realizing many simulations can help to reduce the number of real experiments, thus increasing sample-efficiency. However, because the dynamics model is never perfect, we also need to collect real experimental data to mitigate the bias of the simulator. One way to combine simulation and real data is to use a multi-output Gaussian process that explicitly encodes a joint distribution between the two information sources. This model needs then to be interpreted by BO, which shall select controller parameters from the most informative, but cheaper information source, at each iteration. In Chap. 4 we address this problem.

Another way to increase sample-efficiency is using an informed prior GP to model the objective function. This can be done by building up a probabilistic prior with contextual information of the control problem at hand. In the case of a linear quadratic regulator (LQR), this involves exploiting the mathematical structure of the control problem and identifying how different control parameters vary the value of the quadratic performance objective. This problem is investigated in Chap. 5.

Finally, another possibility is to avoid controller failures while learning, as these can delay the search of the optimum and contribute to hardware wearing off. This

turns the controller tuning problem into a constrained optimization problem, where the constraints are black-box functions representing "failure/success". Although undesirable, failures are also informative about a behavior that should be avoided. Hence, an interesting problem setting arises by limiting the number of failures to a pre-established value. Then, the goal is to find the optimum by leveraging failures in the best possible way. We explore this problem setting in Chap. 6.
Generally speaking, all the problems proposed above attempt to answer the same research question:

> *Can we replace manual controller tuning with Bayesian optimization, and remain sample-efficient in robotics?*

Throughout this thesis, we investigate this research question, and also address the problem statements from above, in four different contributions. In the following, we briefly summarize them, and also present the outline of the thesis.

1.4 Thesis outline and contributions

The main contents of this thesis investigate the four problems proposed in Sec. 1.3 about use Bayesian optimization (BO) for learning controllers in robotics. Before discussing the aforementioned problems, we first provide a thorough description of the automatic controller tuning problem using BO in Chap. 2, together with a few essential mathematical tools (e.g., Gaussian processes) upon which the rest of the thesis is built. Then, we discuss the aforementioned ideas in four main parts (Chap. 3 to 6), presented in chronological order of their publication date. Finally, we conclude the thesis in Chap. 7 with a reflection about present and future impact of BO in robotics, among other inquiries.

In order to ease a quick inspection of the contents of the thesis, we briefly summarize below each one of the four core parts. We briefly state the problem we attempt to solve, the challenges involved and the obtained results on each part. For clarity, we also cite all the publications that arose from each project. In all the publications listed below, my responsibilities as first author were to (i) take the lead on the project, (ii) write the paper, (iii) do all the experiments and (iv) manage all the stages of the submission; all this with help and advice of the other co-authors.

Chapter 3: Automatic LQR tuning using Bayesian optimization

The chapter constitutes the first part of the thesis and presents a framework to automate the controller tuning process, otherwise conducted manually. To this end,

linear optimal control is combined with a specific BO algorithm. We demonstrate the framework on two challenging robotic platforms. Our results highlight the method's potential for automatic controller tuning in robotics. This framework constitutes the core part of the thesis, from which several important challenges arose. These challenges led to the research projects presented in the coming chapters.

The ideas presented in this chapter are largely based on our conference publication, which is cited below.

- Alonso Marco, Philipp Hennig, Jeannette Bohg, Stefan Schaal, and Sebastian Trimpe, "Automatic LQR tuning based on Gaussian process global optimization", In IEEE International Conference on Robotics and Automation (ICRA), pages 270–277, © 2016 IEEE.

The above publication extends results from previous work (Marco, 2015; Marco et al., 2015) and led to further unpublished, yet mature, results, which are presented in this thesis, in Sec. 3.5.

While working on this project, we acknowledged many caveats that BO presents when used to mitigate manual tuning. Specifically, BO is not as efficient as it could be due to many issues, such as lacking informative priors. Those issues opened many research problems, three of which are addressed on each of the subsequent chapters of this thesis. We summarize them next.

Chapter 4: Trading off simulations and physical experiments using BO

When working with highly complex real systems, such as humanoid robots, computer simulators are usually available. They are a model the dynamics of the system, typically used as a preliminary testbench. One possibility to increase sample-efficiency is to include the simulator as part of the optimization loop. Because simulations are also informative about controller performance, they shall help to reduce the necessary number of real experiments. However, simulations provide only biased information about the real performance. In Chap. 4, we propose a BO variant that explicitly models the trade-off between real robot experiments and simulated experiments. Our results, demonstrated on a cart-pole system, indicate that the proposed algorithm spares a great number of real experiments, while alternating between the simulator and the real system. The associated publication is cited below.

- Alonso Marco, Felix Berkenkamp, Philipp Hennig, Angela P. Schoellig, Andreas Krause, Stefan Schaal, Sebastian Trimpe, "Virtual vs. real: Trading off simulations and physical experiments in reinforcement learning with Bayesian optimization", In IEEE International Conference on Robotics and Automation (ICRA),

pages 1557-1563, © 2017 IEEE.

Chapter 5: On the design of LQR kernels

Another way to increase sample-efficiency is using informed priors. Standard (or non-informed) prior choices do not necessarily reflect the shape of the unknown performance objective, thus leading to poor fitting. However, the control problem at hand already imposes structure in the controller itself and also in the shape of the performance objective. If such structure is properly exploited and incorporated in the prior, fitting could be improved and real experiments would be better interpreted in the context of a more correct prior, thus reducing BO iterations. In Chap. 5, we propose a framework to design priors tailored to the specific structure of the control problem at hand. Simulated results demonstrate that good learning performance can be achieved with only a few BO iterations, compared to standard choices.

- Alonso Marco, Philipp Hennig, Stefan Schaal, Sebastian Trimpe, "On the Design of LQR Kernels for Efficient Controller Learning ", In IEEE Annual Conference on Decision and Control (CDC), pages 5193-5200, © 2017 IEEE.

Chapter 6: Controller learning under limited budget of failures

To further increase sample efficiency, the fourth part of the thesis addresses how to keep the number of unstable controllers low. Because unstable controllers tend to have risky repercussions in practice (e.g., hardware wearing off, reparation costs, etc.), we try to avoid them. However, not tolerating failures at all would imply that (i) only local exploration in a safe region is possible, as opposed to global exploration, and (ii) we need to initialize the learning routine with a stable controller, which is an impractical assumption in real robotics. Although generally undesirable, unstable controllers can also be leveraged once they have occurred: They are informative about uninteresting regions that should be avoided in further experiments. Hence, we propose a middle ground solution where collecting failures is not forbidden, but we limit them in number. We present a framework that addresses this problem in Chap. 6. Comparisons against state-of-the-art methods show that the proposed algorithm exploits better the budget of failures, hence estimating better the optimum.

- Alonso Marco, Alexander von Rohr, Dominik Baumann, José Miguel Hernández-Lobato, Sebastian Trimpe, "Excursion Search for Constrained Bayesian Optimization under a Limited Budget of Failures", *under review*.

In the following section we briefly review relevant existing work that connects with each one of the four projects summarized above.

1.5 Literature overview

The common goal to the four aforementioned projects is to fine-tune controller parameters of a robot using Bayesian optimization (BO). Usually, BO uses a Gaussian process (GP) (Rasmussen and Williams, 2006) as a non-parametric model capturing the knowledge about the unknown cost function. At every iteration, BO exploits all past data to infer the shape of the cost function. Furthermore, in the spirit of an active learning algorithm, it suggests the next evaluation in order to learn most about the location of the minimum.
Next, we first briefly detail related work to the BO framework. Then, we place each of the projects summarized above amongst existing literature.

1.5.1 Related work in Bayesian optimization (BO)

There exist an increasing number of BO algorithms based on Gaussian process (GP). Herein, we briefly mention some of the most popular algorithms, many of which have been used in thesis (either as a baseline for comparison, or as the main BO method). Readers interested in a thorough survey about recent BO methods are referred to (Shahriari et al., 2016).

Improvement-based methods attempt to collect evaluations likely to improve upon the best point observed so far. Amongst these methods, *expected improvement* (EI) (Jones et al., 1998; Mockus et al., 1978) selects evaluations that are expected to lie below the current best observation; *probability of improvement* (PI) (Kushner, 1964) chooses evaluations with the highest probability of lying below the current best observation.

Another set of algorithms steer the search in order to yield convergence guarantees. For example, *upper confidence bound* (GP-UCB) (Auer, 2003; Srinivas et al., 2010) captures the historically popular notion of "optimism in the face of uncertainty". It also proposes an explicit exploration vs. exploitation strategy, whose trade-off is regulated at each iteration in order guarantee convergence.

Information-based methods are known to outperform the aforementioned strategies and constitute another broad category. A key difference between these methods and the ones mentioned above is that the former select evaluations that yield increasingly low function values, while the later attempt to gain information about the optimum. The former is the right strategy in settings where the performance

of each individual experiment matters, e.g. the gait of a walking robot is improved online, but it is not allowed to fall. However, in a "prototyping" setting, where the sole use of experiments is to learn about a final good design, the numerical result of each experiment is less important than its information content.

The first information-based method, Entropy Search (ES) (Hennig and Schuler, 2012), inspired many other algorithms in the same direction, e.g., Predictive Entropy Search (PES) (Hernández-Lobato et al., 2014) and Max-Value Entropy Search (MES) (Wang and Jegelka, 2017). While PES proposes a computationally lighter implementation than ES, both focus on gathering information about the location of the minimum. On the contrary, MES targets gathering information about the minimum value, thus resulting a computationally cheaper acquisition function.

Many of these methods have been used in the context of automatic controller tuning of robotic platforms. For example in (Berkenkamp et al., 2016b; Calandra et al., 2016; Schreiter et al., 2015), in different flavors. While in Chap. 3 and 4 we focus on data efficiency by maximizing the information gain in each iteration, Berkenkamp et al. (2016b) and Schreiter et al. (2015) propose methods for safe exploration. Specifically, Berkenkamp et al. (2016b) optimize a state-feedback controller for quadrotor trajectory tracking using a safe BO algorithm (Sui et al., 2015), which builds on GP-UCB. Herein, we propose to parametrize the feedback gain as an LQR policy (Trimpe et al., 2014), whose optimal weights are learned using ES. This algorithm is extended in Chap. 3 and 4 to include information from different sources (such as simulation and physical experiments). Both methods have been successfully applied to learn pole balancing controllers. Calandra et al. (2016) compare PI, EI, and GP-UCB for learning the parameters of a discrete event controller for a walking robot.

1.5.2 Chapter 3: Automatic LQR tuning using Bayesian optimization

The automatic controller framework presented in this chapter is inspired from previous work (Trimpe et al., 2014), where a Linear Quadratic Regulator (LQR) is iteratively improved based on control performance observed in experiments. Therein, the controller parameters of the LQR design are adjusted using simultaneous perturbation stochastic approximation (SPSA) (Spall, 2003) as optimizer of the experimental cost. It obtains a very rough estimate of the cost function gradient from few cost evaluations, and then updates the parameters in its negative direction. While control performance could be improved in experiments on a balancing platform in (Trimpe et al., 2014), this approach does not exploit the available data as much as could

be done. Additionally, rather than exploring the space globally, it only finds local minima.

Automatic tuning of an LQR is also considered in (Grimble, 1984) and (Clarke et al., 1985), for example. In these references, the tuning typically happens by first identifying model parameters from data, and then computing a controller from the updated model. In contrast, we tune the controller gain directly thus bypassing the model identification step. Albeit we exploit a nominal model in the LQR design, this model is not updated during tuning and merely serves to pre-structure the controller parameters.

In contrast to (Trimpe et al., 2014), we propose the use of Entropy Search (ES) (Hennig and Schuler, 2012; Villemonteix et al., 2009), a recent algorithm for global Bayesian optimization (BO), as the minimizer for the LQR tuning problem. ES uses a GP as a non-parametric model capturing the knowledge about the unknown cost function. Thus, we expect ES to be more data-efficient than simple gradient-based approaches as in (Trimpe et al., 2014); that is, to yield better controllers with fewer experiments. For a literature review see Sec. 2.3. A similar approach has been proposed in (Berkenkamp et al., 2016b; Calandra et al., 2016; Metzen, 2015; Schreiter et al., 2015). In (Schreiter et al., 2015), the space of controller parameters is explored by selecting next evaluation points of maximum uncertainty (i.e. maximum variance of the GP). In contrast, ES uses a more sophisticated selection criterion: it selects next evaluation points where the expected information gain is maximal in order to learn most about the global minimum. A particular focus of the method in (Schreiter et al., 2015) is on safe exploration. For this purpose, an additional GP distinguishing safe and unsafe regions (e.g. corresponding to unstable controllers) is learned.

Safe learning is also the focus in (Berkenkamp et al., 2016b), where the BO algorithm for safe exploration by Sui et al. (2015) is used. This work restricts the exploration to controllers that incur a small cost with high probability. The method avoids unsafe controllers and finds the optimum within the safely reachable set of controllers. In contrast, ES explores globally and maximizes information gain in the entire parameter space, regardless of a potentially large costs incurred in an individual experiment.

In the context of dynamic walking, BO is used to learn gait parameters of planar biped robots (Antonova et al., 2016; Calandra et al., 2014; Rai et al., 2018). In (Calandra et al., 2016), the gait is achieved using a discrete event controller, and transitions are triggered based on sensor feedback and the learned parameters. Furthermore, BO has been used to learn whole-body controllers for a humanoid (Charbonneau et al., 2018; Yuan et al., 2019). In (Yeganegi et al., 2019), BO is used

to learn a trade-off between robustness and performance for the generation of center of mass trajectories.

Same as in Chap. 3, Metzen (2015) also uses ES for controller tuning, and extends this to contextual policies for different tasks. While Schreiter et al. (2015) and Metzen (2015) present simulation studies (balancing an inverted pendulum and robot ball throwing, respectively), other authors like Berkenkamp et al. (2016b) and Calandra et al. (2016) demonstrate their algorithms in hardware experiments (quadrotor and 4-DOF walking robot). To the authors' knowledge, (Berkenkamp et al., 2016b) and the work herein are the first to propose and experimentally demonstrate BO for direct tuning of continuous state-feedback controllers on a real robotic platform.

1.5.3 Chapter 4: Trading off simulations and physical experiments using BO

In this chapter, we include a prior model information from a simulator in the learning loop in order to spare evaluations on the real system. In the context of reinforcement learning, this idea has been considered before. A typical approach is two-stage learning, where algorithms are trained for a certain amount of time in simulation in order to warm-start the learning on the real robot (Kober et al., 2013). For example, Cutler and How (2015) report performance improvements when using model information from simulation as a prior for real experiments. Transfer learning is a similar approach that aims to generalize between different tasks, rather than from a simulated model to the real system (Taylor and Stone, 2009). The work in (Cutler et al., 2015) learns an optimal policy and value function of a finite Markov decision process based on models with different accuracies. They rely on hierarchical models and switch to higher accuracy models once a threshold accuracy has been reached at a lower level. A commonly used reinforcement learning method is policy gradients (Peters and Schaal, 2006), where policy parameters are improved locally along the gradient. In this setting, simulation knowledge can be used to estimate the gradient of real experiments (Abbeel et al., 2006). However, policy gradient methods only converge to locally optimal parameters. None of the above methods explicitly considers the cost of experiments on a robot. In this chapter, we actively trade off the different costs and information gains associated with simulation and real experiments and obtain globally optimal parameter estimates.

In the context of Bayesian optimization, controller parameters can be determined globally, as in gait optimization for legged robots (Lizotte et al., 2007) and controller optimization for a snake-like robot (Tesch et al., 2011). In Chap. 3, the

controller parameters of a linear state-feedback controller were optimized using the LQR framework as a low-dimensional representation of controller policies, while in (Abdelrahman et al., 2016) the control policy itself was defined by Bayesian optimization with a specifically chosen kernel. Safety constraints on the robot during the optimization process were considered in (Berkenkamp et al., 2016a). A comparison of different Bayesian and non-Bayesian global optimization methods can be found in (Calandra et al., 2014). All the previous methods use Bayesian optimization directly on the real system. In contrast, we consider an extension of Bayesian optimization that can extract additional information from a simulator and speed up the optimization process.

The methodology herein is related to multi-task Bayesian optimization, where one aims to transfer knowledge about two related tasks (Krause and Ong, 2011; Swersky et al., 2013). A GP model with multiple information sources was first considered in (Kennedy and O'Hagan, 2000). Since then, optimization with multiple information sources has been considered under strict requirements, such as models forming a hierarchy of increasing accuracy and without considering different costs (Forrester et al., 2007; Kandasamy et al., 2019). More recently, Poloczek et al. (2017) used a myopic policy, called the "knowledge gradient" by Frazier et al. (2009), in order determine, which parameters to evaluate.

1.5.4 Chapter 5: On the design of LQR kernels

In this chapter, we propose using informed priors in order to increase sample-efficiency. We assume the objective cost is modeled using a GP, which encodes knowledge (e.g., lengthscale, variance) about the unknown cost function through a kernel function. The design of customized kernels for GP regression has been considered before in the context of control and robotics for related problems. A kernel for bipedal locomotion, which captures typical gait characteristics, is proposed in (Antonova et al., 2016). In (Medina et al., 2016), an impedance-based model is incorporated as prior knowledge for improved predictions in human-robot interaction. For the problem of maximizing power generation in photovoltaic power stations, the authors in (Abdelrahman et al., 2016) incorporate explicit basis functions about known power curves in the kernel. In Chap. 4, a kernel is designed to model information from simulation and physical experiments, in order to leverage both sources of information for RL.

However, none of the above references considers the problem of incorporating the structure of the LQR problem to improve data-efficiency in learning control with BO.

1.5.5 Chapter 6: Controller learning under limited budget of failures

In this chapter, we consider the controller tuning problem subject to avoiding failures with high probability, under a given budget of failures. This can be seen as a optimization problem subject to multiple black-box unknown constraints. To address such constrained optimization problem, Bayesian optimization under unknown constraints (BOC) has been recently proposed. Within this method, there exist two opposite views in regard of how failures are handled. On one extreme, zero-failures strategies (Berkenkamp et al., 2016b; Sui et al., 2015) are needed in safety-critical applications, where failures are not allowed. Such strategies avoid failures by conservatively expanding an initially given safe area, and never exploring beyond the learned safety boundaries. On the other extreme, standard BOC strategies (Gardner et al., 2014; Gelbart et al., 2014; Gramacy and Lee, 2011; Hernández-Lobato et al., 2016; Picheny, 2014; Schonlau et al., 1998) are able to explore outside the initially given safe area, and target alternative, more promising regions. However, as they do not have a limit on the number of incurred failures, they can fail many times. The type of settings we have in mind lie in between these two extremes: We picture applications where paying the price of a pre-fixed budget of failures is affordable in exchange of finding a better optimum than conservative approaches. When learning robot controllers, allowing a limited budget of failures, with the consequent hardware-related issues, may help to find alternative, better optima.

Chapter 2

Preliminaries

In this chapter, we introduce the core problem statement of this thesis. This is the controller tuning problem, described in Sec. 2.1, which will be recurrently referenced throughout the thesis. In addition, we introduce several key concepts needed to understand how the controller tuning problem is solved, such as Gaussian processes (GPs) in Sec. 2.2 and the Bayesian optimization (BO) framework in Sec. 2.3.

2.1 Controller tuning problem

Given an existing model of a real system, controllers tailored to be optimal for such model will most likely be suboptimal on the real system. When the model mismatch, also known as *the reality gap*, is not large, small adjustments in the controller parametrization might help to recover optimality. In this section, we explain how to explicitly formulate the aforementioned fine-tuning problem as an optimization problem.

Let a system be defined by $s_{k+1} = h(s_k, u_k, w_k)$, where $s_k \in \mathbb{R}^S$ is the system state, $u_k \in \mathbb{R}^U$ is the control signal, $w_k \in \mathbb{R}^S$ is process noise, at time step k, and h are the unknown dynamics of the system. We assume the system can be driven from an initial state s_0 to a terminal state s_M, describing a trajectory $\tau_M = \{s_k, u_k\}_{k=0}^{M}$, using a state-feedback controller $u_k = \pi(s_k)$. The associated cost of such trajectory is given by $f(\pi) = \sum_{k=0}^{M} l(s_k, \pi(s_k))$, with a user-defined cost function $l : \mathbb{R}^S \times \mathbb{R}^U \to \mathbb{R}$. Although the true system dynamics h are unknown, a dynamics model $\hat{h}$ might be available, in some cases. As it is widely known from the control literature, some specific classes of models and cost choices l yield tractable optimal control solutions.

At an abstract level, an optimal controller can be computed in closed-form by solving

$$\begin{aligned} \pi^* = \underset{\pi}{\text{argmin}} \quad & \textstyle\sum_{k=0}^{M} l(s_k, u_k) \\ \text{s.t.} \quad & s_{k+1} = \hat{h}(s_k, u_k, w_k) \\ & u_k = \pi(s_k). \end{aligned} \tag{2.1}$$

For example, when the model $\hat{h}$ is linear and the cost l is quadratic, the optimal controller π^* is the linear quadratic regulator (LQR) (Anderson and Moore, 1990). Since the model $\hat{h}$ does not fully resemble the true unknown dynamics h, the controller π that is optimal for the known model $\hat{h}$, will be suboptimal for the true unknown system dynamics h.

Finding a closed-form (exact or approximate) solution to (2.1) often depends on the assumptions we put on the dynamics model $\hat{h}$. For example, if $\hat{h}$ is given as a neural network (Chua et al., 2018), classical techniques from optimal control may not be applied (Anderson and Moore, 1990). In addition, acquiring $\hat{h}$ is not always possible, desirable, or necessary. For instance, π might be parametrized independently of $\hat{h}$ using a PID controller, tuned directly on the real system. In the best case, $\hat{h}$ is analytically given; however, the model mismatch between $\hat{h}$ and h could be too large, causing that π^* exhibits poor performance when applied to the real system h.

Generally speaking, for the controller tuning problem presented herein, we do not assume that a dynamics model $\hat{h}$ exists. In contrast, we assume the existence of a feedback controller $u_k = \pi(s_k)$, given in some parameterizable form. Hence, the existence of a model is optional, and, if given, optimal control frameworks, such as (2.1), could be leveraged.

In order to improve performance of a controller π on the real system h, we parametrize it $\pi_x(s_k) = \pi(s_k; x)$ with parameters $x \in \mathcal{X} \subseteq \mathbb{R}^D$. Then, the *controller tuning problem* consists of adjusting/tuning π_x until the optimal parameters x_* are found. For this, one can simply follow a trial-and-error process, by collecting experimental data on the true system h for each configuration x. This constitutes the usual "manual tuning" process.

However, as the dimensionality increases (i.e., $D > 3$), manual tuning starts becoming non-intuitive. Brute force approaches, such as grid search, require a prohibitive number of real experiments due to the curse of dimensionality, which could cause the hardware to wear off. Alternatively, one can formulate the tuning problem as a black-box optimization problem and attempt to solve it using available algorithms for global optimization. Each parametrization x of the controller π_x steers the true system with a different trajectory, which leads to a different cost $f(\pi_x)$ value. This induces an objective $f(\pi_x) : \mathcal{X} \to \mathbb{R}$, which we write as $f(x)$ to simplify

notation. Then, the tuning problem is to find the parametrization x_* that minimizes the trajectory cost

$$x_* = \underset{x \in \mathcal{X}}{\operatorname{argmin}}\, f(x) \tag{2.2}$$

on the true dynamics h.

2.2 Gaussian process (GP)

Herein, we briefly describe the core elements of a GP. For more details about GPs, we refer the reader to (Rasmussen and Williams, 2006). We replicate below the definition of a GP from (Rasmussen and Williams, 2006, Def. 2.1).

Definition 1 *A Gaussian process is a collection of random variables, any finite number of which have a joint Gaussian distribution.*

In particular, the scalar function $f : \mathcal{X} \to \mathbb{R}$ is modeled with a GP if any finite collection of queries $[f(x_1), f(x_2), \ldots, f(x_t)]$ $\forall x_i \in \mathcal{X}$ have a joint Gaussian distribution. Throughout this thesis, the unknown objective f is modeled with a GP, $f \sim \mathcal{GP}\left(m(x), k(x, x')\right)$, with covariance function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ and prior mean function $m : \mathcal{X} \to \mathbb{R}$. The mean function is generally a parametric function, used to inject prior structure in the GP model. A brief discussion about covariance functions in the context of learning for real systems is later presented in Sec. 2.2.1.

We assume the function f cannot be queried directly, but only through noisy observations

$$y(x) = f(x) + \varepsilon, \quad \varepsilon \sim \mathcal{N}\left(0, \sigma^2\right). \tag{2.3}$$

After having collected t observations of the objective $\mathcal{D}_t = \{X_t, Y_t\} = \{x_1, \ldots, x_t, y_1, \ldots, y_t\}$, its predictive distribution at a location x is given by a univariate Gaussian distribution

$$p(f|\mathcal{D}_t, x) = \mathcal{N}(f(x); \mu(x|\mathcal{D}_t), \sigma^2(x|\mathcal{D}_t)), \tag{2.4}$$

with predictive mean $\mu(x|\mathcal{D}_t) = m(x) + k_t^\top(x)[K_t + \sigma^2 I]^{-1}[Y_t - M_t]$, where the entries of vector $k_t(x)$ are $[k_t(x)]_i = k(x_i, x)$, the entries of the Gram matrix K_t are $[K_t]_{i,j} = k(x_i, x_j)$, the entries of the vector of observations Y_t are $[Y_t]_i = y_i$, and the entries of the vector of prior mean values M_t are $[M_t]_i = m(x_i)$. The predictive variance is given by $\sigma^2(x|\mathcal{D}_t) = k(x, x) - k_t^\top(x)[K_t + \sigma^2 I]^{-1} k_t(x)$. In the remainder of the thesis, we drop the dependency on the current data set $\mathcal{D}_t$ and write $\mu(x)$, $\sigma(x)$ to refer to $\mu(x|\mathcal{D}_t)$, $\sigma(x|\mathcal{D}_t)$, respectively.

Fig. 2.1 (top) exemplifies a Gaussian process posterior in a one dimensional regression problem, conditioned on five observed data points, where the prior mean function m is assumed to be zero.

2.2.1 Covariance functions for real systems

The covariance function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is a core element of a Gaussian process, as it encodes (i) prior knowledge about the objective f, such as smoothness, characteristic length-scales, and signal variance, and (ii) similarity between two function queries $f(x)$ and $f(x')$. The choice of the covariance function usually has an impact on the quality of the regression. A covariance function k can be defined as a positive semidefinite *kernel*[1] .

There are many popular choices for kernels, however, it is non trivial to predict which kernel will perform better for a given regression task. There exist many popular choices for kernels in the context of GP regression. However, given a specific regression task, selecting the kernel that will perform best *prior to data collection* remains an open question.

In practical applications, such as controller tuning in robotics, where data collection is scarce, the kernel choice is important to achieve good learning performance, yet for deciding on it we are left only with intuition. In recent work, the kernel of a GP model itself is estimated from data (Rai et al., 2018; Wang et al., 2018b). However, such kernel is either a neural network or a composition of radial basis functions, respectively. In both cases, prior structure has been injected into the problem to learn the kernels. We devote Chap. 5 to specifically address the kernel design problem with a novel approach that allows to construct a kernel tailored to a specific control problem.

I enumerate next two of the kernels that will be referenced the most throughout the thesis. An introduction to the properties and usages of kernels in the context of GPs can be found in (Rasmussen and Williams, 2006, Sec. 4.2.1).

Squared exponential kernel

The squared exponential (SE) kernel, also known as the Gaussian kernel, and more appropriately named "exponentiated quadratic", encodes functions that admit an infinite number of derivatives, i.e., smooth functions. It is defined as

$$k_{\mathrm{SE}}(x, x') = \sigma_{\mathrm{S}}^2 \exp\left[-\frac{1}{2}(x - x')^\top S(x - x')\right], \tag{2.5}$$

where the diagonal matrix $S = \mathrm{diag}\left[\lambda_1^{-2}, \lambda_2^{-2}, \ldots, \lambda_D^{-2}\right]$ carries information about the *lengthscale* λ_i of the function on each input dimension. The *signal variance* σ_{S}^2 can be connected to the amount of uncertainty about the function.

[1] While a kernel does not need to be positive definite, a covariance function does (Rasmussen and Williams, 2006, Sec. 4.1). Throughout this thesis, we use the word "kernel" to refer to the covariance function of a GP.

The smoothness property is rather limiting when it comes to robotic applications. For example, such kernels are not appropriate to model dynamical systems that exhibit discontinuities (Rasmussen and Williams, 2006, Sec. 5.4.3), nor a good choice to model sparse rewards. In those cases, a popular alternative is the Matérn kernel.

Matérn kernel

Contrary to the SE kernel, the Matérn family of kernels are p-times mean-square differentiable (Rasmussen and Williams, 2006, Sec. 4.1.1). In case of $p = 1$, it is given by

$$k_{\text{mat}}(x, x') = \left(1 + \frac{\sqrt{3}d}{\lambda}\right) \exp\left(-\frac{\sqrt{3}d}{\lambda}\right), \tag{2.6}$$

where $d = \|x - x'\|_2$. This makes it suitable for encoding non-smooth functions and thus, it is usually preferred for modeling real systems.

An interactive GP tool that allows to modify the lengthscale λ and variance σ_{S}^2 of a one-dimensional GP can be found at https://github.com/alonrot/interactiveGP. In the following, we discuss how to estimate GP hyperparameters from data.

2.2.2 Hyperparameter optimization of GPs

GP models are usually considered a *non-parametric* probabilistic regression tool because no assumption is made on the parametric structure of the function. However, the kernel, the mean and the likelihood functions contain free parameters, which are determined by the user. Alternatively, the non-parametric consideration broadens to include such hyperparameters, which are estimated from data, rather than fixed.

There exist numerous methods to estimate the hyperparameters, i.e., to fit the GP model to the collected data. Herein, we enumerate a few of which have been explored throughout this thesis.

Sampling-based methods attempt to sample from the posterior distribution of the hyperparameters (usually intractable), given the data. For this, Elliptical Slice Sampling is easy to implement and no tuning is required (Murray et al., 2010). However, the prior distribution that models the hyperparameters, also known as *hyperprior*, is required to be Gaussian. The No-U-Turn Sampler (NUTS) (Hoffman and Gelman, 2014), a variant of Hamiltonian Monte Carlo, requires little or no tuning and makes no explicit assumption about the hyperprior.

From a rather different perspective, *variational inference* (VI) gathers a broad set of algorithms that approximate the hyperparameters posterior to a variational family. In the context of GPs, GP-LVM (Titsias and Lawrence, 2010) was proposed

to estimate both, the variational parameters and the model hyperparameters, as typically done in sparse GPs models. More generally, ADVI (Kucukelbir et al., 2017) has been proposed as a practical algorithm for variational inference, where the user only needs to provide the probabilistic model and the dataset.

Finally, *maximum marginal likelihood* (type-II ML), also known as *maximum a posteriori* (MAP), returns a point estimate of the optimal hyperparameters. The marginal likelihood can be analytically computed for a GP when the likelihood function is a Gaussian density, as in (2.3). Given a collection of t observations $\mathcal{D}_t = \{Y_t, X_t\}$, the marginal likelihood (also known as *evidence*) is given in its logarithmic form as (Rasmussen and Williams, 2006, Eq. (2.30))

$$\log p(\mathcal{D}_t|x_t) = -\frac{1}{2}Y_t^\top \left[K_t + \sigma^2 I\right]^{-1} Y_t - \frac{1}{2}\log|K_t + \sigma^2 I| - \frac{t}{2}\log 2\pi, \tag{2.7}$$

where K_t and σ^2 have been introduced along with (2.4). Throughout this thesis, we have mostly used MAP with appropriate hyperpriors for the hyperparameters, e.g., Chap. 3 and 6.

2.3 Automatic controller tuning using Bayesian optimization

In this section, we explain the Bayesian optimization (BO) framework to automatically tune controller parameters, which amounts to solving the black-box unconstrained expensive-to-evaluate optimization problem (2.2). Alg. 1 summarizes in pseudocode the usage of BO for automatic controller tuning. The BO framework presented herein will be extensively used in the coming parts of the thesis (Chap. 3 to 6).

BO denotes a class of algorithms for black-box global optimization problems in which data collection is expensive (Kushner, 1964) and thus, only few evaluations are possible. This setting is no different from the automatic controller tuning problem in robotics Sec. 2.1, where an expensive-to-evaluate black-box function is optimized (2.2). Therein, each query $f(x_t)$ involves doing a robot experiment with controller parametrization x_t. Such experiments carry associated costs (e.g., time consuming, human effort), which make them expensive.

There exist many different BO strategies to steer the global optimization. Whereas the tuning problem (2.2) is agnostic to the selected BO method, its outcome (e.g., incurred number of experiments before convergence) may differ from one another.

In all cases, BO assumes the objective is modeled as a GP (Sec. 2.2), and uses such model to iteratively select next function evaluations in order to solve (2.2)

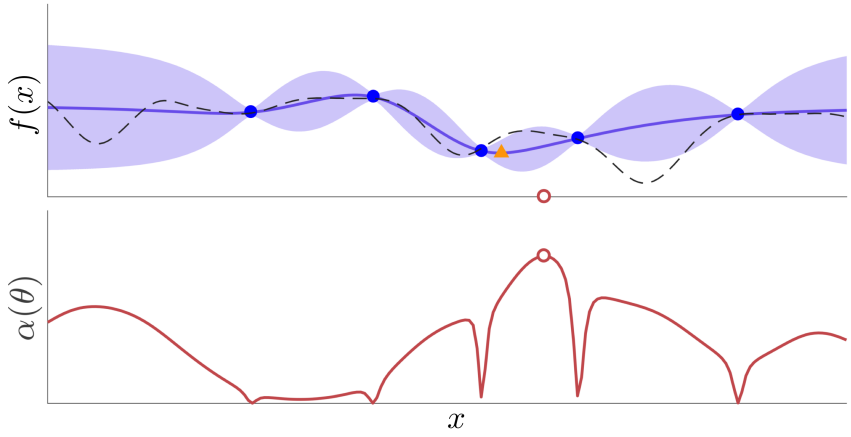


Figure 2.1: Intermediate stage of a BO algorithm, before finding the true minimum. Top: GP posterior conditioned on observed data. The unknown function (dashed line) is approximated by the GP, which is characterized by the predictive mean function (solid line) and the predictive variance (colored surface). The latter is represented with ± two standard deviations. Thus, unknown functions are expected to be contained within the surface limits with a 95% confidence. The current estimate of the global minimum (triangle) does not coincide with the true minimum yet. Bottom: Acquisition function α computed with entropy search (ES). The next parameters x_{t+1} are chosen at the maximum of α (hollow red dot). © 2019 IEEE.

efficiently. Next, we discuss the main steps taken generally taken by Bayesian optimization algorithms at each iteration.

At iteration t, BO decides which controller x_{t+1} shall be queried on the objective function f at the next iteration $t+1$. The decision-making is driven by an *acquisition function* $\alpha : \mathcal{X} \to \mathbb{R}$, whose maximizer is the location of the next query:

$$x_{t+1} = \operatorname*{argmax}_{x \in \mathcal{X}} \alpha(x). \tag{2.8}$$

The value $\alpha(x)$ is computed using the predictive distribution $p(f|\mathcal{D}_t, x)$, given in (2.4). Hence, each decision at iteration t is made taking the collected data $\mathcal{D}_t$ into account. Although the auxiliary optimization problem (2.8) is typically non-convex, querying α is usually computationally cheap. Thus, it can be solved using gradient-free or gradient-based local optimization methods with random restarts. In Fig. 2.1 (bottom), we see the acquisition function computed using the posterior GP model conditioned on the observed data.

The next parameters x_{t+1}, selected at the maximum of α, are used to perform a new experiment. The acquired observation $y_{t+1} = y(x_{t+1})$ is added to the existing data set $\mathcal{D}_t \leftarrow \mathcal{D}_t \cup \{y_{t+1}, x_{t+1}\}$ and α is again maximized in the next iteration. This procedure is repeated until a pre-defined stopping criterion (e.g., observations bounded below a pre-defined threshold) is met. Through this iterative procedure,

Algorithm 1 Learning control with Bayesian optimization

1: Specify objective function f with finite horizon T (cf. Sec. 2.1)
2: Specify GP prior (mean m, kernel k)
3: Initialize: controller parameters x_1; data set $\mathcal{D} = \emptyset$
4: **for** $t = 1$ to T **do**
5: perform closed-loop experiment with controller x_t
6: compute y_t from experimental data $\{s_k, u_k\}_{k=0}^{M}$
7: add evaluation $\{x_t, y_t\}$ to data set $\mathcal{D}_i$
8: Compute predictive distribution (2.4)
9: [Optional:] Optimize hyperparameters (e.g., maximize (2.7))
10: Compute next controller x_{t+1} via (2.8)
11: **end for**
12: Determine "best guess" x^{BG} for the controller parameters, e.g., minimum of posterior mean (2.9)
13: **return** x^{BG}

the framework is expected to explore relevant regions of the cost, infer the shape of the cost function, and eventually yield the global minimum within $\mathcal{X}$.

At the end of the algorithm, an estimate for the global minimum x_* is reported as the minimizer of the predictive mean of the GP

$$x_* = \operatorname*{argmin}_{x \in \mathcal{X}} \mu(x), \tag{2.9}$$

where $\mu(x)$ is the mean of the predictive distribution (2.4). Since $\mu(x)$ is cheap to evaluate and its gradients are given analytically, (2.9) can be approximately solved by using local methods with sufficient random restarts, same as as for (2.8).

2.3.1 A note on Entropy Search

In this thesis, we have used Entropy Search (ES) (Hennig and Schuler, 2012) in two projects: In Chap. 3, ES is used for automatic controller tuning, while in Chap. 4, it is extended to a multi-fidelity setting.

ES explicitly approximates a probability distribution about the location of the minimum at each iteration, and computes its entropy, i.e., information about the minimum. Then, the next location is selected where the expected information increment is maximal, i.e., where we expect to gain most information about the minimum. These procedures can be divided in two main steps, which we briefly summarize next on an abstract level; for details, we refer to the original paper (Hennig and Schuler, 2012). Pseudocode for ES can be found in Alg. 2. An alternative

version to the original ES code[2], which provides a more friendly user interface, plotting tools, and corrects bugs is available in https://github.com/alonrot/userES. A reimplementation of the original algorithm in c++, which increases computational speed by about ten times with respect to the original code, can be found at https://github.com/alonrot/EntropySearchCpp.

Distribution about the location of the minimum

ES models the knowledge about the location of the global minimum with a probability density $p_{\min}$, which quantifies the likelihood of the spatial distribution of the minimum

$$p_{\min}(x) = p(x = \underset{\tilde{x}\in\mathcal{X}}{\operatorname{argmin}}\, f(\tilde{x})). \tag{2.10}$$

This probability density (2.10) is analytically intractable and needs to be approximated to a discrete distribution $q_{\min}$, defined over a discretized version of the compact domain $\mathcal{X}$. ES discretizes the domain $\mathcal{X}$ as an irregular grid, which puts higher resolution in regions more likely to contain the minimum.

Acquisition function

In order to estimate the information that ES has gathered about the minimum location, the distribution $q_{\min}$ is compared to the uniform distribution u by means of the Kullback-Leibler divergence (relative entropy)

$$H_{q_{\min}} = D_{\mathrm{KL}}(q_{\min}||u). \tag{2.11}$$

The intuition behind this choice is that the uniform distribution contains no information about the location of the minimum ($H_{q_{\min}} = 0$), while a Dirac delta distribution centered at the location of the true minimum would maximize (2.11). Therefore, $H_{q_{\min}} \geq 0$ is used as a measure of how much we know about the minimum location.

We are not interested in $H_{q_{\min}}$ per se, but in how much this information would increase if we did an experiment on a new location x. Because we cannot know the exact outcome of this experiment beforehand $y(x)$, we have to commit to its expected value, which is feasible using the GP model. We define the *expected change in entropy* at an unobserved location x as

$$\alpha(x) = \mathbb{E}_{y(x)}\left[H_{\tilde{q}_{\min}(x,y(x))}\right] - H_{q_{\min}} = \mathbb{E}_{y(x)}\left[\Delta H(x)\right] \tag{2.12}$$

where $\tilde{q}_{\min}(x, y(x))$ depends implicitly on a new hypothetical observation $(x, y(x))$, and represents a new distribution about the minimum (different from $q_{\min}$) if we had included $(x, y(x))$ into the current dataset.

[2]https://github.com/ProbabilisticNumerics/entropy-search

Algorithm 2 As its inputs, EntropySearch takes the type of covariance function k, the likelihood l, a fixed number of evaluations T, and (possibly) existing data points $\{x_0, y_0\}$. Instead of stopping after T iterations, alternative stopping criteria can be used.

initialize x_1
$y_1 \leftarrow \text{CostEvaluation}(x_1)$ ▷ Cost evaluation
$\{x_1, y_1\} \leftarrow \{x_0, y_0\} \cup \{x_1, y_1\}$
procedure EntropySearch(k,l,T,$\{x_1, y_1\}$)
  for $t = 1$ to $T - 1$ **do**
    $[\bar{\mu}, \bar{k}] \leftarrow \text{GP}(k, l, \{x_t, y_t\})$ ▷ GP posterior
    $q_{\min} \leftarrow \text{approx\_qmin}(\bar{\mu}, \bar{k})$ ▷ Approximate $p_{\min}$
    $x_{t+1} \leftarrow \operatorname{argmax}_{x \in \mathcal{X}} \mathbb{E}_{y(x)} [\Delta H(x)]$ ▷ Next location to evaluate at
    $y_{t+1} \leftarrow \text{CostEvaluation}(x_{t+1})$ ▷ Cost evaluation
    $\{x_{t+1}, y_{t+1}\} \leftarrow \{x_t, y_t\} \cup \{x_{t+1}, y_{t+1}\}$
    $x^{\text{BG}} \leftarrow \operatorname{argmin}_{x \in \mathcal{X}} \mu(x)$ ▷ Update current "best guess"
  end for
  return x^{BG}
end procedure

Since the distribution over the minimum $\tilde{q}_{\min}(x, y(x))$ adds a hypothetical new point to the set of collected observations, it contains a larger amount of information than $q_{\min}$, which implies $\alpha(x) \geq 0$. A more detailed definition of (2.12) can be found in (Hennig and Schuler, 2012, Sec. 2.6). The acquisition function in Fig. 2.1 corresponds to an intermediate stage of ES, where the next point is selected by maximizing it (cf. (2.8)).

Chapter 3

Automatic LQR tuning using Bayesian optimization

The ideas presented in chapter are mainly based on the conference publication (Marco et al., 2016). Herein, we propose a framework to solve the automatic controller tuning problem (cf. Sec. 2.1) using Bayesian optimization (BO) (cf. Sec. 2.3). Specifically, we propose a linear optimal control problem, where an initial set of controller gains is automatically improved according to a pre-defined performance objective evaluated from experimental data. The underlying BO algorithm is Entropy Search (ES) (see Sec. 2.3.1 for quick summary), which represents the latent objective as a Gaussian process (GP) (cf. Sec. 2.2) and collects datapoints at locations that are most informative about the minimum. We demonstrate the proposed framework with two robotic platforms: A seven-degree-of-freedom robot arm balancing an inverted pole, and a humanoid robot performing a squatting task. Our results in two-, four- and six-dimensional tuning problems highlight the method's potential for automatic controller tuning on robotic platforms.

This chapter is structured as follows[1]. We introduce the problem and state the contributions in Sec. 3.1. The LQR tuning problem is described in Sec. 3.2. The use of ES for automating the tuning is outlined in Sec. 3.3. The experimental results obtained with the robot arm are presented in Sec. 3.4, and with the humanoid robot in Sec. 3.5. We discuss additional contributions closely related to the work presented herein in Sec. 3.6. Finally, the chapter concludes in Sec. 3.7 with a discussion about the caveats of the proposed method and a discussion about future applications.

[1]For an extensive literature review see Sec. 1.5.2

3.1 Introduction

Designing controllers for balancing systems such as in (Trimpe and D'Andrea, 2012) or (Mason et al., 2014) are typical examples of scenarios, where manual tuning or grid search, can be highly time-consuming. Often, one can without much effort obtain a rough linear model of the system dynamics around an equilibrium configuration, for example, from first principles modeling. Given the linear model, it is then relatively straightforward to compute a stabilizing controller, for instance, using optimal control. When testing this nominal controller on the physical plant, however, one may find the balancing performance unsatisfactory, e.g. due to unmodeled dynamics, parametric uncertainties of the linear model, sensor noise, or imprecise actuation. Thus, fine-tuning the controller gains in experiments on the real system is desirable in order to partly mitigate these effects and obtain improved balancing performance.

We propose an automatic tuning routine where a limited budget of experimental evaluations is allowed (e.g. due to limited experimental time on the plant, or costly experiments). The automatic tuning shall globally explore a given range of controllers and return the best known controller after a fixed number of experiments. During exploration, we assume that it is acceptable for the controller to fail, for example, because other safety mechanisms are in place (Akametalu et al., 2014), or it is uncritical to stop an experiment when reaching safety limits (as is the case in experiment considered herein).

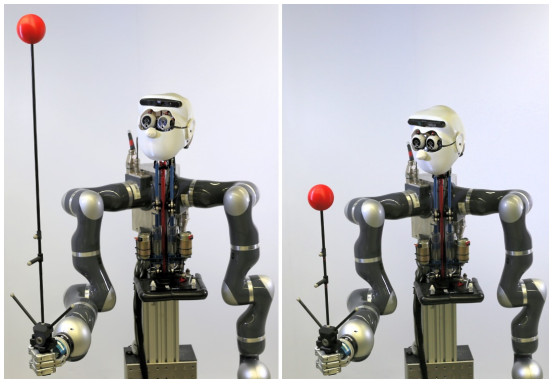


Figure 3.1: The humanoid robot Apollo learns to balance poles of different lengths using the automatic controller tuning framework proposed herein.

The core contribution of this chapter is the development of an automatic controller tuning framework combining Entropy Search (ES) (Hennig and Schuler, 2012), a data-efficient Bayesian optimization (BO) algorithm, with LQR tuning (Trimpe et al., 2014). While ES has been applied to numerical optimization problems before, this work is the first to use it for controller tuning on a complex robotic platform. The effectiveness of the proposed auto-tuning method is demonstrated in experiments of a humanoid robot balancing a pole (see Fig. 3.1). We present successful auto-tuning experiments for parameter spaces of different dimensions (2D and

4D), as well as for initialization with relatively good, but also poor initial controllers.

While early results of this approach are presented in the workshop paper (Marco et al., 2015), the presentation herein is more elaborate and new experimental results are included.

3.2 LQR tuning problem

This section is largely based on the controller tuning problem formulated in Sec. 2.1. Therein, the formulation is at a higher level of abstraction than here, where the formulation is made specific to the LQR tuning problem, as in (Trimpe et al., 2014).

3.2.1 Control design problem

We consider a system that follows a discrete-time non-linear dynamic model

$$s_{k+1} = h(s_k, u_k, w_k) \tag{3.1}$$

with system states $s_k \in \mathbb{R}^S$, control input $u_k \in \mathbb{R}^U$, and zero-mean process noise $w_k \in \mathbb{R}^S$ at time instant k. We assume that (3.1) has an equilibrium at $s_k = 0$, $u_k = 0$ and $w_k = 0$, which we want to keep the system at. We also assume that s_k can be measured and, if not, an appropriate state estimator is used.

For regulation problems such as balancing about an equilibrium, a linear model is often sufficient for control design. Thus, we consider a scenario, where a linear model

$$\tilde{s}_{k+1} = A_{\mathrm{n}}\tilde{s}_k + B_{\mathrm{n}}u_k + w_k \tag{3.2}$$

is given as an approximation of the dynamics (3.1) about the equilibrium at zero. We refer to (3.2) as the *nominal model*, while (3.1) are the true system dynamics, which are unknown.

A common way to measure the performance of a control system is through a quadratic cost function such as

$$f = \lim_{M\to\infty} \frac{1}{M+1}\mathbb{E}\left[\sum_{k=0}^{M} s_k^\top Q s_k + u_k^\top R u_k\right] \tag{3.3}$$

with positive-definite weighting matrices Q and R, and $\mathbb{E}[\cdot]$ the expected value. The cost (3.3) captures a trade-off between control performance (keeping s_k small) and control effort (keeping u_k small).

Ideally, we would like to obtain a state feedback controller for the non-linear plant (3.1) that minimized (3.3). Yet, this non-linear control design problem is

intractable in general. Instead, a straightforward approach that yields a locally optimal solution is to compute the optimal controller minimizing (3.3) for the nominal model (3.2). This controller is given by the well-known Linear Quadratic Regulator (LQR) (Anderson and Moore, 1990, Sec. 2.4)

$$u_k = F s_k \tag{3.4}$$

whose static gain matrix F can readily be computed by solving the discrete-time infinite-horizon LQR problem for the nominal model $(A_\mathrm{n}, B_\mathrm{n})$ and the weights (Q, R). For simplicity, we write

$$F = \mathrm{lqr}(A_\mathrm{n}, B_\mathrm{n}, Q, R). \tag{3.5}$$

If (3.2) perfectly captured the true system dynamics (3.1), then (3.5) would be the optimal controller for the problem at hand. However, in practice, there can be several reasons why the controller (3.5) is suboptimal: the true dynamics are nonlinear, the nominal linear model (3.2) involves parametric uncertainty, or the state is not perfectly measurable (e.g. noisy or incomplete state measurements). While still adhering to the controller structure (3.4), it is thus beneficial to fine tune the nominal design (the gain F) based on experimental data to partly compensate for these effects. This is the goal of the automatic tuning approach, which is detailed next.

3.2.2 LQR tuning problem

Following the approach in (Trimpe et al., 2014), we parametrize the controller gains F in (3.4) as

$$F(x) = \mathrm{lqr}(A_\mathrm{n}, B_\mathrm{n}, W_s(x), W_u(x)) \tag{3.6}$$

where $W_s(x)$ and $W_u(x)$ are *design weights* parametrized in $x \in \mathbb{R}^D$, which are to be varied in the automatic tuning procedure. For instance, $W_s(x)$ and $W_u(x)$ can be diagonal matrices with $x_j > 0$, $j = 1, \ldots, D$, as diagonal entries.

Parametrizing controllers in the LQR weights W_s and W_u as in (3.6), instead of varying the controller gains F directly, restricts the controller search space. This restriction is often desirable for practical reasons. First, we assume that the nominal model (albeit not perfect) represents the true dynamics reasonable well around an equilibrium. In this situation, one wants to avoid controllers that destabilize the nominal plant or have poor robustness properties, which is ensured by the LQR

design[2]. Second, further parametrizing W_s and W_u in x can be helpful to focus on most relevant parameters or to ease the optimization problem. While, for example, a restriction to diagonal weights W_s and W_u is common practice in LQR design (i.e. $n_s + n_u$ parameters), it is not clear how one would reduce the dimensionality of the gain matrix F ($n_s \times n_u$ entries) when tuning this directly. We expect this to be particularly relevant for high-dimensional problems, such as control of a full humanoid robot (Mason et al., 2014).

When varying x, different controller gains $F(x)$ are obtained. These will affect the system performance through (3.4), thus resulting in a different cost value from (3.3) in each experiment. To make the parameter dependence of (3.3) explicit, we write $f = f(x)$. The goal of the automatic LQR tuning is to vary the parameters x such as to minimize the cost (3.3).

Remark: The weights (Q, R) in (3.3) are referred to as *performance weights*. Note that, while the *design weights* $(W_s(x), W_u(x))$ in (3.6) change during the tuning procedure, the performance weights remain unchanged.

3.3 Automatic LQR Tuning

The above LQR tuning problem is summarized as the optimization problem

$$x_* = \underset{x \in \mathcal{X}}{\operatorname{argmin}}\, f(x) \tag{3.7}$$

where we restrict the search of parameters to a bounded domain $\mathcal{X} \subset \mathbb{R}^D$. The domain $\mathcal{X}$ typically represents a region around the nominal design, where performance improvements are to be expected or exploration is considered to be safe.

The shape of the cost function in (3.7) is unknown. Neither gradient information is available nor guarantees of convexity can be expected. Furthermore, $f(x)$ cannot be queried directly because (3.3) cannot be computed from experimental data in practice as it represents an infinite-horizon problem. As is also done in (Trimpe et al., 2014), we thus consider the approximate cost

$$y = \frac{1}{M+1}\left[\sum_{k=0}^{M} s_k^\top Q s_k + u_k^\top R u_k\right] \tag{3.8}$$

[2]According to classical results in control theory (Kalman, 1964) and (Kong et al., 2012), any stabilizing feedback controller (3.4) that yields a return difference greater one (in magnitude) can be obtained for some W_s and W_u as the solution to the LQR problem. The return difference is relevant in the analysis of feedback loops (Anderson and Moore, 1990), and its magnitude exceeding one means favorable robustness properties. Therefore, the LQR parameterization (3.6) only discards controllers that are undesirable because they destabilize the nominal plant, or have poor robustness properties.

Algorithm 3 Automatic LQR Tuning

initialize x^0 such that $W_s(x^0) = Q$, $W_u(x^0) = R$
Initialize Entropy Search (ES) (Alg. 2) with x^0 and set CostEvaluation() = CostEvaluationLQR()
Run ES

function CostEvaluationLQR(x)
    LQR design: $\bar{F} \leftarrow \mathrm{lqr}(A_\mathrm{n}, B_\mathrm{n}, W_s(x), W_u(x))$
    update control law (3.4) with $F = \bar{F}$
    perform experiment and record $\{s_k\}, \{u_k\}$
    Evaluate cost: $y \leftarrow \frac{1}{M+1}\left[\sum_{k=0}^{M} s_k^\top Q s_k + u_k^\top R u_k\right]$
    return y
end function

with a finite, yet long enough horizon T. The cost (3.8) can be considered a noisy evaluation of (3.3), i.e., $y(x) = f(x) + \varepsilon$, $\varepsilon \sim \mathcal{N}(0; \sigma^2)$. Such an evaluation is expensive as it involves conducting an experiment, which lasts few minutes in the considered balancing application.

The proposed method for automatic LQR tuning is obtained by using Entropy Search (ES) (cf. Sec. 2.3.1) to solve the optimization problem (3.7), which arises from the LQR tuning framework presented in Sec. 3.2. To this end, we model the objective (3.3) using a GP, $f \sim \mathcal{GP}\left(m(x), k_\mathrm{SE}(x, x')\right)$, where we assume a zero-mean prior function $m(x) \equiv 0$ and a squared exponential kernel k_SE. Details about the GP model used herein can be found in Sec. 2.2. We gather the hyperparameters of the GP in the set $\mathcal{H} = \{\lambda_1, \ldots, \lambda_D, \sigma_\mathrm{S}, \sigma\}$.

A reasonable initial choice of the parameters x is such that the design weights $(W_s(x), W_u(x))$ equal (Q, R), which encode the desired performance for the system (3.1). Thus, the obtained initial gain F is suboptimal because (3.2) are not the true dynamics. After N evaluations, ES reports the estimated best guess x^BG as the minimum expected cost (cf. (2.9)) when the maximum number of iterations has been reached. Pseudocode for the automatic LQR tuning method is given in Alg. 3.

A tutorial that highlights the insights of the proposed automatic LQR tuning method can be found in https://github.com/alonrot/aLQRtuning_tutorial.

3.4 Experimental results

In this section, we present auto-tuning experiments for learning to balance a pole as shown in Fig. 3.1. A video demonstration that illustrates the second experiment

described in Sec. 3.4.3 is available at https://youtu.be/TrGc4qp3pDM.

3.4.1 System description

We consider a one-dimensional balancing problem: a pole linked to a handle through a rotatory joint with one degree of freedom (DOF) is kept upright by controlling the acceleration of the end-effector of a seven DOF robot arm (Kuka lightweight robot). Figure 3.1 shows the setup for two poles of different length. The angle of the pole is tracked using an external motion capture system (Vicon).

The continuous-time dynamics of the balancing problem (similar to (Schaal, 1997)) are described by:

$$\begin{aligned} mr^2\ddot{\psi}(t) - mgr\sin\psi(t) + mr\cos\psi(t)u(t) + \xi\dot{\psi}(t) = 0 \\ \ddot{s}(t) = u(t) \end{aligned} \tag{3.9}$$

where $\psi(t)$ is the pole angle with respect to the gravity axis, $s(t)$ is the deviation of the end-effector from the zero position, and $u(t)$ is the end-effector acceleration.

Two poles with different lengths are used in the experiments. The center of mass of the short pole lies at $r \simeq 0.33$ m from the axis of the rotatory joint, its mass is $m \simeq 0.27$ kg, the friction coefficient is $\xi \simeq 0.012$ Nms, and the gravity constant is $g = 9.81$ m/s^2. For the long pole, we have $r \simeq 0.64$ m and $m \simeq 0.29$ kg.

A model (3.2) of the system is obtained by linearization of (3.9) about the equilibrium $\psi = 0$, $s = 0$ and discretization with a sampling time of 1 ms. Using the parameters of the short pole, we obtain its nominal model $(A_\mathrm{n}, B_\mathrm{n})$. The non-linear model (3.9) assumes that we can command a discretized end-effector acceleration u_k as control input to the system. In reality, this end-effector acceleration is realized through an appropriate tracking controller for the end-effector following a similar control structure as in (Righetti et al., 2014).

The estimated end-effector position s_k and velocity $\dot{s}_k$ are computed at a sampling rate of 1kHz from the robot's joint encoders using forward kinematics. The pole orientation is captured at 200 Hz by the motion capture system. From this data, we obtain estimates of pole angle ψ_k and angular velocity $\dot{\psi}_k$ through numerical differentiation and low-pass filtering (2nd-order Butterworth, 10 Hz cutoff). With this scheme, no model is required to obtain estimates of all states (in contrast to the Kalman filter used in (Marco et al., 2015)), and it can be used irrespective of which balancing pole is used. The complete state vector of (3.2) is given by $s_k = [\psi_k, \dot{\psi}_k, s_k, \dot{s}_k]^\mathrm{T}$.

When using a state-feedback controller (3.4) for balancing, biases in the angle measurement lead to a steady-state error in the end-effector position (cf. discussion

in (Trimpe and D'Andrea, 2012, p. 67) for a similar balancing problem). To compensate for such steady-state offsets, the state feedback controller (3.4) is augmented with an integrator on the end-effector position, which is a standard way to achieve zero steady-state error (see e.g. (Åström and Murray, 2008, Sec. 6.4)). That is, we implement the control law $u_k = Fs_k + F_z z_k$ instead of (3.4), where z_k is the integrator state.

Although F_z can readily be included in the LQR formulation (3.6) and tuned alongside the other gains (as was done by Marco et al. (2015)), we fix $F_z = -0.3$ here for simplicity. Since the integrator is not a physical state (it is implemented in the controller) and merely affects the long-term behavior, we do not include it in the computation of the cost (3.8).

3.4.2 Implementation choices

We choose the performance weights to be

$$Q = \mathrm{diag}(1, 100, 10, 200), \; R = 10 \tag{3.10}$$

where $\mathrm{diag}(\cdot)$ denotes the diagonal matrix with the arguments on the diagonal. We desire to have a quiet overall motion in the system. Therefore, we penalize the velocities $\dot{\psi}_k$ and $\dot{s}_k$ more than the position states.

We conducted two types of tuning experiments, one with two parameters and another one with four. The corresponding design weights are

- 2D tuning experiments:

$$W_x(x) = \mathrm{diag}(1, 50x_1, 10, 50x_2), \; W_u(x) = 10 \tag{3.11}$$

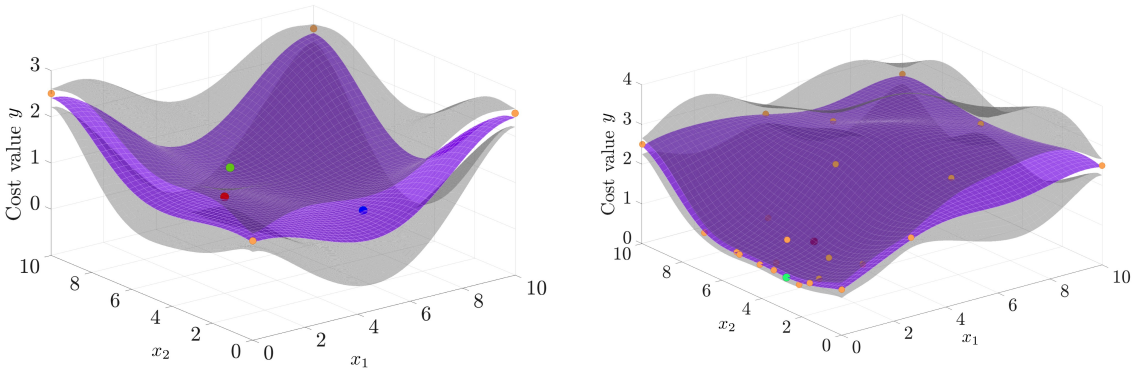
  where the parameters $x = [x_1, x_2]$ can vary in $[0.01, 10]$, and $x^0 = [2, 4]$ is chosen as initial value.
- 4D tuning experiments:

$$\begin{aligned} W_x(x) &= \mathrm{diag}(x_1, 25x_2, 10x_3, 25x_4), \\ W_u(x) &= 10 \end{aligned} \tag{3.12}$$

  with $x = [x_1, x_2, x_3, x_4]$, $x_j \in [0.01, 10]$, and $x^0 = [1, 4, 1, 8]$.

In both cases, the initial choice x^0 is such that the design weights equal the performance weights. That is, the first controller tested corresponds to the nominal LQR design (3.5).

Balancing experiments were run for 2 minutes, i.e. a discrete time horizon of $M = 1.2 \cdot 10^5$ steps. We start the experiments from roughly the same initial condition.



(a) ES initialization

(b) GP posterior after 20 iterations

Figure 3.2: GPs at (a) the start and (b) the end of the first tuning experiment. The GP mean is represented in violet and ± two standard deviations in gray. The red dot corresponds to the initial controller, computed at location $x^0 = [2, 4]$. The green dot represents the current best guess for the location of the minimum. The blue dot is the location suggested by ES to evaluate next, and orange dots represent previous evaluations.The best guess found after 20 iterations (green dot in (b)) has significantly lower cost than the initial controller (red dot).

To remove the effect of the transient and slightly varying initial conditions, we omit the first 30 s from each experiment.

Because the nominal model does not capture the true dynamics, some LQR controllers obtained during the tuning procedure destabilized the system. This means that the system exceeded either acceleration bounds or safety constraints on the end-effector position. In these cases, the experiment was stopped and a fixed heuristic cost f_u was assigned to the experiment. Values for f_u are typically chosen slightly larger than the performance of a stable but poor controller. We used $f_\mathrm{u} = 3.0$ and $f_\mathrm{u} = 5.0$ for the 2D and 4D experiments, respectively.

Before running ES, a few experiments were done to acquire knowledge about the hyperparameters $\mathcal{H}$. A Gamma prior distribution was assumed over each hy-

Table 3.1: Characterization of the gamma prior over $\mathcal{H}$

	2D exploration		4D exploration	
	$\mathbb{E}[\cdot]$	Std $[\cdot]$	$\mathbb{E}[\cdot]$	Std $[\cdot]$
Lengthscale λ_j	2.5	0.11	2.00	0.63
Signal variance σ_S	0.2	0.02	0.75	0.075
Likelihood noise σ^2	0.033	0.0033	0.033	0.010

perparameter with expected values and variances shown in Table 3.1. For the first iteration of ES, we use these expectations as initial set $\mathcal{H}$. After each iteration, $\mathcal{H}$ is updated as the result of maximizing the GP marginal likelihood.

3.4.3 Results from 2D experiments

For the 2D experiments (3.11), we first use a short pole (Fig. 3.1, right) and the best available linear model, showing that the framework is able to improve the initial controller. Secondly, we exchange the pole with one of double length (Fig. 3.1, left), but keep the same nominal model. We show, for the latter case, that even with a 50% underestimated model, the framework finds a stable controller with good performance. In both cases, we use the design weights (3.11).

Using an accurate nominal model

ES was initialized with five evaluations, i.e. the initial controller x^0, and evaluations at the four corners of the domain $[0.01, 10]^2$. Fig. 3.2a shows the 2D Gaussian process including the five initial data points. The algorithm can also work without these initial evaluations; however, we found that they provide useful pre-structuring of the GP and tend to speed up the learning. This way, the algorithm focuses on interesting regions more quickly.

Executing Alg. 3 for 20 iterations (i.e. 20 balancing experiments) resulted in the posterior GP shown in Fig. 3.2b. The "best guess" $x^{\mathrm{BG}} = [0.01, 2.80]$ (green dot) is what ES suggests to be the location of the minimum of the underlying cost (3.3).

In order to evaluate the result of the automatic LQR tuning, we computed the cost of the resulting controller (best guess after 20 iterations) in five separate balancing experiments. The average and standard deviation of these experiments are shown in Table 3.2 (left column, bottom), together with the average and standard deviation of the initial controller, computed in the same way before starting the exploration (left column, top). Even though the initial controller was obtained from the best linear model we had, the performance was still improved by 31.9%.

Using a poor nominal model

In this experiment, we take the same nominal model as in the previous case, but we use a longer pole in the experimental demonstrator (Fig. 3.1, left). The initial controller, computed with x^0, destabilizes the system, which can be explained by the nominal model significantly misrepresenting the true dynamics. As shown in Figure 3.3, after 20 iterations, ES suggested $x^{\mathrm{BG}} = [3.25, 0.01]$ as the best controller. The

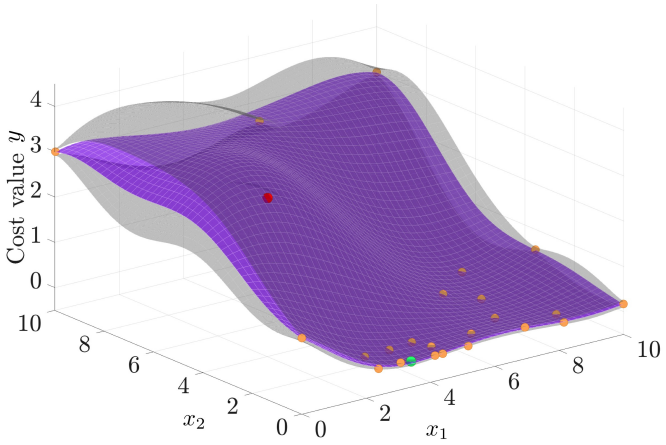


Figure 3.3: Final GP posterior for the second tuning experiment using a wrong nominal model. The color scheme is the same as in Fig. 3.2.

results of evaluating this controller five times, in comparison to the initial controller, are shown in Table 3.2 (middle column).

3.4.4 Results from 4D experiment

The 4D tuning experiment, realized with the long pole, uses the same nominal model as in the previous experiments (i.e., a poor linear model for the real plant), and the design weights (3.12). We show that the framework is able to improve the controller found during the 2D experiments with the long pole, but in a higher dimensional space.

The first controller x^0 destabilizes the system. After 46 iterations, ES suggests $x^{\mathrm{BG}} = [4.21, 7.47, 0.43, 0.01]$, which in comparison with the 2D experiments with the long pole, performs about 31.7% better (see Table 3.2). We actually ran this experiment until iteration 50, however, the algorithm did not lead to further improvements.

Figure 3.4 shows the cost function evaluations over the course of the tuning ex-

Table 3.2: Cost values y for three tuning experiments

	2D experiments Good model		2D experiments Poor model		4D experiments Poor model	
	mean	std	mean	std	mean	std
x^0	1.12	0.11	f_{u}	-	f_{u}	-
x^{BG}	0.76	0.058	0.059	0.012	0.040	0.0031

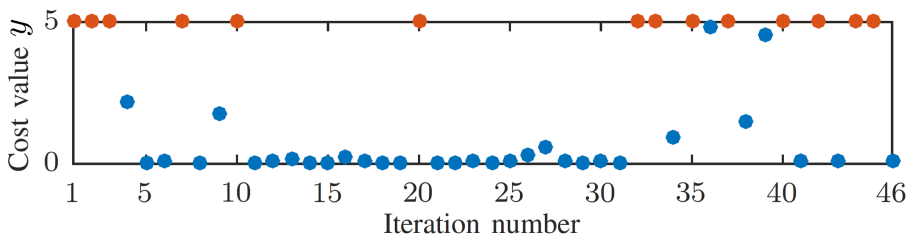


Figure 3.4: Cost values obtained at each experiment distinguishing stable controllers (blue dots), and unstable controllers (red dots).

periment. The fact that unstable controllers are obtained throughout the experiment reflects how the global search tends to cover all areas.

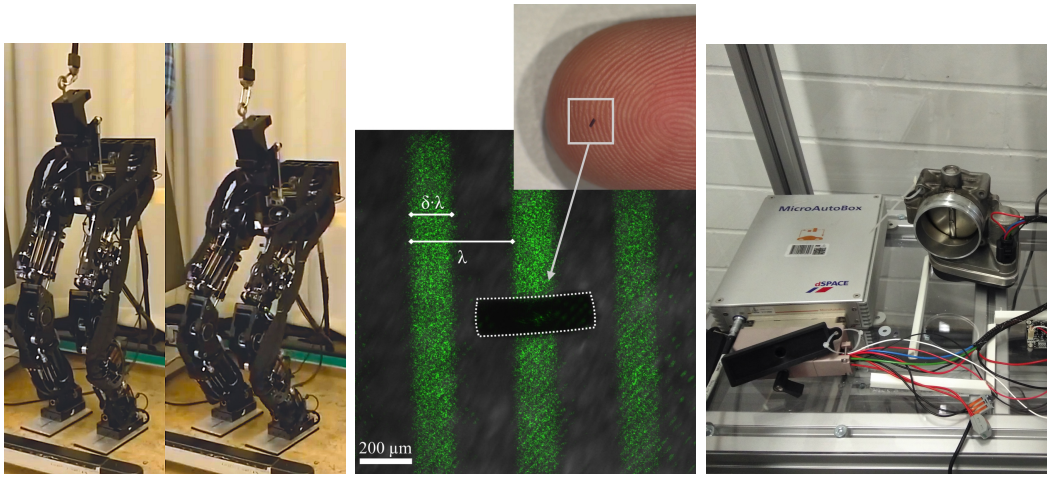
Before starting the 2D experiments, we spent some effort selecting the method's parameters, such as hyperparameters and parameter ranges. In contrast, we started the 4D experiments without such prior tuning. In particular, we kept the same performance weights, chose similar design weights, and started with the same values for the hyperparameters $\mathcal{H}$ and penalty f_{u}. However, we had to restart the search twice in order to slightly adjust $\mathcal{H}$, and f_{u}.

In general, any method reasoning about functions on continuous domains from a finite number of data points relies on prior assumptions (see (Hennig and Schuler, 2012, Sec. 1.1) for a discussion). We were quite pleased with the outcome of the tuning experiments and, in particular, that not much had to be changed moving from the 2D to 4D experiment. Nonetheless, developing general rules for choosing the parameters of GP-based optimizers like ES (maybe specific for certain problems) seems important for future developments.

3.5 Automatic LQR tuning on a humanoid robot

In the present chapter, we demonstrated the automatic LQR tuning framework (cf. Sec. 3.3) by learning the gains of a cart-pole system as an entry-level problem. Because the pole is handled by a robot arm, this platform poses a bigger challenge than standard cart-pole systems. However, the tuning is carried at the highest level in the control hierarchy, i.e., directly on the cart-pole system. In contrast, the low-level feedback and feedforward control loops that are needed to move the robot arm are assumed to be given and properly tuned. These low-level controllers are much more high dimensional and thus, more challenging to tune. In consequence, the tuning problem presented herein, which is at the highest level in the control hierarchy, remains low dimensional, i.e., up to four dimensions (cf. Sec. 3.4).

In order to explore the applicability of the automatic LQR tuning framework, we



(a) Hermes (lower torso) (b) Microrrobot (c) Throttle valve

Figure 3.5: (a) Two-legged hydraulic robot. (b) Soft microrobot powered and controlled by structured light field. The picture shows the robot (black with dotted outline) and linear light waves with wavelength λ and duty cycle δ; © 2018 IEEE. (c) Throttle valve platform used as experimental testbed; © 2019 IEEE.

considered scaling up the dimensionality of the tuning problem by tuning an LQR controller on a humanoid robot at the lowest level: From torques to states. For this, we leveraged existing work (Mason et al., 2014), where a low-level LQR controller was implemented to achieve a squatting task on the humanoid robot Hermes (see Fig. 3.5a). In such work, the diagonal terms of the weighting matrices of the LQR controller were manually tuned. This required non-trivial *domain knowledge* (also referred to as "expert knowledge"), and significant effort and time. Hence, the goal of this project is to show that the automatic LQR tuning framework can achieve same or better performance than the manually tuned controller, by mitigating the hurdle of manual tuning and reducing the amount of human supervision and domain knowledge.

Our results show that, by using a clever parametrization of the diagonal matrices of the LQR weights, Entropy Search (ES) improves the best controller reported in (Mason et al., 2014) in 20 iterations. The obtained results within this project remain for the moment unpublished, yet they are mature enough.

Among the challenges existing in automatically tuning a controller on a humanoid robot, an important one is tackling the high dimensionality of the problem with ES. In the following, we first discuss how to overcome that issue and then we present the experimental setup and the results.

3.5.1 Effective controller parametrization in Hermes

In this section, we discuss the challenges involved in finding an appropriate parametrization for a high dimensional robot in the context of the LQR tuning approach proposed in Sec. 3.2. Same as in previous work (Mason et al., 2014), we used the robot Hermes without its torso. This non-linear and high-dimensional system counts with $u \in \mathbb{R}^{14}$ active hydraulic actuators: Three in the ankle, one in the knee and three in the hip, on each leg. The state space $(q, \dot{q}, s, \dot{s}) \in \mathbb{R}^{40}$ includes the joint positions $q \in \mathbb{R}^{14}$ (one per actuator) and velocities $\dot{q} \in \mathbb{R}^{14}$ and the pose (position and orientation) of the center of mass of the robot $s \in \mathbb{R}^6$ and its velocity $\dot{s} \in \mathbb{R}^6$. Since the squatting task requires both feet to remain on the ground, the center of mass could be estimated using forward kinematics.

Although the true system is non-linear, the LQR problem assumes a linear model. The authors Mason et al. (2014) relied on an existing linearized model of the full dynamics around an equilibrium point, obtained from first principles. In the context of the automatic LQR tuning framework (Sec. 3.2 and 3.3), this constitutes the *nominal model*, which is needed to compute the feedback gain $F(x)$ (3.6) by means of the design weights $(W_s(x), W_u(x))$. Given the dimensionality of the system, the terms in the diagonal of the design weights add up to $D = 14 + 40 = 54$ parameters.

Such dimensionality remains arguably large for the tuning problem with BO and just a few evaluations. To further reduce it, we propose a lower dimensional parametrization by leveraging some characteristics of the squatting movement, such as symmetry between legs. Specifically, we parametrize jointly those terms that are penalizing joints that participate in the squatting movement with the same role. Effectively, this parametrization reduced the dimensionality of the problem from $D = 54$ to $D = 6$, i.e., $x \in \mathbb{R}^6$. For example, the first parameter x_1 penalizes simultaneously the flexion-extension movement of the ankle of both legs, while the sixth parameter x_6 penalizes simultaneously rotatory movement of the ankle and the hip from both legs. This dimensionality reductions allowed to learn a good controller with a few evaluations using ES.

A significant challenge and key to success in this project was programming a robust communication interface between ES and Hermes. Our interface completely automates the learning loop, so that once the optimization started, no further human interaction is required. Details about such interface are reported in App. A.2, and its c++ implementation is publicly available at https://github.com/alonrot/userES_pubsub_lqr.

For this project, we reimplemented ES in c++, which increased computational speed by about ten times with respect to the original Matlab implementation. The c++ implementation is available at https://github.com/alonrot/EntropySearchCpp.

Also, an alternative version to the original Matlab code can be found in https://github.com/alonrot/userES.

Implementation choices

The squatting task is realized by tracking a sine reference s_k^{ref} that moves the center of mass of the robot up and down [3] . Details about the reference trajectory are given in App. A.1.

We measure the performance of the squatting task by modifying by penalizing the tracking error $\left[s_k^{\text{ref}} - s_k\right]$ in the quadratic cost function (3.3) as follows:

$$f = \lim_{M\to\infty} \frac{1}{M+1} \mathbb{E}\left[\sum_{k=0}^{M} \left[s_k^{\text{ref}} - s_k\right]^\top Q \left[s_k^{\text{ref}} - s_k\right] + u_k^\top R u_k\right]. \tag{3.13}$$

The objective f is modeled as a GP, i.e., $f \sim \mathcal{GP}\left(0, k_{\text{mat}}\right)$, where the Matérn kernel is given in (2.6) and a prior zero mean is assumed. The hyperparameters of the GP were fixed to $\lambda_i = 0.1$, $\sigma = 0.01$, $\sigma_{\text{S}} = 0.5$. Similarly to (3.8), the observations y are given by removing the limit and the expectation from (3.13), and setting a fixed time horizon of 5 seconds, i.e., $M = 5 \cdot 10^3$, which corresponds to 6 squats.

We used similar performance weights (Q, R) to those used in (Mason et al., 2014), which we report in App. A.1.

To achieve the aforementioned dimensionality reduction, we first set $W_u(x) = R$ and $W_s(x) = Q$, and then modify the pertinent terms of the diagonal of $W_s(x)$ as

$$\begin{aligned}
[W_s(x)]_{6,6} &= [W_s(x)]_{13,13} = 10^{2x_1+4} \\
[W_s(x)]_{4,4} &= [W_s(x)]_{11,11} = 10^{2x_2+4} \\
[W_s(x)]_{1,1} &= [W_s(x)]_{8,8} = 10^{2x_3+4} \\
[W_s(x)]_{2,2} &= [W_s(x)]_{9,9} = 10^{2x_4+3} \\
[W_s(x)]_{7,7} &= [W_s(x)]_{14,14} = 10^{x_5+5} \\
[W_s(x)]_{3,3} &= [W_s(x)]_{5,5} = [W_s(x)]_{10,10} = [W_s(x)]_{12,12} = 10^{4x_6+1},
\end{aligned} \tag{3.14}$$

where the controller parameters $x \in \mathcal{X}$ are bounded to a unit hypercube $\mathcal{X} = [0,1]^D$ for simplicity. The pair of indices $(6, 13)$, $(4, 11)$ and $(1, 8)$ are coupling together the same joints in both legs and refer to the flexion-extension movements of the ankle, knee and hip, respectively. The pair of indices $(2, 9)$ and $(7, 14)$ are also coupling together the same joints in both legs and refer to the abduction-adduction movement

[3] Because the LQR tuning framework (Sec. 3.2) penalizes deviation errors from a zero equilibrium, the obtained control gain $F(x)$ remains valid for tracking a reference, as long as the robot state remains close to the equilibrium point.

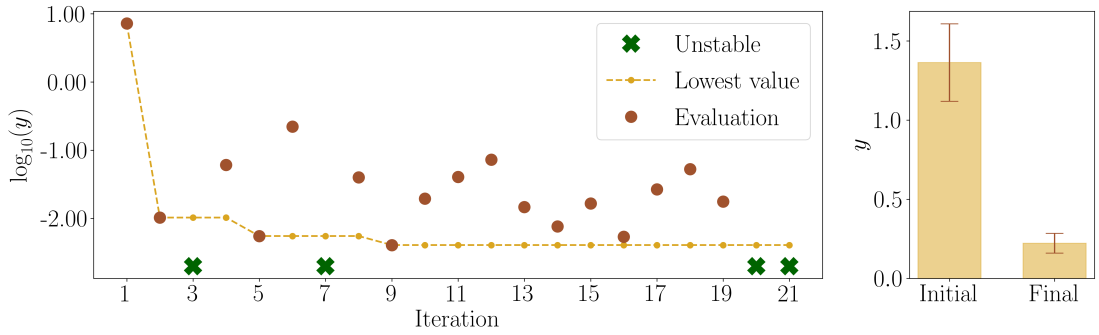


Figure 3.6: Left: Cost values y obtained in 20 iterations of ES. The cost corresponding to the initial parameters x^0 is shown at iteration 1. Right: Comparison of the performance given by the best guess x^{BG} against the initial performance, averaged over five and three experiments, respectively. The error bars show standard deviation.

of the hip and the ankle, respectively. Finally, the indices $3, 5, 10, 12$ couple together the rotatory joints of the ankle and the hip from both legs.

Same as in Sec. 3.4.2, the initial parameters x^0 were chosen such that $W_s(x^0) = Q$ and $W_u(x^0) = R$. Such initial parameters yield poor squatting performance, yet it does not destabilize the robot. After $N = 20$ iterations, ES reports the "best guess" x^{BG} as the learned controller parameters (cf. (2.9)).

For those controllers that destabilize the robot, we assigned a heuristic cost $f_{\text{u}} = 3.0$.

Result

Fig. 3.6 (left) shows the obtained cost values while running ES. The best guess is reported at $x^{\text{BG}} = [0.77, 0.28, 0.5, 0.48, 0.63, 0.37]$. In Fig. 3.6 (right) we show the value of the initial and the final controllers. To obtain them, we conducted five experiments at the best guess location x^{BG} and three experiments at the initial location x^0.

To assess the robustness of the learned controller, we (i) increased the frequency of the squatting task from 0.8 Hz to 1.2 Hz and (ii) poke the robot with a stick to make sure it is robust against disturbances while squatting. A video summary of the learning routine, the resulting controller and the robustness tests can be found at at https://youtu.be/udJAK6OIWEc.

3.6 Additional contributions

As in the work presented in this chapter, herein we briefly discuss two collaborations where BO was used to automatically tune controllers on real systems.

3.6.1 Gait learning for soft microrobots

In (von Rohr et al., 2018), the goal is to learn a gait controller for a soft microrobot, made of a photoresponsive material (see Fig. 3.5b). Therein, the controller consisted on a light field falling onto the microrobot, which generated a gait. We used BO to tune the parameters of the light field. This resulted in a 115% improvement in the locomotion performance as compared to an informed initial guess, only after 20 experiments. In this case, no accurate locomotion models were available, contrary to the work presented in this chapter, where an approximate dynamics model is used. Additionally, an initial guess on the objective function was made by acquiring 56 cost evaluations.

3.6.2 Automatic controller tuning in an industrial setting

Additionally, in (Neumann-Brosig et al., 2019), BO was used to automatically tune the controller of a throttle valve (see Fig. 3.5c), as an industrial application example. Our results indicate that BO outperforms manual calibration as it consistently achieves better performance with a lower number of experiments. Contrary the work presented in this chapter, where the controller performance is measured with a quadratic cost, herein we used performance metrics typically used in the industry, such as "signal overshoot" and "response time" (T_{90}).

3.6.3 Apollo pole balancing using Kinect

Something to notice in the robotic setup described in Sec. 3.4 (i.e., balancing an inverted pole with Apollo) is the use of a motion capture system (Vicon) to estimate the pole angular position. This type of off-board sensors can be seen as a disadvantage in the context of autonomous robots working among humans in the near future; therein, the robot will most likely have to rely on on-board sensors to move freely through the environment. The goal of this project is to replace the motion capture system (Vicon) with a Kinect sensor, which is mounted on Apollo's forehead. Such sensor provides RGB and depth information at a much lower rate (i.e., 30 Hz, while Vicon can operate at 200 Hz). In addition, the noise is larger and the delays can be up to 2-3 frames. All this amounts to a "vision-in-the-loop" much challenging

problem, which needs from state-of-the-art computer vision techniques to be solved efficiently.

In this project, we used a particle filter (Issac et al., 2016) to estimate the position of the pole. More concretely, a model-based 3D-tracking filtering algorithm uses depth images to estimate the position of the pole. An outstanding feature of such method is that it compensates for the measurement fat-tails, usually present in depth sensory data. We showed that it is possible to balance the pole using an LQR controller, despite of the inherent delays of the sensor. However, despite its potential, this project is not on a mature state for publication yet.

3.7 Conclusions

In this chapter, we have used Bayesian optimization (BO) for automatic controller tuning. We develop, and successfully demonstrate a framework based on LQR tuning (Trimpe et al., 2014) and Entropy Search (ES) (Hennig and Schuler, 2012) on two robotic platforms: An inverted pendulum balanced with the humanoid robot Apollo and a squatting task performed by the two-legged robot Hermes. This work is the first to apply ES in experiments for automatic controller tuning.

The results with Apollo show that the auto-tuning algorithm was successful in a 2D and a 4D experiment, both when the method was initialized with an unstable and with a stable controller. While the 2D experiment could presumably also be handled by grid search or manual tuning, and thus mostly served as a proof of concept, the 4D tuning problem can already be considered difficult for a human.

The results on Hermes demonstrate that it is possible to automatically tune the low-level controller parameters on a high-dimensional system using BO. More specifically, our results show improvement upon a poor controller learned after 20 experiments. In addition, the LQR tuning framework and the effective dimensionality reduction (Sec. 3.5.1) help to mitigate manual tuning effort.

In this section, we discuss the advantages and drawbacks of the proposed automatic LQR tuning method. Additionally, we discuss future research lines to address open questions related to this project.

3.7.1 Discussion

There are two key elements for the proposed automatic LQR tuning framework to succeed in practice. First, a thoughtful parametrization of the design weights (cf. Sec. 3.2.2) is important and can help to reduce the dimensionality of the tuning problem. Spending the effort of finding such parametrization has a payback on

autonomy, as it mitigates the human interaction with the robot and also allows to automatically improve a poor controller only with high level human supervision (e.g., for safety). Second, robustness in the communication framework was crucial for success, as many technical issues can easily break the communication between the optimizer (ES) and the robot. For example, if the robot needs to be emergency-stopped due to an aggressive unstable controller, the optimizer needs to be informed to wait for the robot to be restarted, instead of continuing or timing out (for more details about the communication framework see App. A.2).

Generally speaking, the proposed automatic LQR tuning framework poses some clear benefits, which we discuss below.

- A clear benefit of the LQR tuning framework (Sec. 3.2) in a high dimensional setting, such as in Hermes, is related to the dimensionality reduction: While the naive, but straightforward choice would be to directly learn the feedback gain matrix, i.e., $x = F$, this would imply a $14 \times 40 = 560$ dimensional problem, which is a too large dimensionality for learning from a few evaluations using BO. Furthermore, we would not have leveraged the availability of the nominal model. In contrast, the LQR framework learns a parametrized feedback gain $F(x)$ (3.6) through the diagonal of the LQR weighting matrices ($W_s(x)$, $W_u(x)$), which implies $14 + 40 = 54$ parameters instead of 560.
- Although not completely reduced, human supervision was mitigated during the learning process. This was possible due to the robust communication framework that allowed to carry out the entire optimization loop without human intervention. For security reasons, an operator has to be ready to stop the robot platform by pressing an emergency button. However, this is a common safety measure in many robotic platforms.
- The communcation framework can be reused without major alterations to optimize controller parameters in other tasks.

A key question for the development of truly automatic tuning methods is the amount of "prior engineering" that has to be spent to get the method to work. In particular, the 4D experiments on Apollo were promising since not a lot of tuning, and only few restarts were necessary. The 6D experiments on Hermes also involved a few restarts, but it required careful parametrization of the design weights. Below, we discuss some aspects about the required "prior engineering" for the automatic LQR tuning to work in practice.

1) In the project with Hermes, non-trivial efforts were spent on properly tuning the matrices Q and R to compute the quadratic cost (3.13). Because some states could be noisier or larger in magnitude than others, they could outweigh all others in the final cost value. To avoid this, domain knowledge on the robotic

platform is required and real experiments need to be manually carried to properly tune Q and R, before starting ES.

2) In the project with Hermes, reducing the dimensionality of the parameter space down to 6 in the design weights carried some drawbacks. First, finding the appropriate coupling terms in the design weights required (i) knowing a priori the type of movement to be made (i.e., squatting), and (ii) using domain knowledge to decide what joints to couple together. Second, this process required doing experiments on the robot before starting ES.
3) The LQR problem assumes a linear system. This poses two clear disadvantages: (i) in case of a robot (which are typically non-linear systems), a linearized model around an equilibrium point is assumed to be available, and (ii) the movements of the robot are restricted to be quasi-static around the equilibrium point. For instance, automatic LQR tuning cannot work if the robot is walking, jumping or extending a leg.

It is generally important to understand how to overcome these problems. For example, about the first issue: When working with real systems, coming up with an effective, yet simple, reward function unavoidably requires domain knowledge about the platform. However, reducing the number of prior experiments is possible by having less complicated reward functions. In the particular case presented in this chapter, we could simply replace the proposed quadratic cost with the tracking error of the center of mass. In such case, a single state participates in the final cost value, which solves the problem of trading off the influence of different states. In other settings, many states might have to be penalized. In such case, an additional BO routine could be executed in an outer loop to aid the automatic tuning of such parameters, constituting a meta-learning approach.

About the second issue: Dimensionality reduction of a humanoid robot using domain knowledge has been recently proposed in the context of BO (Yuan et al., 2019). In the present case, although symmetries in the squatting movement made it possible to reduce the dimensionality, such symmetries might not be present in other tasks. The number of evaluations required in standard BO to have a good coverage requires an exponential amount of evaluations. Provided the intrinsic effective dimensionality is low, REMBO (Wang et al., 2016) has been proposed as a solution to tackle high dimensional spaces. Furthermore, (Wang and Jegelka, 2017) proposes an additive model that allows to scale to higher dimensions.

About the third issue: The automatic tuning framework presented in this chapter is not strictly tied to LQR problems. For example, BO could be used to automatically tune parameters of the iterative LQR (Li and Todorov, 2004) control architecture, which is designed for robots operating along trajectories, rather than equilib-

rium points, and assumes a non-linear model of the system. Alternatively, model-based reinforcement learning (MBRL) algorithms usually involve a small number of parameters, either in the reward shaping or in the algorithm itself (Chua et al., 2018; Deisenroth and Rasmussen, 2011; Kamthe and Deisenroth, 2018). In those cases, BO could aid learning such parameters in an outer optimization loop. In fact, BO has been used to tune controller parameters of very different control architectures (Antonova et al., 2016; Bansal et al., 2017; Büchler et al., 2019) (see Sec. 1.5.2 for a review).

3.7.2 Open problems

Besides the aforementioned issues, there are a number of open problems that we encountered while working on these projects. Herein, be briefly discuss them and how they connect with the subsequent chapters of this thesis.

First, an important aspect of the presented framework is the treatment given to unstable controllers. Both, in the project with Apollo and the project with Hermes, many aggressive controllers were encountered during the learning routine. In these cases, the platform was stopped by pressing an emergency button. In that case, the obtained data was scarce or non existent, and not comparable with the data obtained from stable controllers. In Chap. 6 we specifically address this issue by introducing safety constraints into the tuning problem.

Second, in these projects we have chosen standard kernels to configure the Gaussian process models. Because we have no accurate understanding of the true shape of the cost function before (and also not after) we carry out experiments on the robot, it is unclear what is the best kernel choice. Would it be possible to design an expressive kernel tailored to the control problem at hand? In Chap. 5 we study this and some related open questions.

Third, as typically happens in robotics, before trying a controller on the real system, we make sure that such controller does work in simulation. In Chap. 4 we study how to incorporate the simulation into the learning loop.

Chapter 4

Virtual vs. real: Trading off simulations and physical experiments using Bayesian optimization

This chapter presents the contents of our conference publication (Marco et al., 2017). A video recording of the spotlight presentation at the conference can be found at https://youtu.be/6ddBAX8-BHQ. The ideas presented herein naturally emerged motivated by one of the shortcomings of the controller learning framework proposed in Chap. 3. Therein, the proposed method is not as sample-efficient as it could be if additional sources of information were injected into the learning loop. Since simulators are usually are readily available for most robotic platforms and informative about what behavior to be expected in reality, we study herein the possibility of including it as an additional information source.

The algorithm presented in this chapter is an extension of Entropy Search (cf. Sec. 2.3.1) to the case of multiple information sources. The result is a principled way to automatically combine cheap, but inaccurate information from simulations with expensive and accurate physical experiments in a cost-effective manner. We apply the resulting method to a cart-pole system, which confirms that the algorithm can find good control policies with fewer experiments than standard Bayesian optimization on the physical system only. Furthermore, we also use herein the LQR tuning framework proposed in Sec. 3.2.

In this chapter[1], we first introduce the problem and enumerate the contributions in Sec. 4.1. Then, we formulate the problem in Sec. 4.2 and discuss the proposed method in Sec. 4.3. After, we present our results in Sec. 4.4 and finally conclude in Sec. 4.5.

[1]For an extensive literature review see Sec. 1.5.3

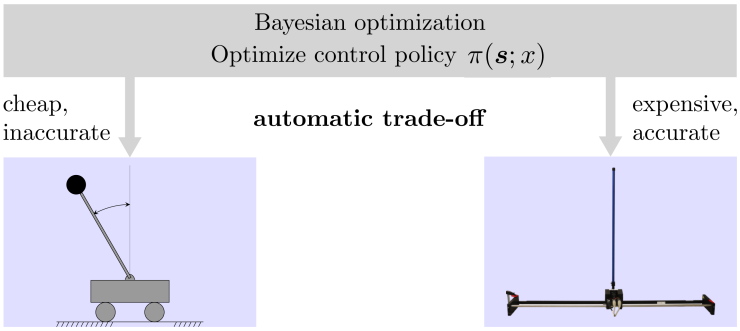


Figure 4.1: The proposed algorithm optimizes the parameters x of a control policy based on data of a cheap, but inaccurate simulation and expensive data from the real system. By actively trading off between the information that can be gained from each system relative to their costs, the algorithm requires significantly fewer evaluations on the physical system.

4.1 Introduction

Typically, the control policies that are used in robotics depend on a small set of tuning parameters. To achieve the best performance on the real system, these parameters are usually tuned manually in experiments on the physical platform. Policy search methods in reinforcement learning aim to automate this process (Sutton and Barto, 1998). However, without prior knowledge, these methods can require significant amounts of experimental time before determining optimal, or even only reasonable parameters. In robotics, simulation models of the robotic system are usually available, e.g., as a by-product of the design process. While exploiting knowledge from simulation models has been considered before, no principled way to trade off between the relative costs and accuracies of simulations and experiments exists (Kober et al., 2013). As a result, state-of-the-art reinforcement learning methods require more experimental time on the real system than necessary.

In this chapter, we present a Bayesian optimization algorithm that can automatically optimize the parameters of control policies based on data from different information sources, such as simulations and experiments. Specifically, we use an extension of Entropy Search (Hennig and Schuler, 2012; Villemonteix et al., 2009), a Bayesian optimization framework for information-efficient global optimization. Therein, entropy is used to measure the information content of simulations and experiments. Since this is an appropriate unit of measure for the utility of both sources, our algorithm is able to compare physically meaningful quantities in the same units on either side, and trade off the amount of information gained from different sources with their respective costs. As a result, the algorithm can automatically decide whether to evaluate cheap, but inaccurate simulations or perform

expensive and precise real experiments. This results in fewer physical experiments to determine optimal parameters (see Fig. 4.1). We apply the method to optimize the policy of a cart-pole system and show that this approach can speed up the optimization process significantly compared to standard Bayesian optimization (cf. Sec. 2.3). The main contributions of this chapter are (i) a novel Bayesian optimization algorithm that can trade off between costs of multiple information sources and (ii) the first application of such a framework to the problem of reinforcement learning and optimization of controller parameters.

For convenience within the next sections, we rename the concepts *accuracy* and *cost*: We now refer to the lack of accuracy of a controller as *cost*. Furthermore, we now denominate the cost of retrieving an evaluation from a specific information source as *effort*.

4.2 Problem Statement

Throughout this chapter, we aim at solving the same controller tuning problem presented in Sec. 2.1. As stated therein, the goal is to find an optimal controller to complete a certain task on a dynamic system. We assume no dynamics model is given, but a controller $u_k = \pi(s_k; x)$, parametrized with parameters $x \in \mathcal{X}$, is assumed to be known. The goal is to determine the optimal parameters x_* that globally minimize the cost f of a task,

$$x_* = \underset{x \in \mathcal{D}}{\operatorname{argmin}}\, f(x), \tag{4.1}$$

where queries $f(x)$ involve doing an experiment on the real robot and measuring an error signal over a fixed time horizon. Since these experiments cause wear in the robot and take time, the goal is to minimize the number of iterations before the optimal parameters in (4.1) are determined. To this end, we use Bayesian optimization (BO) (cf. Sec. 2.3).

We assume that a simulation of the system is available, which we want to exploit to solve (4.1) more efficiently with fewer evaluations on the physical system. While simulations are only an approximation of the real world and cannot be used to determine the optimal parameters in (4.1) directly, they can be used to provide an estimate $f_{\text{sim}}(x)$ of the true cost. We use this estimate to obtain information about the location of the optimal parameters on the real system. As a result, at each iteration t, we do not only choose the next parameters x_{t+1} to evaluate, but also whether to perform a simulation or an experiment.

Both experiments, in the real world and in simulation, have physically meaningful evaluation efforts associated to them. For example, the effort may account for the

amount of time required to complete an experiment/simulation and for monetary costs such as wear in the system. The overall goal is to minimize the total effort incurred in the experiments and simulations until the optimal parameters (4.1) on the real system are found.

We assume the cost is modeled with a Gaussian process (GP) $f \sim \mathcal{GP}\left(m(x), k(x, x')\right)$. We use the same notation introduced in Sec. 2.2. Furthermore, same as in Chap. 3, we use Entropy Search (ES) (cf. Sec. 2.3.1) to address the automatic controller tuning problem (4.1).

4.3 Reinforcement Learning with Simulations

In this section, we show how the ES algorithm can be extended to multiple sources of information, such as simulations and physical experiments. The two main challenges are modeling the errors of the simulator in a principled way and trading off evaluation effort and information gain. In the following, we focus on the case where only one simulation is available for ease of exposition. However, the approach can easily be extended to an arbitrary number of information sources.

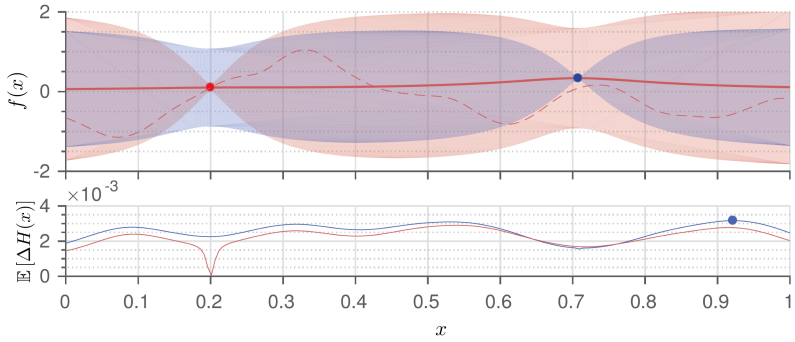
4.3.1 GP model for multiple information sources

To model the choice between simulation and physical experiment, we use a specific kernel structure that is similar to the one used in (Poloczek et al., 2017). The key idea is to model the cost on the real system as being partly explained through the simulator plus some error term. That is, $f(x) = f_{\text{sim}}(x) + f_{\text{err}}(x)$, where the true cost consists of the estimated cost of the simulation, $f_{\text{sim}}(x)$, and a systematic error term, $f_{\text{err}}(x)$. To incorporate this in the GP framework of Sec. 2.2, we extend the parameter vector by an additional binary variable δ, which indicates whether the cost is evaluated in simulation ($\delta = 0$) or on the physical system ($\delta = 1$). Based on the extended parameter $\hat{x} = (x, \delta)$, we can model the cost by adapting the GP kernel to

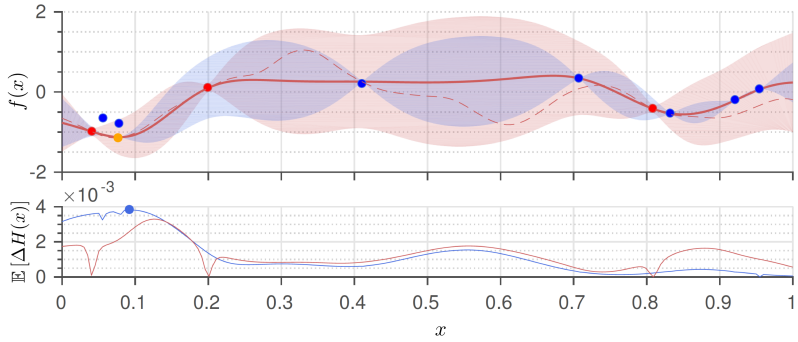
$$k(\hat{x}, \hat{x}') = k_{\text{sim}}(x, x') + k_{\delta}(\delta, \delta')\, k_{\text{err}}(x, x'). \tag{4.2}$$

The kernels $k_{\text{sim}}(\cdot, \cdot)$ and $k_{\text{err}}(\cdot, \cdot)$ model the cost function on the simulator and its difference to the cost on the physical system, respectively. The kernel $k_{\delta}(\delta, \delta') = \delta\delta'$ is equal to one if both parameters indicate a physical experiment and zero otherwise.

From Sec. 2.2, we know that the kernel (4.2) models the covariances for different parameters. Intuitively, the kernel (4.2) encodes that two experiments on the physical system covary strongly. However, if one of the δ-variables is zero (i.e., a simulation), then the covariance between the two values is captured by k_{sim}. Effectively,



(a) Exploration stage after two evaluations



(b) Exploration stage after ten evaluations

Figure 4.2: Synthetic example of how simulations and physical experiments can be combined by trading off information and evaluation effort. In (a), top, it is shown the GP posterior conditioned on one simulation (blue dot) and one physical experiment (red dot). The GP model from Sec. 4.3.1 encodes that a portion of the uncertainty in the cost of the real system (red shaded) can be explained through the simulator (blue shaded). The red dashed line represents the cost function of the physical system. The cost function of the simulator is omitted for simplicity. In (a), bottom, it is shown the expected information gain per unit of effort of the simulator (blue line), and of the physical system (red line). The most informative point (blue dot) is selected among the two sources by the proposed method as next evaluation (in this case, a simulation). In (b), top, it is shown the GP posterior after nine iterations. The global minimum (orange dot) is found close to the true minimum.

the error covariance is switched off in simulations in order to model that simulations cannot provide all the information about f. By choosing the kernels k_{sim} and k_{err}, we can model to what extend f can be explained by the simulator and thereby its quality. This is illustrated with a synthetic example in Fig. 4.2. The total variance of the cost on the physical system is shown in red. Before any data is observed, it is equal to the uncertainty about the simulation plus the uncertainty about the error. As shown in Fig. 4.2a, the blue shaded region highlights the variance of the simulator. Evaluations in simulation (blue dots) reduce the uncertainty of this blue

shaded region, but reduce only partially the uncertainty about the true cost (red). In contrast, an evaluation on the real system (red dot) allows one to learn the true cost f directly, thus reducing the total uncertainty (red), while some uncertainty about the variance of f_{sim} remains (blue). Having uncertainty about the simulation is by itself irrelevant for the proposed method, because we solely aim to minimize the performance on the physical system.

Next to the kernel, we account for different amounts of noise in simulation (typically noise-free) and on the real system. That is, the noise variance of measurements, σ^2 in (2.3), takes different values, σ^2_{exp} and σ^2_{sim}, depending on whether an experiment or a simulation is chosen. With this kernel and noise structure, the two information sources can be modeled by a single GP, and the predictive distribution can be computed using (2.4).

4.3.2 Optimization

With the GP model defined, we now consider how it can be used to trade off accuracy for evaluation effort. To this end, we extend Entropy Search (ES) (cf. Sec. 2.3.1) to account for multiple information sources. As before, we want to minimize the cost (4.1) on the real system. This means, the distribution over the minimum is defined in terms of the same cost (2.10) as in standard ES. In order to approximate p_{min}, we need to use the GP kernel with the additional δ factor fixed to one,

$$p_{\text{min}}(x) = p(x = \operatorname*{argmin}_{\tilde{x}\in\mathcal{X},\, \delta=1} f(\tilde{x}, \delta)). \tag{4.3}$$

As in ES, the goal is to arrive at a distribution p_{min} that has low entropy (i.e., very peaked on a certain location). The expected change in entropy is an appropriate measure for this. However, this quantity additionally depends on the variable δ, so that the algorithm has an additional degree of freedom in the parameters to optimize. If one were to use the same optimization problem as in (2.12), the algorithm would always choose to evaluate parameters with $\delta = 1$. This is because the experiments with $\delta = 1$ provide information about the cost function f directly, while an evaluation with $\delta = 0$ only provides information about part of the cost, f_{sim}.

To trade off between the two choices more appropriately, we associate an effort measure with both kinds of evaluations; t_{sim} for the simulation and t_{exp} for physical experiments. While simulations are less informative about p_{min}, they are significantly cheaper than experiments on a physical platform so that $t_{\text{sim}} < t_{\text{exp}}$. These effort measures can have physically meaningful units, such as the amount of time taken by a simulation relative to a physical experiment. While the effort measures are important to trade off the relative gains in information, they do not require

tuning. For example, setting the effort of the simulator too high may lead to more experiments on the physical system than necessary, but the optimal parameters on the real system are found regardless.

A key advantage of using entropy to determine progress toward the goal is that it is a consistent unit of measurement for both information sources, even in the case of different noise variances. As a result, we can compare the gain in information about the location of the minimum (i.e., $p_{\min}$) in simulation and physical experiments relative to their efforts. Thus, we select the next parameters, x_{t+1}, and where to evaluate them, δ, according to

$$\underset{x\in\mathcal{D},\, i\in\{\text{sim},\text{exp}\}}{\operatorname{argmax}} \quad \mathbb{E}\left[\Delta H_i(x)\right] \,/\, t_i. \tag{4.4}$$

The expected gain in entropy, $\mathbb{E}\left[\Delta H_i(x)\right]$, depends on whether we evaluate in simulation or physical experiment. By selecting the best gain per unit of effort, the algorithm automatically decides which kind of evaluation decreases the uncertainty about the location of the minimum the most, relative to effort. Importantly, since the GP model in (4.2) is adaptive to the quality of the simulator, the acquisition function (4.4) leads to informed decisions about whether the simulator is reliable enough to lead to additional information.

We illustrate a typical run of the algorithm in Fig. 4.2. The algorithm was initialized with one physical experiment (red dot in Fig. 4.2a) for the purpose of illustration. The evaluation effort of the simulator was set to 40% less of that of the real system. As a result, it is advantageous to exploit initially the low effort that takes to do simulations. The algorithm automatically decides to do so, as can be seen in Fig. 4.2a. The simulation (blue dot) decreases the amount of uncertainty about the simulation model, but provides only partial information about the true cost of the system. As a result, the method eventually starts to evaluate parameters on the real system. Notice that this is not the same as two stage learning, because the algorithm can decide to switch back to simulations if this is beneficial. This is especially important in situations where the quality of the simulation is not known in advance and the hyperparameters of the kernels in (4.2) are optimized. Eventually, the algorithm converges to a distribution $p_{\min}$ that is peaked around the minima of the cost function. Since the model can exploit cheap information from simulation, fewer physical experiments are needed to determine the minimum than if only physical experiments were used.

Because the proposed method extends Entropy Search (ES) to multiple information sources, we refer to it as *Multi-fidelity Entropy Search* (MF-ES). The method outputs the best guess location for the global minimum x^{BG}, which is obtained as in (2.9). Our Matlab implementation of MF-ES can be found at

https://github.com/alonrot/mfES.

4.4 Experimental Results

In this section, we evaluate MF-ES for optimizing the feedback controller of an unstable cart-pole system, as illustrated in Fig. 4.1.

4.4.1 System description

As experimental setup, we use the Quanser Linear Inverted Pendulum (Quanser, 2015). The dynamic equations of the cart-pole system were already introduced in (3.9). To simulate the dynamics, we used the model parameters are reported in (Quanser, 2015). The cart-pole setup is connected through dedicated hardware to a standard Laptop and can be controlled via Matlab/Simulink. A nonlinear Simulink model of the system dynamics (3.9) is provided by the manufacturer and used as the simulator in our setting.

4.4.2 Controller Tuning Problem

Herein, we use the same controller tuning framework presented in Sec. 3.2. Therein, a feedback controller $u_k = F(x)s_k$ is used to control the system, and the feedback gain $F(x) \in \mathbb{R}^{1\times 4}$ is parametrized using the design weights $W_s(x)$ and $W_u(x)$ of the LQR problem (see Sec. 3.2.2 for details). As in Sec. 3.3, the goal is to minimize a quadratic cost function (3.3), defined by means of the performance weights $Q = \text{diag}\,(1, 1, 1, 0.1)$ and $R = 10^{-1.5}$, and a sufficiently long time horizon M. Following Sec. 3.2.2, we parametrized the design weights as

$$W_s(x) = \text{diag}(10^{x_1}, 1, 1, 0.1), \qquad x_1 \in [-3, 2], \tag{4.5}$$

$$W_u(x) = 10^{-x_2}, \qquad x_2 \in [1, 5]. \tag{4.6}$$

Hence, we are left with tuning two parameters, $x \in \mathbb{R}^2$. In spite of the low dimensionality of the problem, our goal is to illustrate how MF-ES works with a real robotic platform. In Chap. 3 we already showed that both, the automatic LQR tuning framework and Entropy Search work well in higher dimensional problems. We also emphasize that LQR weights are only one possible way to parameterize feedback controllers; alternative parameterizations (Roberts et al., 2011) or direct tuning of the gains F (Berkenkamp et al., 2016a) is also possible. The method proposed herein is independent of the specific parameterization used.

An evaluation of the cost $f_{\text{exp}}(x)$ is obtained by computing the controller gain (3.6) based on the weight matrices (4.5), (4.6), performing a 30 s balancing experiment on the physical system, and computing the cost according to (3.8) from the experimental data $\{x_k, u_k\}_{k=0}^{M}$, with $M = 3 \cdot 10^4$. A simulation evaluation $f_{\text{sim}}(x)$ is obtained in the same manner by running a 30 s simulation instead.

If a candidate controller violates safety limits on the states and inputs, it is determined as *unstable*, and we assign a fixed penalty of $f_{\text{exp}} = 0.06$ and $f_{\text{sim}} = 0.04$ for physical experiment and simulation, respectively. These numbers are chosen conservatively larger than the cost of the worse stabilizing controller observed after some a priori initial evaluations. Thus, evaluations during the learning procedure shall not result in higher costs than these.

The controller is automatically tuned over roll-outs without human intervention. To this end, a nominal[2] controller $x_{\text{nom}} = [0,\ 1.5]$ is balancing the pole when no tuning experiment is being performed. The optimizer triggers new experiments, when an evaluation on the real system is required. As soon as the experiment is finished, or instability is detected, the system switches back to the nominal controller. The nominal controller shows very poor performance, which shall be improved with the proposed RL method.

4.4.3 Bayesian Optimization Settings

We apply the method of Sec. 4.3, MF-ES, to optimize the experimental cost (3.3) by querying simulations and experiments. The efforts in (4.4) correspond to the approximate times we need to wait until a simulation is computed and a physical experiment is performed, $t_{\text{sim}} = 1\,\text{s}$ and $t_{\text{exp}} = 30\,\text{s}$, i.e., simulations require 30 times less effort than physical experiments.

For the GP model, we choose the rational quadratic kernel with $\alpha = 1/4$ (see (Rasmussen and Williams, 2006)) for both k_{sim} and k_{err} in (4.2). Hyperparameters, such as length scales and output variances, were chosen from some initial experiments and then held fixed during optimization. As prior mean functions, we use $m_{\text{sim}}(x) \equiv 0.04$ and $m_{\text{err}}(x) \equiv 0.02$, respectively, for the simulation and error GP. These choices correspond to the penalties f_{sim} and f_{exp} given for unstable controllers (adding m_{sim} and m_{err} for the experiment). Hence, the prior mean is pessimistic in the sense that we believe a controller to be unstable before seeing any data. The prior variance of k_{sim} and k_{err} are chosen as $\sigma^2_{S,\text{sim}} = 1.6 \times 10^{-5}$ and $\sigma^2_{S,\text{err}} = 3.84 \times 10^{-4}$ respectively.

[2]The nominal controller is the optimal controller if the true dynamics was linear according to the nominal model (A, B). Then, choosing $W_s(x)$ and $W_u(x)$ corresponding to the cost (3.3) yields the optimal controller F, see Sec. 3.2 for details.

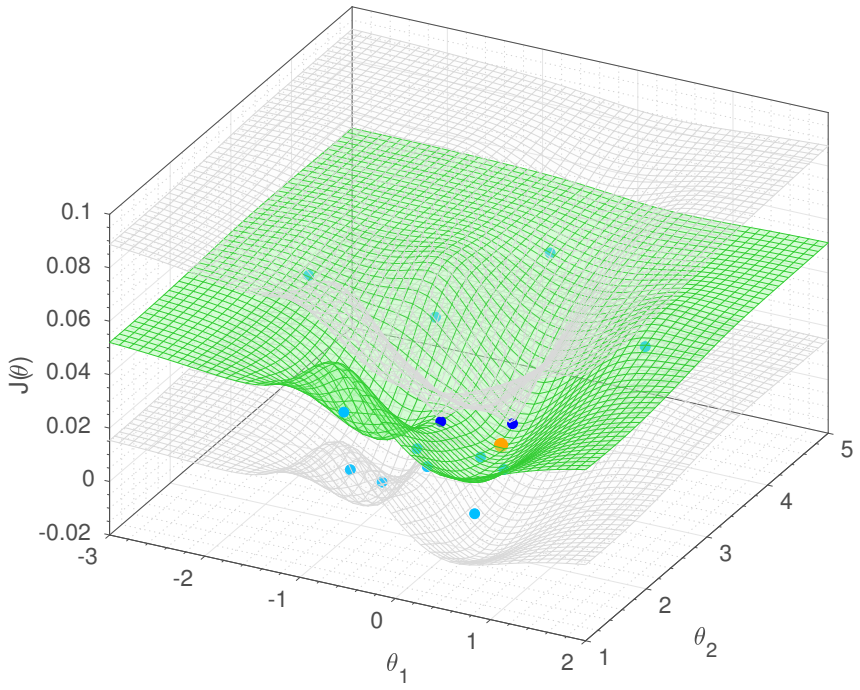


Figure 4.3: GP posterior after termination of the exploration with MF-ES. The evaluations on the simulator (light blue) are systematically below the evaluations on the real system (dark blue). This bias is captured by the GP model assuming a lower prior mean for the simulator data, as mentioned in Sec. 4.4.3. The posterior mean (green surface) and ± 2 std (gray surface) predict the underlying cost function of the real system, conditioned on the observed data from both simulator and experiments. The best guess location for the global minimum, x^{BG}, is represented by the orange dot.

The noise standard deviation of an evaluation on the real system, as defined in (2.3), has been estimated to $\sigma_{\mathrm{exp}} = 2.08 \times 10^{-4}$, while the noise of the simulator has been set to $\sigma_{\mathrm{sim}} = 10^{-5}$, roughly twenty times lower.

We stop the exploration when the GP posterior mean at the best guess x^{BG} (i.e., the current estimate of the global minimum) has not changed significantly (within a range of $\sigma_{\mathrm{err}}/4$ over the last 3 iterations), and we are sufficiently certain about its value (posterior standard deviation at x^{BG} less than $\sigma_{\mathrm{err}}/2$). Once the exploration has terminated, we evaluate the final best guess controller on a physical experiment and take its cost as the outcome of the learning procedure.

4.4.4 Results

A video summary describing the method and the obtained results can be found at https://youtu.be/oq9Qgq1Ipp8. We run MF-ES on the LQR problem described in Sec. 4.4.2. Fig. 4.3 shows the final GP cost function landscape after the learning procedure, highlighting simulations (in light blue) and experiments (in dark blue).

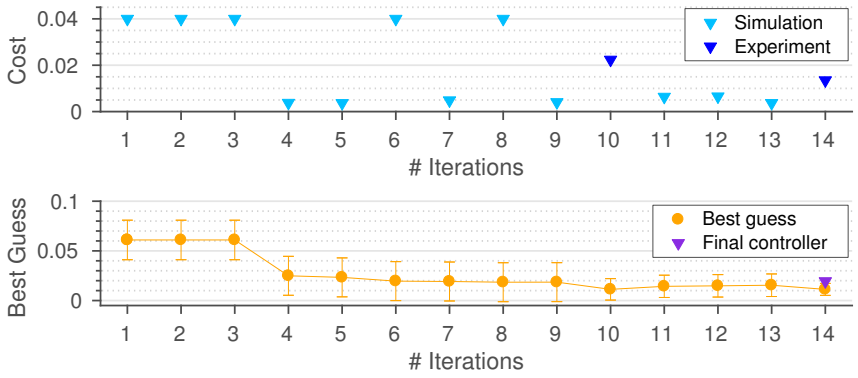


Figure 4.4: (Top) Cost obtained at each iteration with the proposed approach during one exploration run. When the exploration terminates, the best guess is evaluated on the physical system (violet dot). (Bottom) Evolution of the GP posterior mean at the best guess $\mu(x^{\mathrm{BG}})$ and std $\pm\sigma(x^{\mathrm{BG}})$.

For the same learning run, Fig. 4.4 (top) illustrates how MF-ES alternates between simulations and physical experiments over iterations. As can be seen, the algorithm first performs multiple cheap simulations, which allow to identify regions of unstable controllers (i.e., regions of high predicted cost in Fig. 4.3) without any real experiment. At iterations 10 and 14, the algorithm demands two expensive physical experiments. The reason is that a time unit spent in simulation is expected to be less informative than on a physical experiment. Thereby, experiment time should be better spent on the physical system. Fig. 4.4 (bottom) shows the GP posterior mean and standard deviation of the best guess at each iteration. The stopping criterion terminates the exploration after 14 iterations because the GP posterior mean of the last three best guesses were steady enough. Finally, the algorithm selects the last global minimum, $x^{\mathrm{BG}} = [0.212,\ 2.42]$ (orange dot in Fig. 4.3), as the final controller, which was evaluated on the physical system retrieving a low cost $f(x^{\mathrm{BG}}) = 0.0194$.

As a remark, we observe that the algorithm alternates between simulations and experiments in a non-trivial way, which cannot be reproduced with a simple two-stage learning process, where simulations are used to seed experimental reinforcement learning. Furthermore, in Fig. 4.3, we can see that the posterior mean around $x = [-2, 4]$ falls back to the prior in the absence of evaluations. As pointed out in Sec. 4.4.3, the prior mean is pessimistic in the sense that predicts instability in unforeseen areas, which is a reasonable assumption in controller tuning of real systems.

In order to illustrate the benefit of trading off data from *experiments and simulations*, we compare MF-ES to ES (Hennig and Schuler, 2012), which uses *only physical experiments*. The latter corresponds to the automatic controller tuning setting in Sec. 3.3. We run each of these methods ten times on the controller tuning

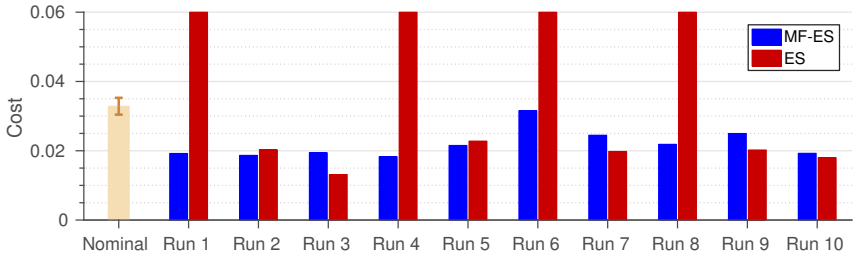


Figure 4.5: Comparison of the final controller cost at each run between the proposed approach (MF-ES) and ES. The cost of the nominal controller (beige) with ± 2 std is shown for reference.

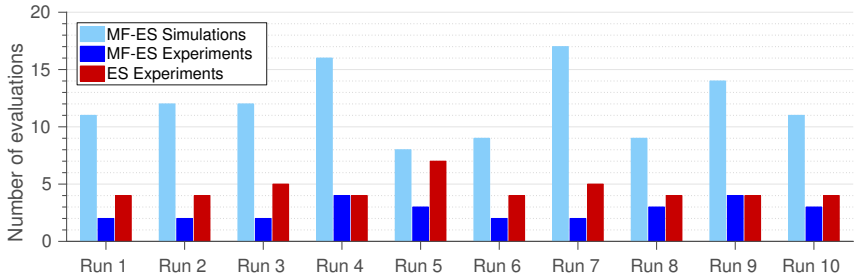


Figure 4.6: Number of physical experiments at each run for MF-ES (dark blue) and standard ES (dark red), as well as simulations for MF-ES (light blue).

problem. The results are discussed in Fig. 4.5 and Fig. 4.6.

In Fig. 4.5, we show the cost of the final controller at each run, for both methods. The cost of the nominal controller (green) is shown as a reference. MF-ES finds controllers that are 33.23% better, on average. Moreover, it consistently finds stabilizing controllers, while ES fails to find a stabilizing solution in 4 out of 10 cases (cost of 0.06).

Fig. 4.6 compares the number of physical experiments performed with MF-ES (dark blue) and with ES (dark red). While ES needs on average 3.5 physical experiments, MF-ES needs 2.7 (22.86% less) plus 11.9 simulations. These results demonstrate that MF-ES can find, on average, better controllers with a lower number of real experiments by also leveraging information from simulations.

4.5 Conclusions

In this chapter, we have proposed an extension of Entropy Search (ES) to the multi-fidelity setting in order to adaptively select between multiple information sources with different accuracies and evaluation efforts. We learn the controller parameters of a real cart-pole system using our framework, which trades off between experiments on the physical plant and the simulator. The experimental results confirm that

using prior model information from a robot simulator can reduce the amount of data required to globally find good control policies.

We have also shown that the applicability of the automatic LQR tuning framework proposed in Sec. 3.3 easily extends to a setting with multiple information sources. Additionally, our extension of ES to the multi-fidelity setting (MF-ES) constitutes a novel contribution by itself.

We discuss below MF-ES in the context of related work posterior to its publication. Although the field has somewhat evolved, MF-ES remains relevant. Finally, we propose future ideas on how to extend and improve MF-ES.

4.5.1 Discussion and future lines

Although we have demonstrated the effectiveness of the proposed method (MF-ES) in a low dimensional parameter space, recent work has applied MF-ES to more complex real systems. For example, in (Rodriguez et al., 2018), simulations and real experiments are combined in order to learn gait stabilization parameters on a real bipedal robot.

In spite of its advantages, MF-ES comes with a few shortcomings. For example, the applicability of this method is restricted to knowing, or having a way to estimate, the effort involved in gathering evaluations from the different information sources. There could be cases in which the effort cannot be accurately estimated ahead in time, for example in finances, when assessing the monetary cost of assets needed for an investment. In those kind of settings, the trade off between information gain and effort might be changing over iterations, which poses an interesting research direction, worth to investigate. Additionally, the prior variance given to the simulator serves as a proxy to establish "how informative it is". Because the simulator does not change over iterations, the prior variance must stay constant. This can be seen as a drawback, as the flexibility of the GP model is not being fully exploited. A simple solution could be to obtain the simulator prior variance by scaling down the real cost variance with a fixed factor. Then, the prior variance of the real cost can be optimized, while the factor keeps reflecting "how informative" the simulator is with respect to the true function. Finally, such factor could be estimated a priori by (i) sampling a large number of experiments from both, the simulator and the real system, and (ii) computing the ratios of successful controllers on each case.

From a more high-level perspective, a plausible counterargument to this work naturally arises when applying it in settings where simulations are way cheaper than real evaluation. For instance, one may wonder why restricting ourselves to a pipeline of sequential real/simulated evaluations if simulations are essentially "for free". In these settings, the proposed method herein would appear useless, as one could (i)

conduct many simulations prior to evaluations on the real system, or (ii) conduct simulations in batches in parallel to the real experiment as it runs. In the following, we discuss why MF-ES is still useful in both cases.

While the first case might seem appealing, a plausible number of simulations grows exponentially with the dimensionality of the problem, which still leaves room for a method that alternates between simulations and real experiments on demand, like MF-ES.

For example, the authors Rai et al. (2018, 2019) describe a setting in which a walking controller for a biped robot is estimated using BO (specifically, expected improvement). They have access to a high-fidelity simulator and argue that simulations come at no cost, since they have access to infinite resources. Thus, they propose a kernel informed by the high-fidelity simulator, which is first learned off-line using samples from the simulator and then, used to learn a 9-dimensional controller on the real robot with BO. While this could seems a reasonable, they report that only 100,000 simulations were conducted, which implies a coarse grid of less than 4 points per dimension. In spite of the high-fidelity simulator, there are high volumes in the simulation space still unexplored. Furthermore, each simulation took non-negligible time to complete (i.e., five seconds each; a total of five days). In this type of setting, MF-ES has potential for improvement: It could be warm-started using the pre-computed informed kernel, and help to cover the unexplored chunks of volume in the simulator as it is demanded by the information/effort trade off.

In a similar context, (Antonova et al., 2018) propose to pre-learn an informed kernel using samples from a database of simulated grasping tasks. Then, such kernel was used to learn on the real system by randomly alternating with a non-informed kernel (e.g., squared exponential) to mitigate the bias. Herein, MF-ES could also be used to keep updating the informed kernel by collecting simulation evaluations.

Generally, poor simulators tend to bias the search, which casts doubt over the "information never hurts" argument. However, with MF-ES this argument still holds as we can explicitly state how much confidence we give to the simulator in the first place. As opposed to (Antonova et al., 2018), where the bias of the simulator needs to be explicitly mitigated, our approach automatically does so, which we believe poses a great benefit.

In the second case, MF-ES could be parallelized when it comes to query for the simulator using batch Bayesian optimization (Daxberger and Low, 2017; Wang et al., 2017, 2018a). Specifically, while the real experiment is being queried, BO keeps querying simulations in batches, which should speed up convergence. Toward this direction, some ideas from (Hernández-Lobato et al., 2016, Sec. 5) could be leveraged. The authors propose a framework to speed up the acquisition of a new

candidate in BO, when the experimental evaluation is sufficiently slow.

Finally, recent work (Wu et al., 2019) combines batch BO in the multi-fidelity setting. However, this would be impractical herein, as the robot experiments have to be conducted sequentially, i.e., we assume a single robot, rather than a robot farm.

4.5.2 Vision

There is an increasing interest in the robot learning community on including the simulator in the learning loop. This has been commonly addressed as "closing the reality gap" (Chatzilygeroudis et al., 2019; Mirletz et al., 2015; Zhu et al., 2018). Despite of this recent interest, one may wonder whether this is a trend, or if it is meant to stay over time.

Nowadays, high-fidelity simulators are becoming more present and more accurate. With the increasing growth in computational power, it is unavoidable to think, *could there be a point in time in which simulators are as informative as the real robot experiments?*. Indeed, if that was the case, then, algorithms like MF-ES and much of the aforementioned research would become obsolete. However, there are two aspects to keep in mind. First, it is clear that no robot model, as good as it could be, can perfectly match reality, even if infinite computational power is available. Second, if robots are ever functioning among humans, they will constantly need to adapt their controllers and their dynamics to new situations. This is due to changes in the environment, but also possible changes in their morphology (e.g., slight loosening of articulations, deformities caused by impacts over time, etc). Building and programming those robots with a certain degree of adaptability and robustness is definitely desirable from the user standpoint, to avoid frequent reparations. That said, algorithms like MF-ES, that account for a mixture between fresh data acquired from the environment and a pre-established simulator (more or less accurate), will keep having a place in future developments in robot learning.

Chapter 5

On the design of LQR kernels

In this chapter, we present ideas from our conference publication (Marco et al., 2017). A video recording of the presentation at the conference can be found at https://youtu.be/zsC6Lufkl_E.

The ideas discussed herein build upon the automatic LQR tuning framework introduced in Chap. 3. Therein, Bayesian optimization (BO) (Sec. 2.3) is proposed as a powerful framework for direct controller tuning from experimental trials on nonlinear dynamic systems. To model the performance objective function that encodes the tuning performance, BO typically relies on Gaussian processes (GPs) (Sec. 2.2). In Chap. 3 and 4, the used GP models are described with a non-informative kernel. This can, however, lead to poor learning outcomes on standard quadratic control problems. For a first-order system, we construct two kernels that specifically leverage the structure of the well-known Linear Quadratic Regulator (LQR), yet retain the flexibility of Bayesian nonparametric learning. Simulations of uncertain linear and nonlinear systems demonstrate that the LQR kernels yield superior learning performance.

This chapter is structured as follows[1]. First, we introduce the problem and the contributions in Sec. 5.1. Second, we introduce the considered learning control problem in Sec. 5.2. Third, we use the BO framework for learning control (cf. Sec. 2.3) in the special case of a scalar problem and develop the LQR kernels in Sec. 5.3. Numerical results in Sec. 5.4 illustrate the improved learning performance of the proposed kernels over a standard kernel. The chapter concludes with remarks in Sec. 5.5.

[1] For an extensive literature review see Sec. 1.5.4

5.1 Introduction

While BO provides a promising framework for learning controllers in fairly general settings, the full power of Bayesian learning is often not exploited. A key advantage of Bayesian methods is that they allow for combining prior problem knowledge with learning from data in a principled manner. In case of GP models, this concerns specifically the choice of the *kernel*, which captures the covariance between function values at different inputs and is thus the core component to model prior knowledge about the function shape. By choosing standard kernels, however, naive BO approaches do often not exploit this opportunity to improve learning performance.

In this chapter, we show how structural knowledge about the optimal control problem at hand can be leveraged for designing specific kernels in order to improve data efficiency in learning control. For a first-order nonlinear quadratic optimal control problem, we propose two *LQR kernels* that leverage the structure of the famous Linear Quadratic Regulator (LQR) problem given in form of an *approximate linear model* of the *true nonlinear dynamics*. The proposed kernels leverage the structure of the LQR problem, while retaining the flexibility of nonparametric GP regression. In detail, we state below the contributions of this work:

1) We discuss how the structure of the well-known LQR problem can be leveraged for efficient learning of controllers for nonlinear systems.
2) This discussion leads to the proposal of two new kernels for GP regression in the context of learning control: the *parametric* and *nonparametric LQR kernels*.
3) The improved learning performance achieved with these kernels over a standard kernel is demonstrated through numerical simulations.

Notation: A Gaussian random variable z with mean m and variance V is denoted by $z \sim \mathcal{N}(m; V)$. The expected value of x is denoted by $\mathbb{E}[x]$, while $\mathbb{V}[x_1, x_2]$ denotes the covariance of x_1 and x_2. We also use $\mathbb{V}[x_1] := \mathbb{V}[x_1, x_1]$.

5.2 Learning Control Problem

The goal of this section is to introduce the controller tuning problem presented in Chap. 2. More specifically, we take the variant presented in Sec. 3.2. As therein, we consider a nonlinear stochastic system with unknown dynamics

$$s_{k+1} = h(s_k, u_k, v_k) \tag{5.1}$$

with state $s_k \in \mathbb{R}^S$, control input $u_k \in \mathbb{R}^U$, and random noise $v_k \sim \mathcal{N}(0; V)$. We assume that the state s_k can be measured or otherwise estimated. The control

objective is to find a state-feedback controller

$$u_k = F s_k \tag{5.2}$$

with controller gain $F \in \mathbb{R}^{U \times S}$ such that the quadratic cost

$$f = \lim_{M \to \infty} \frac{1}{M+1} \mathbb{E}\left[\sum_{k=0}^{M} s_k^\top Q s_k + u_k^\top R u_k\right] \tag{5.3}$$

with symmetric weights $Q \geq 0$ and $R > 0$ is minimized. A quadratic cost is a very common choice in optimal control (Anderson and Moore, 1990) expressing the designer's preference in the fundamental trade-off between control performance and control effort.

While the solution of the above problem is standard when the dynamics (5.1) are known and linear (Anderson and Moore, 1990), here, we face the problem of optimal control under *unknown* and *nonlinear* dynamics.

To address this problem, we use the automatic controller tuning approach proposed in Chap. 2. For this, we parametrize the control gain F with parameters $x \in \mathcal{X}$. Instead of the indirect approach proposed in Sec. 3.3, and also used in Sec. 4.4.2, which parametrizes the weights of a linear quadratic regulator (LQR) to compute F, we take here a direct parametrization $x = \mathrm{vec}\,(F)$. In order to find the optimal parameters x_*, we use Bayesian optimization (BO) (Sec. 2.3) to search the optimum of (5.3) by directly evaluating the performance of candidate controllers x on the real system (5.1).

The dependence of the cost function f (5.3) on x is unknown *a priori* because of the lack of knowledge of the of the system (5.1). We model f using a Gaussian process (GP) $f \sim \mathcal{GP}\,(m(x), k(x, x'))$. We use the same notation introduced in Sec. 2.2.

Furthermore, in contrast to Chap. 3 and 4, the BO algorithm used herein is *expected improvement* (EI) (Jones et al., 1998; Mockus et al., 1978).

5.3 Constructing LQR Kernels

In this section, we present two kernels that are specifically designed for the control problem of Sec. 5.2 and incorporate approximate model knowledge in form of a linear approximation to (5.1). Because for a linear system, the solution to the optimal control problem of Sec. 5.2 is the well-known LQR, we term these kernels *LQR kernels*. The following derivations are presented for a first-order system (5.1), where all variables are scalars ($S = U = 1$). Small letters are used in place of the capital ones to emphasize scalar variables (e.g., v instead of V in (5.1)). We consider

minimization of the cost (5.3), which is rewritten as

$$f = \lim_{M\to\infty} \frac{1}{M+1}\mathbb{E}\left[\sum_{k=0}^{M} qs_k^2 + ru_k^2\right]. \tag{5.4}$$

We start by considering a learning example with a standard kernel choice for the GP, which motivates why a specifically designed kernel can be desirable.

5.3.1 Problems with standard kernel

The most common kernel choice in GP regression is arguably the squared exponential (SE) kernel k_{SE} (2.5). Let us consider the problem of learning the cost function f (5.4) via GP regression with SE kernel for the following linear example:

Example 1 *Let* (5.1) *be given by* $s_{k+1} = 0.9s_k + u_k + v_k$ *with* $v_k \sim \mathcal{N}(0;1)$.

Fig. 5.1 shows the prior GP and the posterior GP after obtaining four data points (i.e., after four evaluations of controllers x_i). A few issues are apparent from the posterior: (i) the kernel has problems with the different length scales of the function (f is steep around $x = -0.1$, but rather flat in the center region); (ii) the GP does not generalize well to regions where no data has been seen (e.g., around $x = -1.6$, where the posterior mean resorts to the prior); and (iii) the overall fit is not very good.

Clearly, the fit will improve with more data, but, for efficient and fast learning of controllers, we are particularly interested in good fits from just few data points. Hence, we seek to improve the fitting properties of the GP by exploiting knowledge about the system (5.1) in terms of an approximate linear model.

5.3.2 Incorporating prior knowledge

In no practical situation, one has a perfect model of the system to be controlled. At the same time, it is often possible to obtain a rough model, e.g., from first principles modeling or some system identification procedure. Here, we assume that an uncertain linear model

$$s_{k+1} = as_k + bu_k + v_k \tag{5.5}$$

$$a \in [a_{\min}, a_{\max}], b \in [b_{\min}, b_{\max}] \tag{5.6}$$

is available as an approximation to (5.1), e.g., from linearization of a first principles model with (possibly) some uncertainty about the physical parameters.

In the following, we will consider controller gains f such that the system (5.5) is guaranteed stable for all parameters (5.6). That is, we consider $x \in \mathcal{F}$ with

$$\mathcal{F} := \{x \in \mathbb{R} \,|\, |a + bx| < 1$$

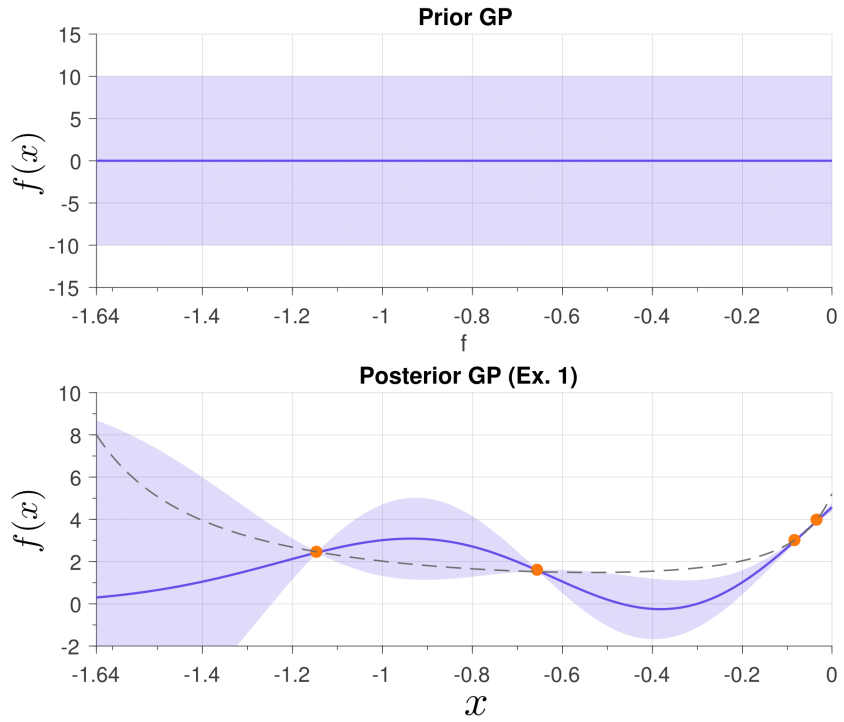


Figure 5.1: Prior and posterior GPs of Example 1 using the squared exponential kernel k_{SE} (hyperparameters: $\sigma_{\text{S}}^2 = 25$, $\lambda = 0.4$). The thick line represents the GP mean, and the light blue area the GP variance (+/- two standard deviations). The true function is shown in the bottom plot in dashed gray, and data points in orange.

$$\forall a \in [a_{\min}, a_{\max}], b \in [b_{\min}, b_{\max}]\}. \tag{5.7}$$

This restriction makes sense, for example, in safety critical applications, where one wants to avoid the risk of trying an unstable controller based on the system knowledge available (i.e., (5.5), (5.6)). Moreover, the restriction to $\mathcal{F}$ will ensure that subsequent calculations are well-defined.

If a and b were known, the functional dependence of the cost J on the controller gain f in (5.2) for the linear system (5.5) could be derived using standard control theory.

Fact 1 *Consider the system* (5.5) *with known parameters* a *and* b*, and let* $|a+bx| < 1$*. Then, the cost* (5.4) *is given by*[2]

$$f = v\frac{q + rx^2}{1 - (a + bx)^2} =: \phi_{(a,b)}(x). \tag{5.8}$$

Proof The controlled process $s_{k+1} = (a + bx)s_k + v_k$ is stable by assumption and thus converges to a stationary process with zero mean and variance $\mathbb{E}[s_k^2] = P$,

[2]In the notation $\phi_{(a,b)}(x)$, we omit the parametric dependence on v, q and r since v is a multiplicative constant, which does not play a role in the later optimization, and we assume that q and r are fixed.

where P is the unique positive solution to $P - P(a+bx)^2 = v$, (Anderson and Moore, 2005, Theorem 3.1). For stationary s_k, (5.4) resolves into

$$f = \mathbb{E}\left[qs_k^2 + ru_k^2\right] = (q + rx^2)\mathbb{E}\left[s_k^2\right] = (q + rx^2)P.$$

Equation (5.8) then follows. ■

If we are uncertain about a and b, a collection of possible costs (5.8) emerge from all the possible combinations of a and b within their ranges (5.6). We assume this collection of costs to be explained by a Gaussian process f_{LQR}. While any arbitrary choice of a and b in (5.5) is only an approximation to (5.1), it yields a cost (5.8) that contains useful structural information, which can be leveraged for faster learning controllers from data. In the next sections, we show how this prior knowledge can be exploited to construct LQR kernels.

5.3.3 Parametric LQR kernel

A reasonable choice for $f_{\text{LQR}}(x)$ is

$$f_{\text{LQR}}(x) = w\,\phi_{(\bar{a},\bar{b})}(x), \quad w \sim \mathcal{N}(\bar{w}; \sigma_w^2) \tag{5.9}$$

where $\bar{a} := (a_{\min} + a_{\max})/2$ and $\bar{b} := (b_{\min} + b_{\max})/2$ are the midpoints of the uncertainty intervals (5.6).

Equation (5.9) is a standard parametric model with a single feature $\phi_{(\bar{a},\bar{b})}(x)$ and Gaussian prior. It is well-known (see e.g. Rasmussen and Williams (2006)) that $f_{\text{LQR}}(x)$ is a GP,

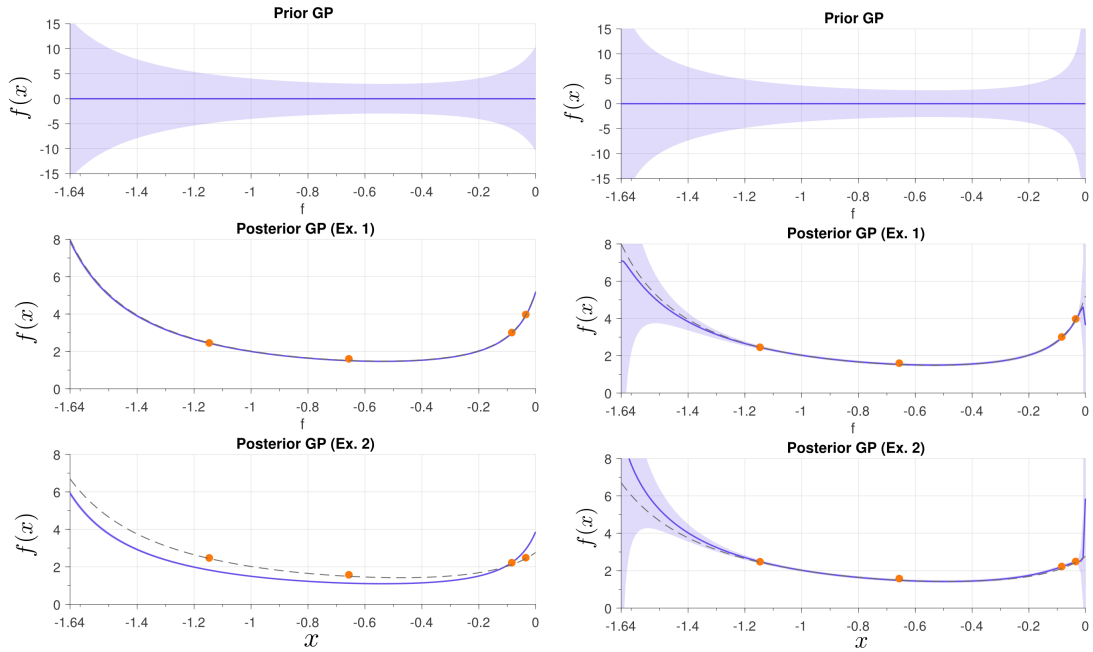
$$f_{\text{LQR}}(x) \sim \mathcal{GP}\left(0, k_{\text{pLQR}}(x, x')\right) \tag{5.10}$$

where we have assumed $\bar{w} = 0$, with kernel

$$\begin{aligned} k_{\text{pLQR}}(x, x') &:= \sigma_w^2 \phi_{(\bar{a},\bar{b})}(x)\phi_{(\bar{a},\bar{b})}(x') \\ &= \sigma_{\text{p}}^2 \frac{v^2(q + rx^2)(q + rx'^2)}{(1 - (\bar{a} + \bar{b}x)^2)(1 - (\bar{a} + \bar{b}x')^2)} \end{aligned} \tag{5.11}$$

and hyperparameters $\sigma_{\text{p}} := \sigma_w$, $\bar{a}$, and $\bar{b}$. We refer to (5.11) as the *parametric LQR kernel* because it captures the cost function for the linear system $(\bar{a}, \bar{b})$ with quadratic cost, and thus the structure of the LQR problem.

To illustrate the performance of the parametric LQR kernel k_{pLQR}, we revisit Example 1 using this kernel instead of the SE kernel. The top two graphs of Fig. 5.2a show the prior and posterior GP for the same data points as in Fig. 5.1 when using k_{pLQR} with hyperparameters $\bar{a} = 0.9$ and $\bar{b} = 1$. We see that the posterior fit from



(a) Parametric LQR kernel (b) Nonparametric LQR kernel

Figure 5.2: (a) GP fit using the parametric LQR kernel k_{pLQR} (hyperparameters: $\sigma_{\text{p}}^2 = 1$, $\bar{a} = 0.9$, $\bar{b} = 1$). The color code is the same as in Fig. 5.1. The hyperparameters are exact for Example 1, while they are off by about 10% for Example 2. (b) GP fit using the nonparametric LQR kernel k_{nLQR} (hyperparameters: $\sigma_{\text{n}}^2 = 20$, $a_{\min} = 0.8$, $a_{\max} = 1.0$, $b_{\min} = 0.9$, and $b_{\max} = 1.1$). Colors are the same as in Fig. 5.1. Both examples are fitted well.

only four data points is almost perfect and much better than the one in Fig. 5.1. This is, of course, not a big surprise because the hyperparameters match the true underlying system of Example 1 perfectly. The fitting performance deteriorates, however, if the hyperparameters are off. This can be seen in the bottom graph of Fig. 5.2a, which shows the posterior GP with the same hyperparameters, but for the system

Example 2 $s_{k+1} = 0.8s_k + 0.9u_k + v_k$ *with* $v_k \sim \mathcal{N}(0; 1)$.

To improve the fit in this situation, one can use hyperparameter optimization (see Sec. 2.2.2) to find improved parameters $\bar{a}$ and $\bar{b}$ that better explain the data. The simulation results in Sec. 5.4 will show that this can be a viable approach. An alternative is to design more flexible and expressive kernels, which allow for fitting more general models. This we discuss next.

5.3.4 Nonparametric LQR kernel

The kernel (5.11) captures the structure of the cost function for the LQR problem with one specific linear model $(\bar{a}, \bar{b})$. A straightforward way to increase the flexibility of the kernel in order to fit more general problems is to use m basis functions (or features) $\phi_{(a_i,b_i)}$ corresponding to m models $(a_1, b_1), (a_2, b_2), \ldots, (a_m, b_m)$,

$$\begin{aligned} f_{\mathrm{LQR}}(x) &= \underbrace{\begin{bmatrix}\phi_{(a_1,b_1)}(x) & \phi_{(a_2,b_2)}(x) & \cdots & \phi_{(a_m,b_m)}(x)\end{bmatrix}}_{=:\Phi^\top(x)} w \\ &= \Phi^\top(x) w \end{aligned} \tag{5.12}$$

with $w \in \mathbb{R}^m$, $w \sim \mathcal{N}(\bar{w}; \Sigma_w)$. The derivation of the corresponding kernel is analogous to (5.11) (see (Rasmussen and Williams, 2006, Sec. 2.7)) and yields

$$k_{\mathrm{pLQR},m}(x, x') = \Phi^\top(x)\, \Sigma_w\, \Phi(x'). \tag{5.13}$$

Same as (5.9), the model (5.12) represents a parametric model for the LQR cost f_{LQR}. That is, its flexibility is essentially limited to the number of explicit features $\phi_{(a_i,b_i)}$. Using powerful kernel techniques (Schölkopf and Smola, 2002), the parametric model can be turned into a nonparametric one, which includes an infinite number of features while retaining finite computational complexity. The key idea is to consider the kernel (5.13) in the limit of infinitely many features corresponding to models $a \in [a_{\min}, a_{\max}]$ and $b \in [b_{\min}, b_{\max}]$. The derivation follows ideas similar to how the standard SE kernel can be derived from basic features, (Rasmussen and Williams, 2006, p. 84).

Consider the partitions of $[a_{\min}, a_{\max}]$ and $[b_{\min}, b_{\max}]$ into m equidistant intervals, and let $\{a_i\}_{1:m}$ and $\{b_i\}_{1:m}$ be the lower (or upper) interval limits. Consider the model (5.12) with feature vector

$$\Phi^\top(x) = \begin{bmatrix}\phi_{(a_1,b_1)}(x) \cdots \phi_{(a_i,b_j)}(x) \cdots \phi_{(a_m,b_m)}(x)\end{bmatrix}$$

which includes all combinations $\phi_{(a_i,b_j)}$ for $i, j \in \{1, \ldots, m\}$, and the parametric prior $\bar{w} = 0$ and $\Sigma_w = \frac{\sigma_{\mathrm{n}}^2 (a_{\max} - a_{\min})(b_{\max} - b_{\min})}{m^2} I$ for some $\sigma_{\mathrm{n}} \in \mathbb{R}$. The kernel (5.13) then becomes

$$\begin{aligned} k_{\mathrm{pLQR},m}(x, x') = &\frac{\sigma_{\mathrm{n}}^2 (a_{\max} - a_{\min})(b_{\max} - b_{\min})}{m^2} \\ &\times \sum_{j=1}^{m} \sum_{i=1}^{m} \phi_{(a_i,b_j)}(x)\, \phi_{(a_i,b_j)}(x'). \end{aligned} \tag{5.14}$$

Since $\phi_{(a_i,b_j)}$ is continuous on $\mathcal{F}$, the finite sum (5.14) converges to the Riemann integral in the limits as $m \to \infty$. We can thus define the *nonparametric LQR kernel*

$$k_{\mathrm{nLQR}}(x, x') = \lim_{m \to \infty} k_{\mathrm{pLQR},m}(x, x')$$

$$= \sigma_{\mathrm{n}}^2 \int_{b_{\min}}^{b_{\max}} \int_{a_{\min}}^{a_{\max}} \phi_{(a,b)}(x)\phi_{(a,b)}(x')\, da\, db \tag{5.15}$$

for $x, x' \in \mathcal{F}$ with the signal variance σ_{n}^2 and the integration boundaries $a_{\min}$, $a_{\max}$, $b_{\min}$, and $b_{\max}$ as hyperparameters. While the kernel in (5.15) represents the structure of the cost function (5.8) for an infinite number of models (all $a \in [a_{\min}, a_{\max}]$ and $b \in [b_{\min}, b_{\max}]$), its computation is finite consisting of solving the integral in (5.15). By contrast, the computational complexity of the parametric kernels (5.13) and (5.14) grows with the number of features m. We prove that the kernel (5.15) is indeed a valid covariance function.

Proposition 2 *$k_{nLQR}(x, x')$ is positive semidefinite for all $x, x' \in \mathcal{F}$.*

Proof Take any collection $\{x_1, x_2, \ldots, x_N\}$ and any $z \in \mathbb{R}^N$. Let K_N be the Gram matrix for the kernel k_{nLQR}. Then

$$\begin{aligned}
z^\top K_N z &= \sum_{j=1}^{N} z_j \Big(\sum_{i=1}^{N} z_i k_{\mathrm{nLQR}}(x_i, x_j) \Big) \\
&= \sigma_{\mathrm{n}}^2 \sum_{j=1}^{N} z_j \sum_{i=1}^{N} z_i \int_{b_{\min}}^{b_{\max}} \int_{a_{\min}}^{a_{\max}} \phi_{(a,b)}(x_i)\phi_{(a,b)}(x_j)\, da\, db \\
&= \sigma_{\mathrm{n}}^2 \int_{b_{\min}}^{b_{\max}} \int_{a_{\min}}^{a_{\max}} \sum_{j=1}^{N} z_j \phi_{(a,b)}(x_j) \sum_{i=1}^{N} z_i \phi_{(a,b)}(x_i)\, da\, db \\
&= \sigma_{\mathrm{n}}^2 \int_{b_{\min}}^{b_{\max}} \int_{a_{\min}}^{a_{\max}} \Big(\sum_{i=1}^{N} z_i \phi_{(a,b)}(x_i) \Big)^2 da\, db \geq 0
\end{aligned}$$

which completes the proof. ■

The above derivation corresponds to the *kernel trick* Rasmussen and Williams (2006); Schölkopf and Smola (2002), which is a core idea of kernel methods and behind many powerful learning algorithms. In essence, the kernel trick means to write a learning algorithm solely in terms of inner products of features and replacing those by a kernel[3]. In particular, this allows for considering an infinite number of features, while retaining finite computation.

Figure 5.2b shows the prior and posterior GP for the nonparametric LQR kernel (5.15). Because the kernel is more flexible than (5.11), it can fit the cost functions for the two different models of Example 1 and Example 2 well.

[3] All computations for the GP regression involved in obtaining the predictive distribution (2.4) are expressed solely in terms of the kernel k (and the prior mean function).

5.3.5 A combined kernel

System (5.5) is an approximation to the true system (5.1). It is thus not desirable to fully commit to the model for solving the optimal control problem. This would mean to directly minimize (5.8) (which would result in the well-known LQR solution). On the other hand, the linear problem contains information about the structure of the optimization problem, which shall be useful also for optimization of the true nonlinear system (5.1), as long as we believe (5.5) to be a reasonable approximation thereof. In other words, we can expect the true cost function J for the nonlinear problem to bear some similarity to (5.8).

We model the cost function (5.3) for the nonlinear system (5.1) as being composed of a part that stems from the approximation as LQR problem and an error term,

$$f(x) = f_{\mathrm{LQR}}(x) + f_{\Delta}(x). \tag{5.16}$$

The term $f_{\Delta}(x)$ captures the error in the model that stems from the fact that the true problem is nonlinear. We model it as a standard GP, e.g., using the SE kernel (2.5): $f_{\Delta}(x) \sim \mathcal{GP}\left((,0)\,, k_{\mathrm{SE}}(x, x')\right)$. We can model $f_{\mathrm{LQR}}(x)$ as a GP (5.10) using either the parametric LQR kernel (5.11) or the nonparametric (5.15) LQR kernel. Then, since the sum of two independent Gaussians is also Gaussian, it follows from (5.16) that also f is a GP (see (Rasmussen and Williams, 2006, Sec. 2.7)), with

$$f(x) \sim \mathcal{GP}\left(0, k_{\mathrm{LQR}}(x, x') + k_{\mathrm{SE}}(x, x')\right)$$

where k_{LQR} can be replaced by (5.11) or (5.15). By choosing the hyperparameters of the kernels, the designer can express how much he or she trusts the LQR versus the SE model. For example, $\sigma_{\mathrm{S}} = 0$ means to fully rely on the LQR kernel.

5.4 Simulations

In this section, we show statistical comparisons of the LQR kernels proposed in Sec. 5.3 against the commonly used SE kernel, in two different settings. In the first setting, we evaluate the performance of each kernel in the context of GP regression. Specifically, we quantify the mismatch between the GP posterior mean, computed from a set of random evaluations, and the underlying cost function. In the second setting, we evaluate each kernel in the context of BO by comparing the learned minimum to the true global minimum.

The GP regression and BO experiments are presented in Sec. 5.4.2 and Sec. 5.4.3, respectively, considering a linear system (5.1). In addition, we also evaluate the BO setting for a nonlinear system in Sec. 5.4.4.

5.4.1 Experimental choices

For the simulations in Sec. 5.4.2 and Sec. 5.4.3, we consider the true system (5.1) to be linear (i.e., (5.5)) with uncertain parameters $a \in [0.8, 1.0]$ and $b \in [0.9, 1.1]$. We consider the optimal control problem as in Sec. 5.2 with $q = r = v = 1$. Feedback controllers (5.2) are considered to be in the range (5.7), $\mathcal{F} = [-1.64, -0.001]$.

For each controller x, the corresponding infinite-horizon LQR cost $f(x)$ is given by (5.8). In practice, only finite-horizon simulations can be realized. Therefore, the outcome of an experiment is noisy, as modeled in (2.3), with $\sigma = 0.05$.

In each simulation, a different linear model (a, b) is obtained by uniformly sampling the ranges above, which yields a different underlying cost function (5.8). Each simulation is repeated for four different kernels:

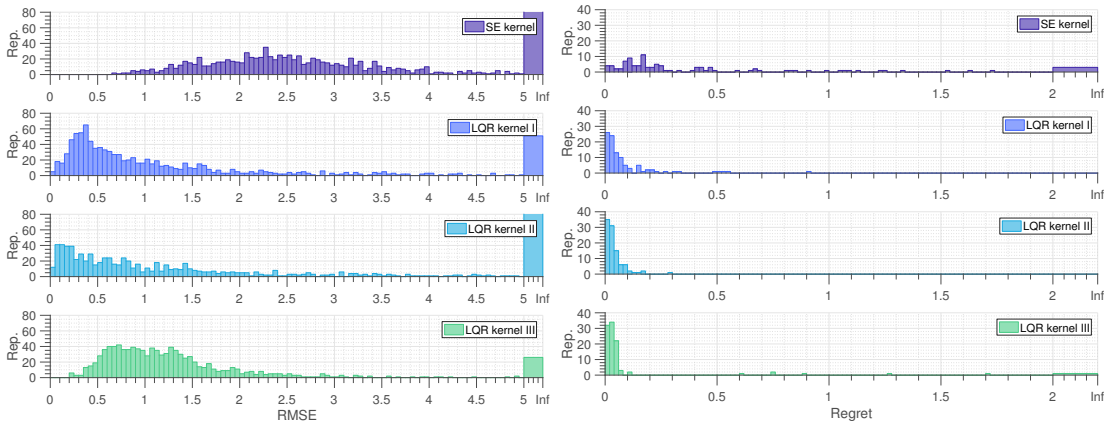
- **SE kernel**: As described in (2.5).
- **LQR kernel I**: Parametric LQR kernel (5.11) with fixed parameters $(\bar{a}, \bar{b})$ (midpoints of uncertainty intervals).
- **LQR kernel II**: Parametric LQR kernel (5.11), with $(\bar{a}, \bar{b})$ optimized from evaluations by maximizing (2.7).
- **LQR kernel III**: Nonparametric LQR kernel (5.15).

The parametric LQR kernel (5.11) is constructed taking the middle points of the uncertainty intervals, i.e., $\bar{a} = 0.9$ and $\bar{b} = 1$. For the nonparametric LQR kernel (5.15), the intervals serve as integration domains. The length scale of the SE kernel is computed as one fifth of the input domain, i.e., $\lambda = 0.327$. The signal variance of the parametric and nonparametric LQR kernel, i.e., σ_{p}^2 and σ_{n}^2, are normalized such that $k_{\mathrm{pLQR}}(\bar{x}, \bar{x}) = k_{\mathrm{nLQR}}(\bar{x}, \bar{x}) = 10$, where $\bar{x} = -0.82$ is the midpoint of $\mathcal{F}$. Since the variance of these two kernels grow fast toward the corners of the domain, we set for the SE kernel $\sigma_{\mathrm{S}}^2 = 50$, for a fair comparison.

5.4.2 GP regression setting

For this statistical comparison, we run 1000 simulations. In each simulation, we compute the GP posterior conditioned on two evaluations randomly sampled from the underlying cost function. We assess the quality of the regression by computing the root mean squared error (RMSE) between the true cost function and the GP posterior mean, both computed on a grid of 100 points over $\mathcal{F}$.

Fig. 5.3a shows the histograms of the RMSE obtained with the different kernels. The LQR kernels clearly outperform the SE kernel in these experiments because they contain structural knowledge about the true cost, which contributes to a better GP fit, even with only two data points. The nonparametric kernel makes good predictions because it inherently contains information about all possible functions within



(a) RMSE (b) Regret

Figure 5.3: (a) Histograms of the root mean squared error (RMSE) between the true cost function and the GP posterior mean conditioned on two evaluations. The histograms are computed out of 1000 simulations. (b) Histogram of the regret incurred by stopping BO after three evaluations. The system considered is the linear system (5.5). The histogram is computed over 100 BO runs.

their uncertainty ranges of (a, b). However, poorly specified integration bounds will decrease its performance. The RMSE statistical analysis confirms this when the integration intervals on (a, b) are a 50% larger than their uncertainties. The parametric kernel with fixed hyperparameters $(\bar{a}, \bar{b})$ has a significant number of outliers since, in many cases, the data is queried from a cost function whose sampled (a, b) are far away from the ones of the kernel. The parametric kernel with hyperparameter optimization also leads to a better fit than SE, but has most outliers (about 75% of the simulations) since hyperparameter optimization is not reliable with just two data points.

We have repeated these experiments with $N = 1$, 5, and 10 evaluations. Table 5.1 shows the averaged RMSE over 1000 simulations for each kernel, and the corresponding standard deviation (in parentheses). The the outliers (i.e., any RMSE above 5) were excluded from these computations. In general, we see that the LQR kernel optimized from data performs better than the others for more than 2 evalu-

Table 5.1: RMSE averaged over 1000 simulations.

N	SE ker.	SE ker. (*)	LQR ker. I	LQR ker. II	LQR ker. III
1	2.76 (0.74)	2.39 (0.81)	**1.01** (0.95)	1.98 (1.29)	1.31 (0.71)
2	2.49 (0.85)	2.42 (0.93)	**1.02** (0.96)	1.09 (1.07)	1.22 (0.67)
5	1.83 (1.02)	1.31 (0.95)	1.10 (1.05)	**0.45** (0.70)	1.20 (0.76)
10	1.13 (0.99)	0.98 (0.86)	0.98 (0.96)	**0.20** (0.46)	1.11 (0.74)

ations.

For a fair comparison, we also include the SE kernel with hyperparameter optimization in the table (marked with an asterisk). Because it performs similar to the SE kernel, and it does not improve upon the LQR kernels, we leave it out of the discussion for the rest of the chapter.

Remark: The hyperparameters (a, b) of the parametric LQR kernel are optimized from data by maximizing the marginal likelihood (2.7). In a sense, optimizing the hyperparameters (a, b) of the LQR kernel from data can be considered similar to doing system identification on the linear system (a, b) using (2.7) as performance metric.

5.4.3 Bayesian optimization setting

In this section, we evaluate the performance of each kernel in the context of BO. For each BO run, the first evaluation is decided randomly within the range of controllers $\mathcal{F}$. Subsequent evaluations are acquired using expected improvement (EI) (Jones et al., 1998; Mockus et al., 1978). We stop the exploration after three evaluations and compute the instantaneous regret (i.e., the absolute error between the true minimum and the minimum of the GP posterior mean) as the outcome of each experiment.

Fig. 5.3b shows the histogram of the regret for each kernel over 100 BO runs. The LQR kernels consistently outperform the SE kernel. The nonparametric kernel shows some outliers because in some cases the GP posterior mean grows large toward negative values, and so does its minimum. This is a wrong prediction of the underlying cost, defined positive. However, this minor issue can easily be detected, and the optimization procedure continued with a randomly sampled controller.

5.4.4 Bayesian optimization setting for a nonlinear system

In this section, we use the same BO setting as in Sec. 5.4.3, but consider now a nonlinear system (5.1), namely

$$s_{k+1} = \tilde{a} \sin(s_k) + \tilde{b} u_k + v_k \tag{5.17}$$

with $v_k \sim \mathcal{N}(0; 1)$ and uncertain parameters $\tilde{a} \in [0.9, 1.1]$, $\tilde{b} \in [0.9, 1.1]$. We control this system from $x_0 = 1$ to zero, using the same controller structure as the one described in Sec. 5.4.1. In this case, the considered range of controllers is $\mathcal{F} = [-1.57, -0.27]$, which corresponds to (5.7), reduced by a 20%. The LQR kernels are built up using the linearized version of (5.17) around the zero equilibrium point, i.e., $s_{k+1} = \tilde{a} s_k + \tilde{b} u_k + v_k$. Table 5.2 shows the regret average and standard deviation. As can be seen, the LQR kernels perform better than the SE kernel.

Table 5.2: Regret averaged over 100 BO runs for a nonlinear system

N	SE kernel	LQR kernel I	LQR kernel II	LQR kernel III
2	1.34 (0.33)	**0.30** (0.21)	0.32 (0.20)	0.35 (0.18)
3	0.49 (0.39)	0.33 (0.38)	**0.31** (0.19)	0.32 (0.18)
4	0.35 (0.27)	0.36 (0.44)	0.32 (0.18)	**0.32** (0.17)
5	0.36 (0.21)	**0.31** (0.36)	0.32 (0.20)	0.32 (0.18)

5.5 Conclusions

In this chapter, we discussed how prior knowledge about the structure of an optimal control problem can be leveraged for data-efficient learning control. Specifically, for a nonlinear quadratic optimal control problem, we showed how an uncertain linear model approximating the true nonlinear dynamics can be exploited in a Bayesian setting. This led to the proposal of two novel kernels, a parametric and a nonparametric version of an LQR kernel, which incorporate the structure of the well-known LQR problem as prior knowledge. Numerical simulations herein demonstrate improved sample-efficiency over standard kernels, i.e., good controllers are learned from fewer experiments.

Approaching the nonlinear quadratic optimal control problem presented herein with pure model-based methods can lead to superior performance for very accurate models compared to the proposed data-based approach. However, this chapter shares the motivation of Chap. 3, where a data-based approach is proposed for the cases in which only poor models are available, and extends it by incorporating the LQR structure into the kernel.

We discuss below the proposed ideas on the design of LQR kernels in the context of related work posterior to its publication. In addition, we propose future ideas on how to extend LQR kernels to high-dimensional systems.

5.5.1 Discussion

There exist recent work where the kernel used for learning with BO is pre-computed using some form of prior knowledge. In the work from Rai et al. (2018, 2019), a kernel is designed using data acquired from an existing high-fidelity simulator, which reduces the number of real experiments required to optimize controller parameters. However, in lack of a high-fidelity simulator, constructing a kernel from simulated data can negatively bias the search, specially if the simulator model is poor. Instead, our approach does not use any given data to construct the kernels: We rely on the mathematical structure of the control problem at hand. The authors in (Wang et al.,

2018b) propose a meta-BO approach, in which the kernel structure is never imposed, but rather inferred from data. For this, they learn using the primal formulation of the GP. However, assumptions about the structure of the primal are needed (i.e., hyperparameters of the required basis functions). In addition, the unknown objective must correspond to its prior and large amounts of data are needed to learn the kernel. It is thus unclear how it would scale to learning on a high dimensional real robot, where data collection is scarce. Our approach is more tailored to the particular control problem at hand, and does not require data to be learned, i.e., its mathematical structure already speaks for the control problem.

In Sec. 5.3.5, we propose a kernel combination that could help to learn on non-linear systems. Therein, we propose to combine an informed kernel (i.e., the LQR kernel) with a non-informed kernel (e.g., squared exponential). A similar idea has been recently used in the context of learning from simulation data (Antonova et al., 2018), where an informed kernel is generated using simulated data and combined with a non-informed kernel. Contrary to this approach, our informed kernel does not require additional data to be constructed. Instead, it is built upon the mathematical structure of the control problem at hand.

As shown in this chapter, incorporating prior knowledge in a robot learning setting is possible, although it is non trivial. Specifically, when the control problem is LQR (i.e., linear model and quadratic cost), the nonparametric LQR kernel (5.15) cannot be analytically written, as it needs to be approximated with a quadrature (5.15). For a multivariate system, integrating out the linear model is unclear, and an interesting research direction.

5.5.2 Vision

The results shown in this chapter are preliminary in the sense that the LQR kernels are developed for a first-order system. While this is the natural first step, this framework could be potentially helpful also in multivariate systems. For example, it could be merged with the automatic LQR tuning framework proposed in Chap. 3 and used to learn controller parameters in more challenging real systems with imperfect state measurements, like the robot arm balancing an inverted pole and the humanoid robot doing squats (cf. Sec. 3.4). In addition, it could be helpful to increase data-efficiency during learning by using it in Chap. 4, which also relies on the aforementioned automatic LQR tuning framework.

Chapter 6

Controller learning under limited budget of failures

This chapter introduces ideas presented in our work (Marco et al., 2020), submitted for publication at a conference. Herein, we discus the treatment given to unstable controllers that may appear during the controller learning routine. This issue is raised in Sec. 3.7.2 in the context of automatic controller tuning in robotics. Therein, unstable controllers are seen as problematic because the robot has to be emergency-stopped and restarted before the learning routine can continue.

From a general perspective, the urge to avoid unstable (or unsafe) controllers is only exacerbated in safety-critical settings, where the cost of a failing controller usually has catastrophic consequences (e.g., life-threating situations in autonomous driving). In contrast, in other settings, like the robot learning experiments discussed in Chap. 3, failures have only mild consequences. Therefore, they can be seen as tolerable costs in exchange for a faster learning process, as they provide rich information about undesired behaviors.

In this chapter, we propose a novel decision maker in the context of Bayesian optimization under unknown constraints. Our method leverages the information provided by encountered failures while regulating the conservativeness of the search as a function of a pre-established budget of failures. Empirical validation shows that our algorithm uses the failures budget efficiently in a variety of optimization experiments, and generally achieves lower regret than state-of-the-art methods. In addition, we propose an original algorithm for unconstrained Bayesian optimization inspired by the notion of excursion sets in stochastic processes, upon which the failures-aware algorithm is built.

This chapter is structured as follows[1]. In Sec. 6.1, we introduce the problem and further elaborate on the contributions. In Sec. 6.2, we characterize excursion sets in

[1]For an extensive literature review see Sec. 1.5.5

Gaussian processes (GP), and explain their benefits when used in BO. In Sec. 6.3, we formalize the proposed novel acquisition function to solve unconstrained problems. In Sec. 6.4, such acquisition is extended for the constrained case in the presence of a budget of failures. In Sec. 6.5, we validate both acquisition functions empirically on common benchmarks for global optimization and real-world applications. We conclude with a discussion in Sec. 6.6.

6.1 Introduction

When deploying machine learning (ML) algorithms in real-world scenarios, a key difficulty lies in the proper management of undesired outcomes. These are usually inevitable when learning under unknown or uncertain circumstances. As an extreme case, in applications like autonomous driving, a failure in the decision-making may lead to human casualties. Such safety-critical scenarios need conservative ML algorithms, which forbid any failures. On the other hand, there exist scenarios in which failures are still undesired, although might not come at a high cost. For example, when learning the controller parameters of an industrial drilling machine to drill faster, a few configurations might break the drill bits, but in exchange, a faster drilling can be learned. In such non-safety-critical applications, failures shall be considered as a valuable source of knowledge, and one would tolerate a limited number of them in exchange for better learning performance.

In this chapter, we propose two complementary, yet distinct contributions. Our first contribution is a failures-aware strategy for BOC that, in contrast to prior work, does not need to be initialized in a safe region and that makes decisions taking into account the budgets of remaining failures and evaluations.

Our second contribution is a novel acquisition function inspired by key notions of the geometry of excursion sets in stochastic processes. In (Adler and Taylor, 2009), an excursion set is defined over smooth manifolds as those points for which a process realization crosses upwards a given threshold. The larger the threshold, the more likely it is that an *upcrossing* will reveal the location of the global maximum. Based on this intuition, we derive an acquisition function, which can be written analytically, is cheap to evaluate, and explicitly includes the process derivative to make optimal decisions.

6.2 Excursion sets in Bayesian optimization

The main goal is to address the unconstrained optimization problem (2.2), where the objective $f : \mathcal{X} \to \mathbb{R}$ is expensive to evaluate, and $\mathcal{X} \subset \mathbb{R}^D$. To this end,

we propose a search strategy inspired by the study of the differential and integral geometry of excursion sets in stochastic processes (Adler and Taylor, 2009). In the particular case of GPs, analytical expressions can be derived for such sets. In the following, we provide the needed mathematical tools and intuition over which our search strategy is constructed.

6.2.1 Excursion sets in Gaussian processes

Let us assume a zero-mean scalar Gaussian process f, with $\mathcal{X} = [0,1]^D$, $D = 1$, and stationary covariance function $k(\tau) = k(\|x - \hat{x}\|_2)$. The excursion set $\{x \in \mathcal{X} : f(x) \geq u\}$ is defined as the set of locations where the process f is above the threshold u. In (Adler and Taylor, 2009, Part II. Geometry), such sets are characterized by the number of upcrossings of process samples through the level u, i.e., $N_u^+ = \#\{x \in \mathcal{X} : f(x) = u, f'(x) > 0\}$, where $f'(x)$ is the derivative of the process. Intuitively, large N_u^+ represents a high frequency of upcrossings, which is connected with having many areas in $\mathcal{X}$ where $f(x)$ lives above u. For a one-dimensional, stationary, almost surely continuous and mean-square differentiable Gaussian process, the expected number of upcrossings (Rasmussen and Williams, 2006, Sec. 4.1) is given by the well-known Rice's formula (Lindgren, 2006, Sec. 3.1.2)

$$\begin{aligned}\mathbb{E}\left[N_u^+\right] &= \int_0^1 \mathbb{E}_{p(f,f'|x)}\left[f' : f = u, f' > 0\right] \mathrm{d}x \\ &= \int_0^1 \int_{-\infty}^{+\infty} \int_0^{+\infty} f' \delta(f-u) p(f, f'|x) \mathrm{d}f' \mathrm{d}f \mathrm{d}x \\ &= \frac{1}{2\pi} \sqrt{\frac{-k''(0)}{k(0)}} \exp\left(-\frac{u^2}{2k(0)}\right),\end{aligned} \tag{6.1}$$

where $p(f, f'|x)$ is the joint density of the process and its derivative, both queried at location x, δ is the Dirac delta, and the second derivative of the covariance function k'' must exist. Interestingly, (6.1) can be used to approximate the probability of finding the supremum of a process realization above a high level u. The growth rate of the approximation error with respect to u is bounded

$$\left|\mathbb{E}\left[N_u^+\right] - \Pr(\sup_{x\in[0,1]} f(x) \geq u)\right| < O(e^{-\beta u^2/k(0)}), \tag{6.2}$$

as $u \to \infty$, with $O(\cdot)$ indicating the limiting behavior of the approximation error and $\beta > 1$ needs to be found (Adler and Taylor, 2009, Sec. 14). The intuitive reasoning behind this is simple: If f crosses a high level u, it is unlikely to do so more than once. Therefore, the probability that f meets its supremum above u is close to the probability that there is an upcrossing of u. Since the number of upcrossings of a

high level will always be small, the probability of an upcrossing is well approximated by $\mathbb{E}\left[N_u^+\right]$.

While the bound in (6.2) does not hold for the general case $D > 1$, we use it as a starting point to build a new acquisition function for $D \geq 1$ (cf. Sec. 6.2.3), which shows empirically superior results than state-of-the-art BO methods. In the following section, we show, for $D = 1$, how $\mathbb{E}\left[N_u^+\right]$ can be leveraged to lead the search towards areas where the number of upcrossings is large, or equivalently, where the global maximum is more likely to be found. Thereafter, we extend the result for $D \geq 1$.

6.2.2 Practical interpretation for use in Bayesian optimization

The expected number of upcrossings (6.1) contains valuable information about the amount of times a sample realization of the process z "upcrosses" the level u. However, (6.1) cannot be used directly for decision-making because it is a global property of the process itself, rather than a local quantity at a specific location x. Next, we provide a practical interpretation that relaxes some of the assumptions made to obtain (6.1) and allows for its use in BO. To this end, we introduce three modifications.

First, when seeking for the optimum of the process, it is more useful to consider both, the up- and down-crossings through the level u, as both of them occur near the optimum when u is large. This quantity is defined in (Lindgren, 2006, Sec. 3.1.2) as the expected number of *crossings*

$$\begin{aligned}\mathbb{E}\left[N_u\right] &= \int_0^1 \mathbb{E}_{p(f,f'|x)}\left[|f'| : f = u\right] \mathrm{d}x \\ &= \int_0^1 \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} |f'|\delta(f-u)p(f,f'|x)\mathrm{d}f'\mathrm{d}f\mathrm{d}x,\end{aligned} \tag{6.3}$$

with $N_u = \#\{x \in [0,1] : f(x) = u\}$.

Second, BO uses pointwise information to decide on how interesting it is to explore a specific location x. (Lindgren, 2006, Theorem 3.1) proposes the *intensity of expected crossings* $\mathbb{E}\left[N_u(x)\right]$, which can be computed by simply removing the domain integral in (6.3)

$$\mathbb{E}\left[N_u(x)\right] = \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} |f'|\delta(f-u)p(f,f'|x)\mathrm{d}f'\mathrm{d}f. \tag{6.4}$$

Third, when conditioning the Gaussian process f on observed data $\mathcal{D}_t^f$, it becomes non-stationary[2], and thus, the predictive distribution of a query $f(x)$ changes

[2]Note that all GPs are non-stationary when conditioned on data, even if the covariance function that defines them is stationary.

as a function of x. The dependency on $\mathcal{D}_t^f$ is introduced in (6.4) following (Lindgren, 2006, Remark 3.2), as

$$\mathbb{E}\left[N_u(x|\mathcal{D}_t^f)\right] = \int_{-\infty}^{+\infty} |f'|p(u, f'|x, \mathcal{D}_t^f)\mathrm{d}f', \tag{6.5}$$

where the joint density is evaluated at $f = u$ after resolving the integral over the Dirac delta. We next provide a brief analysis for solving (6.5).

Using the rule of conditional probability, we have $p(u, f'|x, \mathcal{D}_t^f) = p(u|x, \mathcal{D}_t^f)p(f'|u, x, \mathcal{D}_t^f)$. The first term, $p(u|x, \mathcal{D}_t^f) = \mathcal{N}(u; \mu(x), \sigma^2(x))$, is a Gaussian density[3] evaluated at u, with the predictive mean and variance of the GP model. The second term is also a Gaussian density over the process derivative, conditioned on $f = u$. This can be seen as adding a *virtual* observation u at location x to existing data set. Hence, $p(f'|x, u, \mathcal{D}_t^f) = p(f'|\mathcal{D}_t^f \cup \{x, u\}) = \mathcal{N}(f'; \mu'(x), \nu^2(x))$. Then, (6.5) can be rewritten as

$$\begin{aligned}\mathbb{E}\left[N_u(x|\mathcal{D}_t^f)\right] &= p(u|x, \mathcal{D}_t^f)\int_{-\infty}^{+\infty} |f'|p(f'|\mathcal{D}_t^f \cup \{x, u\})\mathrm{d}f' \\ &= \mathcal{N}(u; \mu(x), \sigma^2(x))\left(2\nu(x)\phi\left(\gamma(x)\right) + \mu'(x)\mathrm{erf}\left(\tfrac{\gamma(x)}{\sqrt{2}}\right)\right),\end{aligned} \tag{6.6}$$

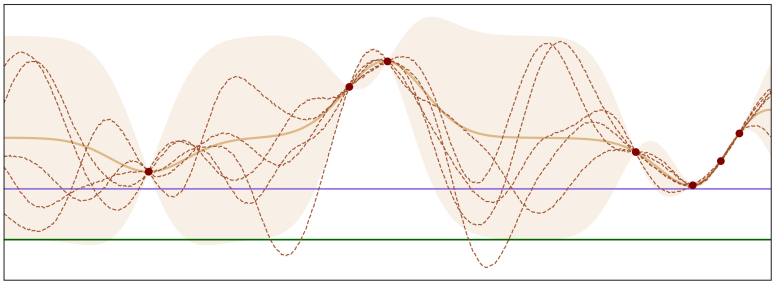
where $\gamma(x) = \mu'(x)/\nu(x)$, ϕ is the probability density function of a standard normal distribution, and $\mathrm{erf}\,(\cdot)$ is the error function (see App. B.1 for a complete derivation). Fig. 6.1 shows $\mathbb{E}[N_u(x|\mathcal{D}_t^f)]$ for two different values of u, where the GP is conditioned on seven observations. As can be seen, different thresholds imply different intensity of crossings for the same process. When the threshold is near collected evaluations, the largest intensity of crossings tends to be concentrated near the data. On the contrary, when it is far from the data, the largest intensity of crossings is found in areas of large variance.

6.2.3 Extension to D dimensions

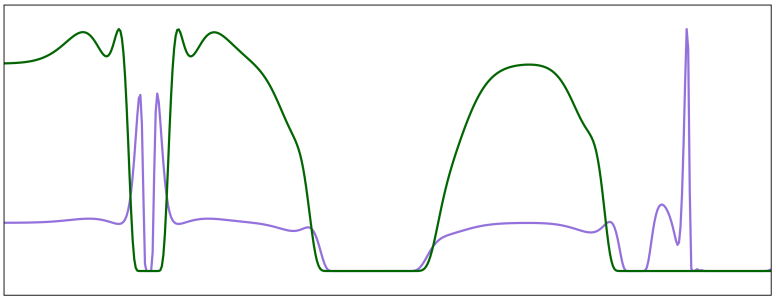
Although (6.6) was derived for $D = 1$, we can extend it to the case $D \geq 1$. Since (6.5) depends on $|f'|$, a natural extension is to consider the L-1 norm of the gradient of the process $\|\nabla f(x)\|_1 = \sum_{j=1}^{D} |\frac{\partial f(x)}{\partial x_j}|$. Following this, we extend (6.6) as $\mathbb{E}_{p(f(x),\nabla f(x))}[\|\nabla f(x)\|_1 : f(x) = u, \mathcal{D}_t^f]$,

$$\begin{aligned}\mathbb{E}\left[N_u(x|\mathcal{D}_t^f)\right] \simeq \mathcal{N}(u; \mu(x), \sigma^2(x))\times \\ \sum_{j=1}^{D}\left(2\nu_j(x)\phi(\gamma_j(x)) + \mu_j(x)\mathrm{erf}\left(\tfrac{\gamma_j(x)}{\sqrt{2}}\right)\right),\end{aligned} \tag{6.7}$$

[3]Using simplified notation, we write $p(u|x, \mathcal{D}_t^f)$ to refer to the density function $p_{f|x,\mathcal{D}_t^f}(\xi)$ evaluated at $\xi = u$. Similarly, we write $p(u, f'|x, \mathcal{D}_t^f)$ to refer to the joint density function $p_{f,f'|x,\mathcal{D}_t^f}(\xi, \zeta)$ evaluated at $\xi = u$ for some value ζ.



(a) Gaussian process posterior



(b) Intensity of expected crossings $\mathbb{E}[N_u(x|\mathcal{D}_t^f)]$

Figure 6.1: (a) Gaussian process posterior conditioned on a set of observations. Given a process realization (dashed lines), two choices for the threshold u (solid lines) determine two different excursion sets. (b) Intensity of expected crossings $\mathbb{E}[N_u(x|\mathcal{D}_t^f)]$ for each threshold u. Higher values correspond to areas where the boundaries of the excursion sets are likely to be, i.e., where the process is more likely to cross u. The curves are normalized to have the same maximum value.

where $\gamma_j(x) = \mu_j(x)/\nu_j(x)$. The gradient $\nabla f(x) \sim \mathcal{N}(\nabla f(x); \nabla\mu(x), V(x))$ follows a multivariate Gaussian, and $\mu_j(x) = [\nabla\mu(x)]_j$ and $\nu_j(x) = ([V(x)]_{jj})^{1/2} = \sqrt{\partial^2 k(x_j, x_j)/\partial x_j^2}$. Note that $\nabla\mu(x)$ and $V(x)$ depend on the extended data set $\mathcal{D}_t^f \cup \{x, u\}$.

In the following sections, we propose two novel algorithms that build upon the quantity (6.7).

6.3 Excursion search algorithm

The modifications applied to (6.1), detailed above, allow extracting useful information about how likely is the process f to cross a certain level u at each location x. When u is a lower bound on the collected data, (6.7) reveals locations where the process is more likely to have a minimum. If we repeatedly evaluate at such locations, one would expect to approach faster the global minimum. In the following,

we characterize (6.7) as an acquisition function for optimal decision-making.

6.3.1 Threshold of crossings as the global minimum

The choice of the threshold u in (6.7) is important when trying to find the global minimum. A hypothetically appropriate low value for u is right above the global minimum $f_* = f(x_*)$, i.e., $u = f_* + \epsilon$, where $\epsilon > 0$ is small. Then, if crossings through $u = f_* + \epsilon$ are likely to occur at a specific area, we know that such area is likely to contain the global minimum, and thus, will show a large $\mathbb{E}[N_u(x|\mathcal{D}_t^f)]$. However, in practice we do not have access to the true f_* of the objective function, and thus, cannot compute u in the aforementioned way. At most, we are able to assume a distribution over the global minimum $f_* \sim p(f_*)$, implied by the GP model on f. In the following, we assume that u follows such distribution, i.e., $u \sim p(u) = p(f_*)$.

It is well-known in extreme value theory (De Haan and Ferreira, 2007) that f_* follows one of the three extreme value distributions: Gumbel, Fréchet, or Weibull, which generally model tails distributions. For example, in (Wang and Jegelka, 2017), the Gumbel distribution is chosen to model $p(f_*)$. However, such distribution has infinite support, while in practice it is not useful to have any probability mass above the best observed evaluation $\eta = \min(y(x_1), \ldots, y(x_t))$. Instead, we consider the Fréchet distribution as a more appropriate choice as it provides finite support $f_* \leq \eta$. For minimization problems, we can define it in terms of its survival function $\mathcal{F}_{s,q}(a) = \Pr(f_* \geq a)$, given by

$$\mathcal{F}_{s,q}(a) = \begin{cases} 0, & \text{if } a > \eta \\ \exp\left(-\left(\frac{\eta - a}{s}\right)^{-q}\right), & \text{if } a \leq \eta \end{cases} \tag{6.8}$$

where $\Pr(f_* \geq a) = \int_a^{+\infty} p(f_*) \mathrm{d}f_*$, and the parameters $s > 0$ and $q > 1$ can be estimated from data following the same approach as in (Wang and Jegelka, 2017, Appendix B). A thorough analysis on the advantage of using the Fréchet distribution, instead of the Gumbel distribution, for gathering samples of f_* can be found in App. B.2. Using the above definition, the stochastic threshold $u \sim p(u) = p(f_*)$, makes the quantity (6.7) also stochastic. We propose to compute its expectation over u, i.e., $\mathbb{E}_{p(u)}[\mathbb{E}[N_u(x|\mathcal{D}_t^f)]] = \mathbb{E}_{p(f_*)}[\mathbb{E}[N_{f_*}(x|\mathcal{D}_t^f)]]$, which we explain next.

6.3.2 Acquisition function

We define the *excursion search* (XS) acquisition function as

$$\alpha_{\mathrm{X}}(x) = \mathbb{E}_{p(f_*)}\left[\mathbb{E}\left[N_{f_*}(x|\mathcal{D}_t^f)\right]\right] \simeq \frac{1}{S}\sum_{l=1}^{S} \mathbb{E}\left[N_{f_*^l}(x|\mathcal{D}_t^f)\right], \tag{6.9}$$

where the outer expectation is intractable and is approximated via sampling. For each sample $f_*^l \sim p(f_*)$, (6.7) needs to be recomputed. The samples can be collected through the inverse of (6.8), $f_*^l = \mathcal{F}_{s,q}^{-1}(\xi^l)$. $\xi^l \sim U(0,1)$ follows a uniform distribution in the unit interval, and $\mathcal{F}_{s,q}^{-1}(\xi^l) = \eta - s(-\log(1-\xi^l))^{-1/q}$.

Intuitively, the XS acquisition function (6.9) reveals areas near the global minimum (i.e., where the gradient crosses the estimated f_* with large norm), instead of directly aiming at potential maximums, minimums, or saddle points. Furthermore, XS inherently trades off exploration with exploitation: At early stages of the search, the estimated Fréchet distribution (6.8) reflects large uncertainty about f_*, which causes the samples f_*^l to lie far from the data. Hence, exploration is encouraged, as shown in Fig. 6.1 (green lines). At later stages, when more data is available, the Fréchet distribution (6.8) shrinks toward the lowest observations, which then encourages exploitation, as shown in Fig. 6.1 (violet lines).

The acquisition (6.9) is our first contribution, and can be used for unconstrained optimization problems, e.g., (2.2).

6.4 Bayesian optimization with a limited budget of failures

In the previous section, we introduced a new acquisition function (6.9) grounded in the connection between the true optimum of the process f and the expected number of crossings through its current estimate (cf. (6.2)). However, such acquisition does not explicitly have into account any budget of failures B or evaluations T. In the following, we propose an algorithm that makes use of B and T to balance the decision making between (i) safely exploring encountered safe areas, and (ii) searching outside the safe areas at the risk of failing, when safe areas contain no further information.

6.4.1 Problem formulation

To the unconstrained problem (2.2), we add G black-box constraints, $g_j : \mathcal{X} \to \mathbb{R}$, $j = \{1, \dots, G\}$, also corrupted by noise and expensive to evaluate. Moreover, we assume a non-safety critical scenario, where violating the constraints is allowed, but it is strictly forbidden to do so more than B times. Analogously, we allow only for a maximum number of $T \geq B$ evaluations. The case $T < B$ is not considered herein, as the budget of failures can simply be ignored. Under these conditions, we formulate the constrained optimization problem with limited budget of failures as

$$x_*^{\mathrm{c}} = \underset{x \in \mathcal{X}}{\operatorname{argmin}} \; f(x), \; \text{s.t.} \; g_1(x) \leq 0, \dots, g_G(x) \leq 0$$

$$\text{under failures} \sum_{t=1}^{T} \Gamma(x_t) \leq B, \tag{6.10}$$

where x_*^{c} is the location of the constrained minimum, and $\Gamma(x_t) = \mathbb{I}\left[g_1(x_t) > 0 \vee \ldots \vee g_G(x_t) > 0\right]$ equals 1 if at least one of the constraints is violated at location x_t, and 0 otherwise. $\mathbb{I}$ is the indicator function, and $g(x_1), \ldots, g(x_T)$ are the collected evaluations of the constraints at locations $x_1, \ldots, x_T$. Since the constraints g_j are unknown, and modeled as independent Gaussian processes $g_j \sim \mathcal{GP}\left(0, k(x, \hat{x})\right)$, queries $f(x)$ and $g(x)$ are stochastic and (6.10) cannot be solved directly. Instead, we address the analogous probabilistic formulation from Gelbart et al. (2014):

$$\begin{aligned} x_*^{\mathrm{c}} \simeq \underset{x \in \mathcal{X}}{\operatorname{argmin}} \ \mu(x), \text{s.t.} \ \prod_{j=1}^{G} \Pr(g_j(x) \leq 0) \geq \rho \\ \text{under failures} \sum_{t=1}^{T} \Gamma(x_t) \leq B, \end{aligned} \tag{6.11}$$

where $\Pr(g_j(x) \leq 0) = \Phi\left(-\mu_j(x)/\sigma_j(x)\right)$, Φ is the cumulative density function of a standard normal distribution, and $\rho \in (0, 1)$ is typically set close to one. The predictive mean μ_j and variance σ_j^2 conditioned on $\mathcal{D}_t^{g_j}$ of each g_j are computed as in Sec. 2.2. In the following, we provide a novel Bayesian optimization strategy to address (6.11).

6.4.2 Safe exploration with dynamic control

In order to include the probability of constraint satisfaction in the decision making, we propose a similar approach to Gelbart et al. (2014) by explicitly adding a probabilistic constraint to the search of the next evaluation

$$\begin{aligned} x_{\text{next}} = \underset{x \in \mathcal{X}}{\operatorname{argmax}} \ \alpha_{\mathrm{X}}(x) \\ \text{s.t.} \ \prod_{i=1}^{K} \Pr(g_j(x) \leq 0) \geq \rho_t, \end{aligned} \tag{6.12}$$

where the parameter $\rho_t \in (0, 1)$ determines how much we are willing to tolerate constraint violation at each iteration t. This leads the search away from areas where the constraint is likely to be violated, as those areas get revealed during the search.

Contrary to (Gelbart et al., 2014), where ρ_t is fixed a priori, we propose to choose it at each iteration, depending on the remaining budget of failures $\Delta B_t = B - \sum_{j=1}^{t} \Gamma(x_j)$ and remaining iterations $\Delta T_t = T - t$. Intuitively, the more failures we have left (large ΔB_t), the more we are willing to tolerate constraint violation (large ρ_t). We achieve this by proposing an automatic control law to drive ρ_t, which we describe next.

Let us define a latent variable $z_t = \Phi^{-1}(\rho_t)$, $z_t \in \mathbb{R}$ that follows a deterministic process $z_{t+1} = z_t + u_t$, using a dynamic feedback controller $u_t = u_t(\Delta B_t, \Delta T_t)$. Such controller drives the process toward one of the two references: $z_{\text{safe}} = \Phi^{-1}(\rho_{\text{safe}})$ and $z_{\text{risk}} = \Phi^{-1}(\rho_{\text{risk}})$, where typical values are $\rho_{\text{safe}} = 0.99$ and $\rho_{\text{risk}} = 0.01$. We define a control law

$$u_t = (z_{\text{safe}} - z_t)\frac{\Gamma(x_t)}{\Delta B_t} + (z_{\text{risk}} - z_t)\frac{\Delta B_t}{2\Delta T_t}, \tag{6.13}$$

with $\Delta B_t > 0$, $\Delta T_t > 0$, and $\Delta B_t \leq \Delta T_t$. The first term drives the process toward z_{safe} when a failure occurs at iteration t, with intensity $1/\Delta B_t$. In this way, the fewer failures are left in the budget, the more urgently the process chases z_{safe}. The second term attempts to push z_t down to z_{risk} with an intensity proportional to the ratio between the remaining failures and iterations.

When $\Delta B_t = 0$, but $\Delta T_t > 0$, only a conservative safe exploration is allowed. To do so, we set $u_t = (z_{\text{safe}} - z_t)$ for the remaining iterations until $t = T$. Additionally, if there are more failures left than remaining iterations, i.e., $\Delta B_t > \Delta T_t$, the remaining budget of failures is not decisive for decision making, and thus, we set $u_t = (z_{\text{risk}} - z_t)$.

The resulting control strategy weights risky versus conservative decision-making by considering the budget of evaluations and iterations left: When no failures occur for a few consecutive iterations, ρ_t is slowly driven toward ρ_{risk}, and when a failure takes place, it lifts up ρ_t toward ρ_{safe}.

6.4.3 Risky search of new safe areas

The probabilistic constraint in (6.12) puts a hard constraint on the decision making by not allowing evaluations in regions that are known to be unsafe. When ρ_t is high, (6.12) will discard regions where no data has been collected and locally explore regions where safe evaluations are present. Such conservative decision making is desirable when $\Delta B_t \ll \Delta T_t$ because it avoids unsafe evaluations. The smaller the ρ_t, the more risky evaluations we can afford, which makes the constraint information less important in the decision making. However, when ρ_t is too low, the probabilistic constraint tends to be ignored, and the decisions are based on the information from the objective. Albeit this indeed counts as the wanted risky exploration strategy, completely ignoring the constraint information could result in repeated evaluations in unsafe areas. To avoid this, we follow the approach from (Gelbart et al., 2014), where the acquisition function is aware of the constraint information, without this being a hard constraint. Therein, locations are chosen at

$$x_{\text{next}} = \underset{x \in \mathcal{X}}{\operatorname{argmax}} \ \alpha_{\text{X}}(x) \prod_{j=1}^{D} \Pr(g_j(x) \leq 0). \tag{6.14}$$

This approach "jumps" outside the current safe areas at the risk of failing, while the multiplying term discourages exploration in areas revealed to be unsafe.

Trading off risky versus safe exploration depends on the remaining budget ΔB_t, and is quantified by ρ_t, as detailed in Sec. 6.4.2. We propose a user-defined decision boundary ρ_b, such that if $\rho_t \leq \rho_\mathrm{b}$, the next location will be selected using (6.14), and (6.12) otherwise.

While (6.12) assumes that a safe area has already been found, this might not be the case at an early stage of the search. In such case, we collect observations using (6.14) and only resort to the risk versus safety trade-off once a safe area has been found.

Pseudocode for the overall framework, named *failures-aware excursion search* (XsF), and an analysis of its computational complexity can be found in App. B.3. XsF returns the estimated location of the constrained minimum x^c_* from (6.11), computed by setting $\rho = \rho_\mathrm{safe}$.

6.5 Empirical analysis and validation

We empirically validate Xs and XsF by comparing their performance against state-of-the-art methods. We consider three different scenarios. In the first one, we validate each method on common challenging benchmarks for global optimization. In the second and third scenarios we compare XsF against state-of-the-art methods in constrained optimization problems. In the second, we optimize the hyperparameters of a neural network to achieve maximum compression without degrading its performance. In the third, we learn the state feedback controller of a cart-pole system. Both, Xs and XsF are implemented in Python. The code, which includes scripts to reproduce the results presented herein, is documented and publicly available at https://github.com/alonrot/excursionsearch.

6.5.1 Experimental setup

To assess the performance of all methods we use *simple regret* $r_T = f(x_\mathrm{bo}) - \min_{x\in\mathcal{X}} f(x)$, where $x_\mathrm{bo} = \arg\min_{t\in[1,T]} y(x_t)$ is the point that yielded the best observation so far. In the constrained case, such point is given by $x_\mathrm{bo} = \min_{t\in[1,T]} y(x_t)$ s.t. $y^g(x_t) \leq 0$. We quantify how often safe evaluations are collected using $\Omega = 100 N_\mathrm{safe}/T$, where N_safe is the number of safe evaluations made at the end of each run.

In all cases, the domain is scaled to the unit hypercube. We set $\rho_\mathrm{safe} = 0.99$, $\rho_\mathrm{risk} = 0.01$, and $\rho_0 = 0.1$. The decision boundary was set at $\rho_\mathrm{b} = 0.5$. Both,

the objective function and the constraint are modeled with a zero-mean GP, with a squared exponential kernel. The lengthscales and the signal variance are fit to the data after each iteration. Further implementation details, such as hyperprior choices and number of random restarts, are reported in App. B.4.

6.5.2 Benchmarks for global optimization

We validate Xs and XsF in two challenging benchmarks for global optimization: Hartman 6D, and Michalewicz 10D (Jamil and Yang, 2013). We allow a budget of evaluations $T = 100$ in all cases and repeat all experiments 50 times for each function using a different seed. As in (Hernández-Lobato et al., 2016; Wang and Jegelka, 2017), we use the same initial evaluation (previously selected at random) across all repetitions.

Excursion search (Xs)

We assess the performance of Xs by comparing against popular BO methods: Expected improvement (EI) (Močkus, 1975), Probability of improvement (PI) (Kushner, 1964), Min-Value Entropy Search (MES) (Wang and Jegelka, 2017), and Gaussian process upper confidence bound (UCB) (Srinivas et al., 2010). Our implementations are based on those used by (Wang and Jegelka, 2017), available online[4]

Fig. 6.2a shows the evolution of the simple regret over iterations in the Michalewicz 10D benchmark. Xs reaches the lowest regret, and none of the methods is able to achieve a regret close to zero, which is not surprising given high dimensionality of the problem and the number of allowed evaluations. Table 6.1 (top) shows statistics on the regret value for both benchmarks. While all methods report a generally high regret in Michalewicz 10D, Xs clearly outperforms all the other methods in Hartman 6D, as it finds a near-zero regret.

Failures-aware excursion search (XsF)

To validate XsF, we propose a constrained optimization problem under a limited budget of failures. For this, we simply impose a constraint to the aforementioned benchmarks $g(x) = \prod_{i=1}^{D} \sin(x_i) - 2^{-D}$. Such function uniformly divides the volume in 2^D sub-hypercubes, and places 2^{D-1} convex disjoint unsafe areas in each one of the sub-hypercubes, so that they are never adjacent to each other. We allow $T = 100$ and a considerably small budget of failures $B = 10$ to all methods. We compare XsF against expected improvement with constraints (EIC) (Gelbart et al., 2014) and predictive entropy search with constraints (PESC) (Hernández-Lobato et al., 2016).

[4] github.com/zi-w/Max-value-Entropy-Search

EIC and PESC are terminated when their budget is depleted. Although individual experiments rarely finish at the same iteration (i.e., some may deplete the budget of failures earlier than others), we use in our results the last regret reported by each algorithm. For EIC, we use our own implementation, while for PESC we use the available open source implementation, included in Spearmint[5].

In Fig. 6.2b, we see that XsF reaches a higher number of total evaluations and consistently achieves lower regret than EIC and PESC. Fig. 6.2b (middle) shows the evolution of the remaining budget of failures ΔB_t over iterations (mean and standard deviation). As can be seen, EIC and PESC deplete the budget faster than XsF. Finally, Fig. 6.2b (bottom) shows the evolution of the ρ_t parameter used to switch between risky and safe strategies in XsF, and also as a threshold for probabilistic constraint satisfaction (cf. Sec. 6.4.2). We differentiate two stages: During the initial iterations ρ_t is low, and thus, risky exploration is preferred, which allows XsF to quickly discover better safe areas. At the last iterations, when the budget is depleted, XsF keeps exploring conservatively the discovered safe areas, with $\rho_t = \rho_{\text{safe}}$.

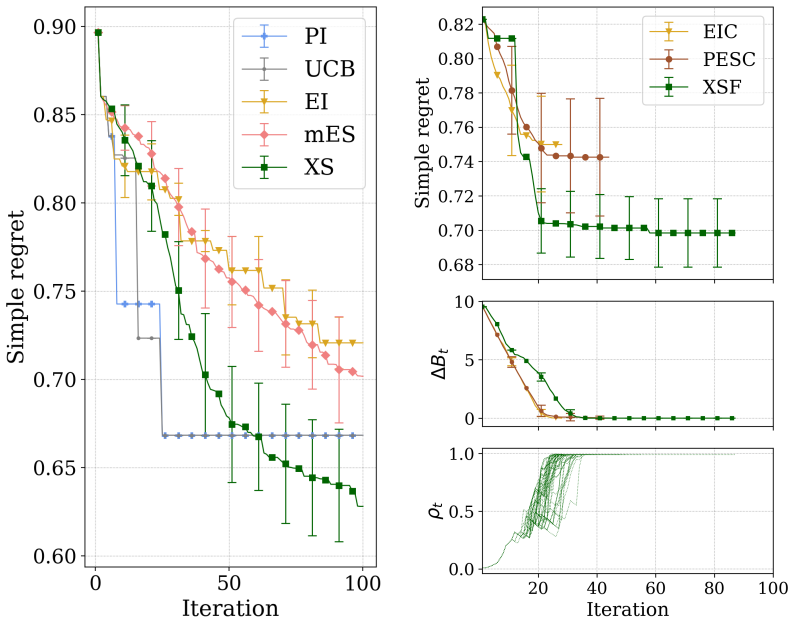
Table 6.1 (bottom) shows the regret for both, the Michalewicz 10D and the Hartman 6D functions in the constrained case. While the regret comparison is similar to the 10D case, the 6D case shows that Xs clearly outperforms the other methods. The quantity Ω confirms that XsF visits safe evaluations more often than the other methods.

Generally, hyperparameter learning influences the performance of the algorithms.

[5] github.com/HIPS/Spearmint/tree/PESC

Table 6.1: Constrained (top) and unconstrained benchmarks (bottom). Simple regret r_T (mean $\pm$ std) and percentage of safe evaluations Ω.

	Hartman 6D		Michalewicz 10D	
	r_T		r_T	
EI	0.75 ± 0.00		0.67 ± 0.00	
MES	0.47 ± 0.00		0.67 ± 0.00	
PI	0.34 ± 0.11		0.72 ± 0.03	
UCB	0.39 ± 0.18		0.70 ± 0.06	
Xs	$\mathbf{0.02 \pm 0.01}$		$\mathbf{0.63 \pm 0.06}$	
	r_T	Ω (%)	r_T	Ω (%)
EIC	0.33 ± 0.35	68 ± 30	0.75 ± 0.06	15 ± 3
PESC	0.14 ± 0.22	61 ± 29	0.74 ± 0.07	16 ± 5
XsF	$\mathbf{0.09 \pm 0.14}$	$\mathbf{90 \pm 16}$	$\mathbf{0.70 \pm 0.04}$	$\mathbf{28 \pm 7}$



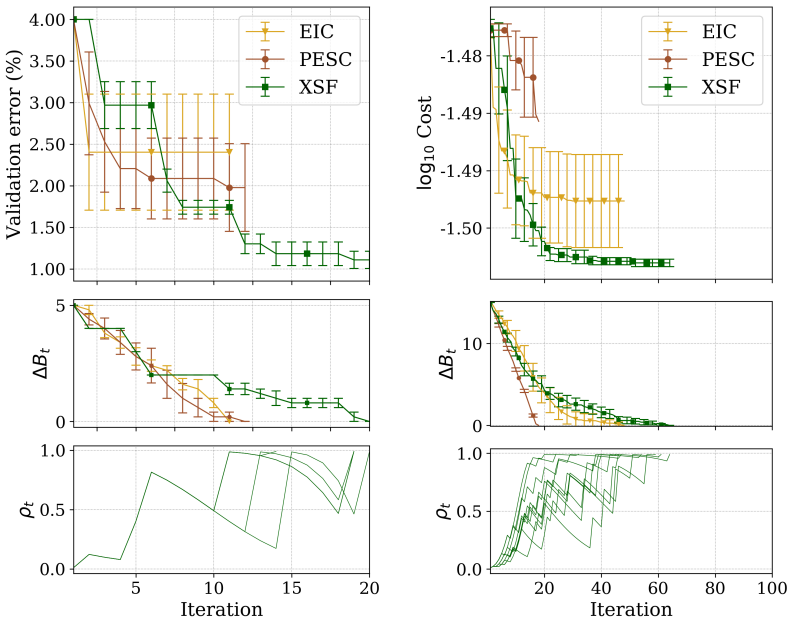
(a) Unconstrained (b) Constrained

Figure 6.2: Performance assessment of XS and XSF on the Michalewicz 10-dimensional benchmark.

In App. B.5, we show experiments with fixed hyperparameters and a correct GP model, where XS and XSF outperform the aforementioned methods.

6.5.3 Compressing a deep neural network

Applying modern deep neural networks (NNs) to large amounts of data typically results in large memory requirements to store the learned weights. Therefore, finding ways of reducing model size without degrading the NN performance has become an important goal in deep learning, for example, to meet storage requirements or to reduce energy consumption. Bayesian compression has been recently proposed as a mean to reduce the NN size: Given an NN architecture, an approximate posterior distribution q on the NN weights is obtained by maximizing the evidence lower bound (ELBO), which balances the expected log-likelihood of samples from q and the theoretical compression size, as given by the KL divergence between q and a prior distribution p (Havasi et al., 2018). A penalization factor β can be used to scale the KL divergence to control the final size of the NN. Finding the value of β that achieves the lowest compression size without significantly degrading NN performance is a challenging and expensive tuning problem. To alleviate the effort of tuning hyperparameters, Bayesian optimization is commonly used. Herein, we propose to minimize the validation error of the NN while keeping its size below



(a) NN compression (b) Cart-pole problem

Figure 6.3: Performance comparison of XsF against EIC and PESC

a threshold, using constrained Bayesian optimization under a limited budget of failures. While in this example failing to comply with the size requirements is not catastrophic, collecting many failures is undesirable.

We use a LeNet-5 on the MNIST dataset, and a required size below 15 kB. The parameters to tune are β, the learning rate χ, and a scaling factor κ on the the number of neurons of all layers. As a reference for our implementation, we used the open source implementation of MIRACLE[6] Havasi et al. (2018). We allow $T = 20$ and $B = 5$ and repeat the experiments 5 times. We fix the training epochs to 20000 for each evaluation (about 25 min in wall-clock time). As shown in Fig. 6.3a, XsF achieves the lowest regret and standard deviation. The best safe observation is reported by XsF, with validation error 0.76% and theoretical NN size of 12.4 kB (x553 compression). The learned parameters are $\beta = 6.56 \times 10^{-7}$, $\chi = 1.35 \times 10^{-3}$ and $\kappa = 10$.

6.5.4 Tuning a feedback controller

Bayesian optimization has been used for learning robot controllers to alleviate manual tuning (Calandra et al., 2016; Rai et al., 2018). Herein, we propose to tune a 4D state feedback controller on a cart-pole system, where unstable controllers found during the search are undesirable, as human intervention is required to reset the

[6] github.com/cambridge-mlg/miracle

platform, but not catastrophic. In this setting, allowing a limited budget of failures might increase chances of finding a better optimum. In practice, a constraint can be placed in the cart position to trigger an emergency stop when it grows large (cf. Chap. 3). Controllers that surpass such limit at any moment during the experiment are considered a failure. We use the simulated cart-pole system[7] from openAI gym Brockman et al. (2016), implemented in the MuJoCo physics engine Todorov et al. (2012). The tasks consists on, first stabilizing the pendulum starting from random initial conditions, and second, disturbing the cart position with a small step. We consider a budget $B = 15$ and $T = 100$, and repeat all experiments 10 times. Fig. 6.3b shows that XsF finds a better controller than the other methods.

6.6 Conclusions

In this chapter, we have presented two novel algorithms for Bayesian optimization (BO): Excursion search (Xs), which is based on the study of excursion sets in Gaussian processes, and failures-aware excursion search (XsF), which trades off risky and safe exploration as a function of the remaining budget of failures through a dynamic feedback controller. Our empirical validation shows that both algorithms outperform state-of-the-art methods. Specifically, in situations in which failing is permitted, but undesirable, XsF makes better use of a given budget of failures by depleting it at a slower rate, while generally achieving lower regret values.

In this section, we present a few ideas on how to further improve the proposed strategies and also discuss the prospective applicability of the proposed method to real-world settings.

6.6.1 Discussion

Despite of the effectiveness of Xs and XsF, empirically demonstrated, there are some aspects to both algorithms that can be further improved, yet they require a careful treatment. We discuss them next.

The main motivation for Xs lies in the error bound (6.2), which indicates that the larger the threshold u, the more likely it is that an "upcrossing" through u leads to the global supremum. In practice, collecting evaluations where the number of upcrossings is expected to be large is likely to result in quickly finding promising regions. This intuition, which is used to build up Xs, is valid for a one-dimensional input space and it is exploited through Rice's formula (6.1). However, its validity remains unclear in higher dimensions due to the following reasons.

[7] gym.openai.com/envs/InvertedPendulum-v2/

First, in (Adler and Taylor, 2009, Sec. 11.7), an extension of Rice's formula (6.1) to higher dimensions is provided. However, such extension relies on another quantity, the *mean Euler characteristic* of manifolds, which characterizes their integral and differential geometry, and is fundamentally different than the aforementioned upcrossings number. In contrast to our heuristic extension (cf. Sec. 6.2.3), such quantity is the *correct* extension of (6.1) to higher dimensions. Although theoretically more appealing, it also has a more complicated and less intuitive interpretation. Second, in (Adler and Taylor, 2009, Sec. 11.7), the error bound (6.2) is formulated in the context of the mean Euler characteristic. However, leveraging it for its use in BO, would imply a practical reinterpretation (cf. Sec. 6.2.2), possibly involving many approximations. Finally, another related quantity, not studied in this work, is the expected number of local maxima above the threshold u, which is described in (Adler and Taylor, 2009, Sec. 4.6).

Generally speaking, leveraging these quantities for practical use in BO is non-trivial. Although our final acquisition diverges from the theoretically correct result shown in (Adler and Taylor, 2009, Sec. 11.7), it remains motivated by the error bound (6.2), which is the core motivation to this result. Furthermore, our acquisition function (6.9) can be written analytically in the D-dimensional case and is cheap to evaluate. There are, nonetheless, two approximations involved: The expectation in Sec. 6.3.2 and the construction of the Fréchet distribution (cf. Sec. 6.3.1). Although this might be seen as a disadvantage, it is not uncommon in BO to use numerical approximations within the formulation of the acquisition function. For example, in (Wang and Jegelka, 2017), the same two types of approximations are used. In addition, other popular methods, such as (Hennig and Schuler, 2012; Hernández-Lobato et al., 2014) involve many more approximations than the ones used herein.

The computation of the Fréchet distribution (cf. Sec. 6.3.1) could also be improved by changing the type of discretization of the input domain. We follow the algorithm described in (Wang and Jegelka, 2017, Appendix B), where the input domain is discretized with a uniform grid. In contrast, (Hennig and Schuler, 2012) use a more sophisticated discretization which involves putting more resolution in areas where the minimum is likely to be. This would result in a more accurate description of the distribution of the global minimum, which is worth investigating.

Finally, studying how the error bound (6.2) can be leveraged to provide theoretical convergence guarantees would characterize the usefulness of the method beyond its effectiveness, which was empirically demonstrated herein. However, we emphasize once again that such convergence guarantees do not hold valid when deploying BO and BOC to automate controller tuning of real systems, like robots, as data collection is scarce.

6.6.2 Future lines

The proposed method for constrained optimization, XsF, is tailored to settings where paying the price of a few failures is affordable in hope of finding better optima. In the best case, these optima will lie outside the initial safe area, in case such an area is given.

In the foreseeable future, when deploying robotic systems to work and live among humans, a minimum level of adaptability will always be required. The robot will constantly be facing new environments and challenging interaction settings where adaptability will play a key role (see Sec. 4.5.2 for a related discussion). Methods like BO and BOC can help in this regard by automatically fine-tuning robot settings with a few trials, while avoiding (or mitigating) failures. However, even in safety critical applications, zero-failures learning can only be guaranteed probabilistically. In other words, failures are, and will always be, an inherent part of any learning process. When incorporated into the learning loop, failures are informative about what behavior should be avoided and, when properly leveraged, they can be a useful tool of information to speed up the learning process. Because of this, flexible algorithms that are neither too conservative nor too risky, like XsF, proposed herein, will potentially have a great benefit in future robotics applications. Furthermore, a particular feature that enhances flexibility when using XsF is the possibility of leaving to the user's choice an upper bound on the number of failures. This immediately contrasts with safe learning methods, like (Berkenkamp et al., 2016b; Sui et al., 2015), where no failures are allowed, and also with standard BO algorithms like (Gelbart et al., 2014; Hernández-Lobato et al., 2016), where the number of failures is unbounded.

Chapter 7

Conclusions

In this thesis, we have presented a selection of projects that we have worked on during my PhD. Herein, we first summarize the overall contributions of each project, then discuss potential research directions arising from them, and finally conclude with a reflection about the role of Bayesian optimization (BO) for robotics in the present and near future.

7.1 Summary

In this thesis, we have presented four contributions, whose motivation has been mainly driven by the following question: *Can we replace manual controller tuning with BO, and remain sample-efficient in robotics?*. In general, we have proposed several methodological contributions that address this question, and supported our research claims with experimental results in real robotic platforms. More specifically, in Chap. 3 we have demonstrated that manual tuning can be replaced with BO in real robotic platforms, which is the core problem of this thesis. Furthermore, we have addressed three theoretically founded algorithms that increase sample-efficiency when learning robot controllers using BO. Our results indicate that same or better performance can be achieved with fewer robot experiments, in general. A complete list of publications that have been developed during this PhD, associated or not with the object of this thesis, can be found in Sec. 7.1.1.

In the following, we explicitly state the contribution of each part of the thesis.

Automatic LQR tuning using Bayesian optimization

In Chap. 3, we have used Bayesian optimization (BO) to automate the tuning of a linear quadratic regulator (LQR). Therein, the performance objective is modeled

using a Gaussian process (GP) (Rasmussen and Williams, 2006). We have developed, and successfully demonstrated this framework using Entropy Search (ES) (Hennig and Schuler, 2012) on two robotic platforms: An inverted pendulum balanced with the humanoid robot Apollo (cf. Sec. 3.4) and a squatting task performed by the two-legged robot Hermes (cf. Sec. 3.5). Furthermore, this work is the first to apply ES in experiments for automatic controller tuning.

The results obtained with the inverted pendulum platform were successful in 2D and 4D tuning experiments, both when the method was initialized with an unstable and with a stable controller. While the 2D experiment could presumably also be handled by grid search or manual tuning, and thus mostly served as a proof of concept, the 4D tuning problem can already be considered difficult for a human.

The results obtained with the robot Hermes on a squatting task indicate that the proposed framework retains its effectiveness in higher-dimensional systems. Therein, we directly learn the low-level controller parameters using BO on a 6D problem. More specifically, our results show improvement upon a poor controller learned after only 20 experiments. In addition, combining the LQR tuning framework with an effective dimensionality reduction (Sec. 3.5.1) enables the deployability of BO, which in return, helps to mitigate manual tuning effort.

Below, we list a set of additional contributions associated to this work.

- A video demonstration that illustrates the automatic LQR tuning framework demonstrated on Apollo: https://youtu.be/TrGc4qp3pDM.
- A video summary showing the learning routine, the learned performance and robustness tests for the automatic LQR tuning on Hermes: https://youtu.be/udJAK60IWEc.
- A c++ implementation of ES, which is about ten times faster than the original Matlab implementation: https://github.com/alonrot/EntropySearchCpp.
- A c++ implementation of the robust communication interface used to communicate ES with Hermes: https://github.com/alonrot/userES_pubsub_lqr.
- An alternative Matlab implementation to the original ES code, which provides a more friendly user interface, plotting tools, and corrects bugs: https://github.com/alonrot/userES.

Trading off simulations and physical experiments using BO

In Chap. 4, we have proposed an extension of ES to the multi-fidelity setting in order to adaptively select between simulations and real experiments. The resulting algorithm automatically trades off information accuracy with evaluation effort. We demonstrate our framework by learning the controller parameters of a real cart-pole system, where simulations were 30 times faster (i.e., cheaper) than real experiments.

The experimental results confirm that using prior model information from a robot simulator can reduce the amount of data required to globally find good control policies.

We have also shown that the applicability of the automatic LQR tuning framework proposed in Sec. 3.3 easily extends to a setting with multiple information sources. Additionally, our extension of ES to the multi-fidelity setting (MF-ES) constitutes a novel contribution by itself.

In addition, a list with supplementary material is given below.

- A video demonstration of MF-ES on a cart-pole system can be found at https://youtu.be/oq9Qgq1Ipp8.
- Our Matlab implementation of MF-ES can be found at https://github.com/alonrot/mfES.

On the design of LQR kernels

In Chap. 5, we have discussed how to leverage the mathematical structure of the well-known LQR problem to speed up the learning process using BO. The corresponding performance objective is modeled using a GP, whose kernel is constructed by exploiting the linear model and the quadratic cost function that constitute the LQR problem. We have proposed two novel kernels: a parametric and a non-parametric version of the LQR kernel. Our numerical simulations demonstrate improved sample-efficiency and increased prediction accuracy over standard kernels, i.e., good controllers are learned from fewer experiments. A video recording of the talk at the conference, where this contribution was presented can be found at https://youtu.be/zsC6Lufkl_E.

Controller learning under limited budget of failures

In Chap. 6, we have presented two novel algorithms for BO: Excursion search (XS), which is based on the study of excursion sets in Gaussian processes, and failures-aware excursion search (XSF), which trades off risky and safe exploration as a function of the remaining budget of failures through a dynamic feedback controller. To our knowledge, XS is the first acquisition function that explicitly exploits the exact gradient posterior probability to make predictions. Our empirical validation shows that both algorithms outperform state-of-the-art methods. Specifically, in situations in which failing is permitted, but undesirable, XSF makes better use of a given budget of failures by depleting it at a slower rate, while generally achieving lower regret values. A Python implementation for both algorithms is available at

https://github.com/alonrot/excursionsearch.

7.1.1 List of publications

Herein, I list core papers in which I am first author (see Sec. 1.4), and upon which this thesis is mainly based. Additionally, I also list a series of papers in which I was a collaborator during my PhD.

- **Alonso Marco**, Philipp Hennig, Jeannette Bohg, Stefan Schaal, and Sebastian Trimpe, "Automatic LQR tuning based on Gaussian process global optimization", In IEEE International Conference on Robotics and Automation (ICRA), pages 270–277, © 2016 IEEE.
- **Alonso Marco**, Felix Berkenkamp, Philipp Hennig, Angela P. Schoellig, Andreas Krause, Stefan Schaal, Sebastian Trimpe, "Virtual vs. real: Trading off simulations and physical experiments in reinforcement learning with Bayesian optimization", In IEEE International Conference on Robotics and Automation (ICRA), pages 1557-1563, © 2017 IEEE.
- **Alonso Marco**, Philipp Hennig, Stefan Schaal, Sebastian Trimpe, "On the Design of LQR Kernels for Efficient Controller Learning", In IEEE Annual Conference on Decision and Control (CDC), pages 5193-5200, © 2017 IEEE.
- **Alonso Marco**, Alexander von Rohr, Dominik Baumann, José Miguel Hernández-Lobato, Sebastian Trimpe, "Excursion Search for Constrained Bayesian Optimization under a Limited Budget of Failures", *under review*.
- Matthias Neumann-Brosig, **Alonso Marco**, Dieter Schwarzmann, Sebastian Trimpe, "Data-efficient Auto-tuning with Bayesian Optimization: An Industrial Control Study", IEEE Transactions on Control Systems Technology, 2019.
- Alexander von Rohr, Sebastian Trimpe, **Alonso Marco**, Peer Fischer, Stefano Palagi, "Gait learning for soft microrobots controlled by light fields", IEEE International Conference on Intelligent Robots and Systems (IROS), 2018.
- Andreas Doerr, Christian Daniel, Duy Nguyen-Tuong, **Alonso Marco**, Stefan Schaal, Marc Toussain, Sebastian Trimpe, "Optimizing Long-term Predictions for Model-based Policy Search", Conference on Robot Learning (CoRL), 2017.
- Andreas Doerr, Duy Nguyen-Tuong, **Alonso Marco**, Stefan Schaal, Sebastian Trimpe, "Model-Based Policy Search for Automatic Tuning of Multivariate PID Controllers", IEEE International Conference on Robotics and Automation (ICRA), pages 5295-5301, 2017.

7.2 Future work

In this section we briefly summarize future directions and open challenges for each one of the four contributions. A more elaborated and extensive discussion can be found in the corresponding conclusions of each chapter (cf. Sec. 3.7, 4.5, 5.5 and 6.6).

In Chap. 3, we have shown how Bayesian optimization (BO) can be used to automate controller tuning in two challenging robotic platforms. Executing this idea in practice entailed many challenges, which we discuss next.

First, designing an appropriate cost functional (3.3) that expresses the desired performance objective was non-trivial. Such cost functional is described by matrices Q and R, which diagonal parameters need to be chosen beforehand. Because such parameters are chosen intuitively, we may realize that they do not quite reflect the desired performance. However, we typically realize this only after having executed the learning algorithm. Thus, iterating over these parameters is a rather slow process. This opens the question of whether is possible to also automatically learn these parameters, e.g., using meta-learning approaches (Gupta et al., 2018; Wang et al., 2018b).

Second, a robust communication framework between the robot and the optimizer was crucial (see Sec. 3.7.1 and App. A.2). In general, many technical issues can easily break the communication, such as unstable controllers (in which case the robot needs to be emergency-stopped), aggressive controllers causing huge vibrations, or other external factors difficult to foresee. Usually, the communication framework becomes robust after a few attempts to run learning experiments on the real system. However, once the programming is finished, it is unclear whether the same loop can work under different conditions (e.g., different robot, different environment, etc.). This poses an important challenge: How can we expect to have robots constantly adapting to new situations in the foreseeable future if the learning loops are so fragile and task-dependent?

Third, for the experiments in both robotic platforms (Sec. 3.4 and 3.5), we needed to thoughtfully parametrize the controllers to achieve a low dimensional tuning problem. One of the reasons for this, is that BO scales badly with the dimensionality. From a more general perspective, optimization problems with large dimensionality require a large number of data points to cover larger volumes. This opens the debate of whether global methods are generally desirable, or even useful, in settings in which data collection is scarce, like parameter optimization in robotics.

The multi-fidelity algorithm (MF-ES) proposed in Chap. 4 can trade off information vs. evaluation effort and has shown to be beneficial in practice. However,

there exist a number of issues that shall need to be addressed in future research.

First, the evaluation efforts of each information source are assumed to be given parameters. For instance, they could be the duration of a simulation and a real experiment. While this seems like a natural choice, it can be problematic in practice. For example, if the simulator is several orders of magnitude cheaper than real experiments, it might take a while until a real experiment is ever demanded by the algorithm. This can increase the total wall clock time it takes to run the optimization, as we need to add the time MF-ES needs to suggest a new point (in our current Matlab implementation, this is about 30 seconds). To alleviate this problem, one can directly tweak the evaluation effort parameters, regardless of any natural choices. This way, one can indirectly control the total duration of the experiment. However, a more fundamental methodology to treat this parameter choice is worth investigating.

Second, the correlation between the cost of the simulator and the real cost has been imposed by means of an additive kernel structure (4.2). Although this has shown to be beneficial, such structure could also be learned directly from data, in the context of multi-output Gaussian processes.

Third, from a more high-level perspective, a plausible counterargument to this work naturally arises when applying it in settings where simulations are way cheaper than real evaluation. For instance, one may wonder why restricting ourselves to a pipeline of sequential real/simulated evaluations if simulations are essentially "for free". In these settings, the proposed method herein would appear useless, as one could (i) conduct many simulations prior to evaluations on the real system, or (ii) conduct simulations in batches in parallel to the real experiment as it runs. In Sec. 4.5.1, we discuss why MF-ES remains useful in both cases.

In Chap. 5, we have shown how to design LQR kernels in a scalar problem, i.e., a one-dimensional state-space system. Although this serves as a proof of concept, scalability to multivariate systems remains an open question. More specifically, computing the nonparametric LQR kernel is already expensive in a one-dimensional setting, as it involves marginalizing out the model parameters (5.15) through a two-dimensional quadrature that needs to be numerically approximated. For a system with S states and U control inputs, the same quadrature is S^3U-dimensional, which scales badly with the dimensionality of the state. Such quadrature needs to be recomputed every time the kernel is called, which implies a non-trivial computational bottleneck, as the kernel function is called very often in BO algorithms. Hence, alternative formulations of the same LQR kernel, or efficient numerical approximations, will need to be investigated. For instance, Bayesian quadrature methods

(Kanagawa and Hennig, 2019) could be used.

Although the two algorithms presented in Chap. 6 have empirically proven to be useful in a variety of global optimization benchmarks, there exist some open problems that we discuss next.

On one hand, the algorithm Xs is built upon basic notions on excursion sets (Adler and Taylor, 2009), i.e., the set of points where a random field upcrosses a given level u. The main motivation for Xs lies in the error bound (6.2), which indicates that the larger the threshold u, the more likely it is that an "upcrossing" through u leads to the global supremum. In practice, collecting evaluations where the number of upcrossings is expected to be large is likely to result in quickly finding promising regions. Although this intuition is valid for a one-dimensional input space, its validity remains unclear in higher dimensions.

On the other hand, although empirical validations of Xs have demonstrated its effectiveness, no convergence guarantees have been provided. Although not strictly necessary, this is a desirable property that helps to understand under which circumstances the algorithm would theoretically converge to the optimum. To this end, a possibility would be to investigate the connection between the aforementioned error bound and standard convergence measures, such as *regret*. For the same reasons, convergence guarantees for XsF are also worth of being investigated.

7.3 Reflections

In this section, I briefly reflect on some relevant questions regarding the present and future of Bayesian optimization in the context of autonomous and intelligent robots. The tone and writing style of this section is intentionally less technical and more colloquial than previous sections. I intend to give my personal opinion on a short variety of present questions, very much connected with the contents and ideas presented in this thesis.

Has BO for learning on real systems come at the propitious moment?

Ever since the first BO algorithms were proposed (Kushner, 1964; Mockus et al., 1978), many optimization strategies have been elaborated (Shahriari et al., 2016). Specially during the past decade, there has been a subtle increase of BO algorithms in count and diversity (cf. Sec. 1.5.1). Most of these methods rely on modeling the performance objective with Gaussian processes (GPs). Hence, probably the published book *Gaussian processes for machine learning* (Rasmussen and Williams,

2006), which serves as a guide for newcomers to the filed, propelled the usage of BO not only in the machine (ML) community, but also in other areas, like robotics.

Roboticists started using BO for feedback controller learning (Berkenkamp et al., 2016b; Calandra et al., 2016) a few years after BO started gaining popularity. To our knowledge, these works are, together with the ones presented in this thesis, the first using BO for learning on a biped robot, a robot manipulator, and a quadrotor. Thereafter, BO was used to other robotic applications, like learning the gaits of a biped robot (Rai et al., 2018, 2019) and learning to control pneumatic artificial muscles (Büchler et al., 2019). In all cases, the core motivation for these works was the need of data-efficient learning methods capable of sparing real experiments to prevent the hardware from wearing-off and reducing human supervision.

From a financial point of view, all the above methods point to saving resources and assets. While saving production costs is generally desirable when manufacturing and maintaining robots, not all actors in the game are equally prone to it. In fact, contemporary to the above work (circa 2016), research conducted at large technological companies started breaking the paradigm of learning with a single robot on a small-scale, single-domain. For example, researchers at Google (Levine et al., 2016, 2018) used 14 networked robots to learn to grasp objects using vision feedback. After 800 000 hours of training (equivalently, 3000 hours) the failure rate of picking objects is 34% on average. Later, (Kalashnikov et al., 2018) proposed a framework for scalable robotic reinforcement learning; they used a database of 580 000 grasp attempts, collected on 7 real robot arms with minimal human intervention.

Clearly, the necessity of learning with scarce data is heavily influenced by the amount of resources available. Nowadays, universities can generally not afford having farms of robots as large technological companies do. Furthermore, reparation costs could be sometimes unaffordable, and a broken robot usually implies halting indefinitely one or more research projects in that lab. In contrast, this remains unproblematic in research carried at large companies.

These two contemporary visions on robot learning, i.e., learning with scarce data using data-efficient methods (e.g., BO) versus learning from brute-force data collection with reinforcement learning, definitely bounds the usefulness of each methodology as a function of available resources. Whether this trend continues unaltered or changes in the incoming future is of course unclear. However, this train of thought opens up the next question:

What is the place of BO for robotics in the future?

Beyond speculation, nowadays we can only state with relatively high certainty that BO will most likely remain useful for fine-tuning existing behaviors, otherwise re-

tuned manually. Same as BO can be used to improve the squatting performance of a high-dimensional robot (cf. Sec. 3.5), it could be useful in the future when robots are aiding humans in day-to-day tasks. Let us give an example: A kitchen robot has learned to cut efficiently a certain type of onions (certain shape, color, texture). If we decide to buy another type of onions, with different characteristics, most likely the robot will need to adapt to them. Under non-trivial assumptions on the robustness of the controller, the dynamics model, and reliability of the sensor equipment, the robot could learn to chop the new onions only from a few trials, at human-level rates. For this, having a method such as BO, that allows to incorporate existing contextual knowledge (e.g., how to chop the first type of onions) as prior, could help to learn from a few trials.

The list of examples where (adult) humans quickly adapt from only a few trials is interminable: Walking on a new carpet, sitting on a different chair, walking up different stairs, pouring water on a different cup, etc. Generally speaking, humans adapt effortless to momentary disturbances or constant changes in their day-to-day routines. Hence, it is not outlandish to expect the same from robots in the future. Of course, programming robots with adaptive algorithms that are very general and require little or none human supervision also implies expertise, effort and costs. This brings up the following question:

Is BO really mitigating the tuning effort, or just shifting it?

As stated in Chap. 3 and in (Berkenkamp et al., 2016b; Calandra et al., 2016; Neumann-Brosig et al., 2019), BO mitigates manual tuning and human intervention. However, BO itself also requires non-trivial efforts to be setup. Specifically, it usually requires (i) tuning hyperparameters of both, the GP model and the BO algorithm, (ii) expertise in machine learning to understand and leverage it properly, and (iii) a robust communication framework between the robot/system and the BO optimizer to minimize human intervention[1].

Tuning hyperparameters of BO algorithms is non-trivial, task-dependent and requires expertise. Commonly, they appear in the hyperprior distributions of the Gaussian process models (cf. Sec. 2.2.2), but also in BO algorithms themselves. However, these are usually a only handful of hyperparameters, which are very distinct among themselves, and understanding their functionality and implications in the algorithm only requires a high-level intuition. This is less involved than manually conducting the search of the controller parameters themselves. Furthermore, after paying the initial cost of acquiring expertise and programming the commu-

[1]For an example, see a description of the communication framework (App. A.2) we developed to learn with a two-legged robot (Sec. 3.5)

nication framework, reusing the same framework to re-tune similar plants should come at lower effort and amount of human supervision than re-tuning the controller parameters directly. From a broad perspective, we could say that BO methods are beneficial in the long-term.

Is robot-learning research going private?

In the first question posed in this section, we raised the concern of large technological companies generating large datasets by training on robot farms for long periods of time. Maybe, more important than who generates such large databases is whether they are open-source and shared with the other actors. The inevitable risk of privately acquiring and hosting such large datasets, e.g., as happens in large technological companies, is nowadays buffering the impact of robot-learning research produced in universities. If we measure impact by mean of how quickly and effectively pieces of a research field become part of our day-to-day life, such companies are definitely advantageous with respect to universities, at least within the context of robot-learning.

In an effort of contributing to the research community, a recent collaborative effort has been put in hosting an open source database in which 4 different institutions participated. Specifically, RoboNet (Dasari et al., 2019) has been proposed as an open database for sharing robot experience in pick-and-place robot manipulation tasks. It provides an initial pool of 15 million video frames, collected across 7 different robot platforms, with a large amount of data diversity. Their experiments show the possibility to generalize across new objects, tasks, camera viewpoints, grippers, or even entirely new robots.

It is the responsibility of our community as a whole to keep sharing our findings and contributing altogether to the technological progress, rather than letting new ideas, data, and algorithms hidden under the veil of competitiveness and profits.

Is BO less data-efficient than model-based reinforcement learning?

Certainly, there exist more methods in the robot learning community that aim at data-efficiency. While recent model-free reinforcement learning methods remain very data hungry (Espeholt et al., 2018; Hessel et al., 2018) to be deployed in real systems, an existing powerful tool is the use of model-based reinforcement learning (MBRL) methods. These methods aim at learning a specific task with little or no prior information by first acquiring a dynamics model, and continuously improving it in order to match a performance goal. To this end, some of the methods that are tailored to work on real robots represent the dynamics model with a neural

network (Chua et al., 2018; Nagabandi et al., 2019), while some other do it with a Gaussian process (Deisenroth and Rasmussen, 2011; Kamthe and Deisenroth, 2018). When comparing the usage in robotics of MBRL and BO, we can find drawbacks and advantages in both of them regarding data-efficiency and applicability to real robots. We discuss them below.

The greatest benefit of MBRL methods is that they leverage the time-dependent state trajectories acquired in experiments to improve the dynamics model, while BO typically compresses such data into a single performance data point. In this regard, MBRL can increase sample-efficiency, as the collected data is more extensively exploited. However, the greatest disadvantage of MBRL algorithms is that their scalability to high-dimensional robots is limited by their computational complexity. Usually, MBRL methods leverage model predictive control (MPC) and similar approaches to predict state trajectories in a receding-horizon fashion. Appropriate lengths for the prediction horizon scale badly with the complexity of the task and the state dimensionality. Hence, applicability of MBRL to high-dimensional robots (e.g., walking robots) is limited by either (i) the frequency at which the prediction loops can run, or (ii) the prediction horizon length. In any case, using MBRL for robotic settings that require fast dynamics, such as walking, remains unclear.

On the contrary, BO methods usually proceed episodically and leverage the collected data off-line, after the episode has finished. Because of this, their applicability is not limited by the complexity of the task at hand. Of course, their biggest disadvantage is the input dimensionality that they can handle. To this end, ongoing research tackles high-dimensional BO (Wang and Jegelka, 2017; Wang et al., 2016), although its applicability for learning on real robotic systems is currently being explored.

An open research question is how to combine the best from both worlds, BO and MBRL. For instance, from a rather broad perspective, MBRL could be used as in (Chua et al., 2018; Kamthe and Deisenroth, 2018) to exploit the acquired data more efficiently, while an outer BO loop iterates on the parameters of a probabilistic prior that encodes possible motions (e.g., dynamic motion primitives, central pattern generators, etc.). By injecting prior structure in the problem (the role of BO), we shall remain sample-efficient, while leveraging the time-dependent trajectory (role of MBRL).

Bibliography

P. Abbeel, M. Quigley, and A. Y. Ng. Using inaccurate models in reinforcement learning. In *International Conference on Machine Learning (ICML)*, pages 1–8, 2006.

H. Abdelrahman, F. Berkenkamp, J. Poland, and A. Krause. Bayesian optimization for maximum power point tracking in photovoltaic power plants. In *European Control Conference*, pages 2078–2083, 2016.

R. J. Adler and J. E. Taylor. *Random fields and geometry*. Springer Science and Business Media, 2009.

A. K. Akametalu, S. Kaynama, J. F. Fisac, M. N. Zeilinger, J. H. Gillula, and C. J. Tomlin. Reachability-based safe learning with Gaussian processes. In *Conference on Decision and Control (CDC)*, pages 1424–1431. IEEE, 2014.

B. D. Anderson and J. B. Moore. *Optimal Control: Linear Quadratic Methods*. Dover Publications, 1990.

B. D. O. Anderson and J. B. Moore. *Optimal Filtering*. Dover Publications, Mineola, New York, 2005.

R. Antonova, A. Rai, and C. G. Atkeson. Sample efficient optimization for learning controllers for bipedal locomotion. In *International Conference on Humanoid Robots (Humanoids)*, pages 22–28. IEEE, 2016.

R. Antonova, M. Kokic, J. A. Stork, and D. Kragic. Global search with Bernoulli alternation kernel for task-oriented grasping informed by simulation. In *Conference on Robot Learning (CoRL)*, volume 87 of *Proceedings of Machine Learning Research*, pages 641–650, 2018.

K. J. Åström and R. M. Murray. *Feedback systems: An introduction for scientists and engineers*. Princeton university press, 2008.

P. Auer. Using confidence bounds for exploitation-exploration trade-offs. *The Journal of Machine Learning Research (JMLR)*, 3:397–422, 2003.

S. Bansal, R. Calandra, T. Xiao, S. Levine, and C. J. Tomiin. Goal-driven dynamics learning via bayesian optimization. In *Conference on Decision and Control (CDC)*, pages 5168–5173. IEEE, 2017.

S. Bennett. Nicholas minorsky and the automatic steering of ships. *Control Systems Magazine*, 4(4):10–15, 1984.

S. Bennett. A brief history of automatic control. *IEEE Control Systems Magazine*, 16(3):17–25, 1996.

F. Berkenkamp, A. Krause, and A. P. Schoellig. Bayesian optimization with safety constraints: Safe and automatic parameter tuning in robotics. Technical report, 2016a. URL https://arxiv.org/abs/1602.04450.

F. Berkenkamp, A. P. Schoellig, and A. Krause. Safe controller optimization for quadrotors with Gaussian processes. In *International Conference on Robotics and Automation (ICRA)*, pages 491–496. IEEE, 2016b.

G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba. OpenAI Gym, 2016.

D. Büchler, R. Calandra, and J. Peters. Learning to control highly accelerated ballistic movements on muscular robots. *arXiv preprint arXiv:1904.03665*, 2019.

R. H. Byrd, P. Lu, J. Nocedal, and C. Zhu. A limited memory algorithm for bound constrained optimization. *SIAM Journal on scientific computing*, 16(5):1190–1208, 1995.

R. Calandra, A. Seyfarth, J. Peters, and M. P. Deisenroth. An experimental comparison of Bayesian optimization for bipedal locomotion. In *International Conference on Robotics and Automation (ICRA)*, pages 1951–1958. IEEE, 2014.

R. Calandra, A. Seyfarth, J. Peters, and M. P. Deisenroth. Bayesian optimization for learning gaits under uncertainty. *Annals of Mathematics and Artificial Intelligence*, 76(1-2):5–23, 2016.

M. Charbonneau, V. Modugno, F. Nori, G. Oriolo, D. Pucci, and S. Ivaldi. Learning robust task priorities of QP-based whole-body torque-controllers. In *International Conference on Humanoid Robots (Humanoids)*, pages 1–9. IEEE, 2018.

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. *IEEE Transactions on Robotics*, 2019.

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 (NeurIPS)*, pages 4754–4765, 2018.

D. W. Clarke, P. P. Kanjilal, and C. Mohtadi. A generalized LQG approach to self-tuning control – Part I. Aspects of design. *International Journal of Control*, 41(6):1509–1523, 1985.

M. Cutler and J. P. How. Efficient reinforcement learning for robots using informative simulated priors. In *International Conference on Robotics and Automation*, pages 2605–2612. IEEE, 2015.

M. Cutler, T. J. Walsh, and J. P. How. Real-world reinforcement learning via multifidelity simulators. *IEEE Transactions on Robotics*, 31(3):655–671, 2015.

S. Dasari, F. Ebert, S. Tian, S. Nair, B. Bucher, K. Schmeckpeper, S. Singh, S. Levine, and C. Finn. Robonet: Large-scale multi-robot learning. In *CoRL 2019: Volume 100 Proceedings of Machine Learning Research*, 2019.

E. A. Daxberger and B. K. H. Low. Distributed batch Gaussian process optimization. In *International Conference on Machine Learning (ICML)*, volume 70, pages 951–960. JMLR. org, 2017.

L. De Haan and A. Ferreira. *Extreme value theory: An introduction*. Springer Science & Business Media, 2007.

M. Deisenroth and C. E. Rasmussen. PILCO: A model-based and data-efficient approach to policy search. In *International Conference on machine learning (ICML)*, pages 465–472, 2011.

A. El-Fakdi and M. Carreras. Two-step gradient-based reinforcement learning for underwater robotics behavior learning. *Robotics and Autonomous Systems*, 61(3): 271–282, 2013.

D. B. Ender. Process control performance: Not as good as you think. *Control Engineering*, 40(10):180–190, 1993.

L. Espeholt, H. Soyer, R. Munos, K. Simonyan, V. Mnih, T. Ward, Y. Doron, V. Firoiu, T. Harley, I. Dunning, et al. IMPALA: Scalable distributed deep-RL with importance weighted actor-learner architectures. In *International Conference on Machine Learning (ICML)*, pages 1407–1416, 2018.

A. I. J. Forrester, A. Sóbester, and A. J. Keane. Multi-fidelity optimization via surrogate modelling. *Royal Society of London A: Mathematical, Physical and Engineering Sciences*, 463(2088):3251–3269, 2007.

P. Frazier, W. Powell, and S. Dayanik. The knowledge-gradient policy for correlated normal beliefs. *INFORMS Journal on Computing*, 21(4):599–613, 2009.

J. R. Gardner, M. J. Kusner, Z. E. Xu, K. Q. Weinberger, and J. P. Cunningham. Bayesian optimization with inequality constraints. In *International Conference on Machine Learning (ICML)*, pages 937–945, 2014.

M. A. Gelbart, J. Snoek, and R. P. Adams. Bayesian optimization with unknown constraints. In *Conference on Uncertainty in Artificial Intelligence*, pages 250–259, 2014.

R. B. Gramacy and H. Lee. Optimization under unknown constraints. *Bayesian Statistics 9*, 2011.

M. J. Grimble. Implicit and explicit LQG self-tuning controllers. *Automatica*, 20 (5):661–669, 1984.

A. Gupta, R. Mendonca, Y. Liu, P. Abbeel, and S. Levine. Meta-reinforcement learning of structured exploration strategies. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 5302–5311, 2018.

S. Ha, P. Xu, Z. Tan, S. Levine, and J. Tan. Learning to walk in the real world with minimal human effort. *arXiv preprint arXiv:2002.08550*, 2020.

M. Havasi, R. Peharz, and J. M. Hernández-Lobato. Minimal random code learning: Getting bits back from compressed model parameters. In *International Conference on Learning Representations (ICLR)*. OpenReview.net, 2018.

P. Hennig and C. J. Schuler. Entropy search for information-efficient global optimization. *The Journal of Machine Learning Research (JMLR)*, 13(1):1809–1837, 2012.

J. M. Hernández-Lobato, M. W. Hoffman, and Z. Ghahramani. Predictive entropy search for efficient global optimization of black-box functions. In *Advances in neural information processing systems (NeurIPS)*, pages 918–926, 2014.

J. M. Hernández-Lobato, M. A. Gelbart, R. P. Adams, M. W. Hoffman, and Z. Ghahramani. A general framework for constrained Bayesian optimization using information-based search. *The Journal of Machine Learning Research (JMLR)*, 17(1):5549–5601, 2016.

M. Hessel, J. Modayil, H. Van Hasselt, T. Schaul, G. Ostrovski, W. Dabney, D. Horgan, B. Piot, M. Azar, and D. Silver. Rainbow: Combining improvements in deep reinforcement learning. In *AAAI Conference on Artificial Intelligence*, 2018.

M. D. Hoffman and A. Gelman. The No-U-Turn sampler: Adaptively setting path lengths in Hamiltonian Monte Carlo. *Journal of Machine Learning Research (JMLR)*, 15(1):1593–1623, 2014.

J. Issac, M. Wüthrich, C. Garcia Cifuentes, J. Bohg, S. Trimpe, and S. Schaal. Depth-based object tracking using a robust Gaussian filter. In *International Conference on Robotics and Automation (ICRA)*. IEEE, 2016. URL http://arxiv.org/abs/1602.06157.

M. Jamil and X.-S. Yang. A literature survey of benchmark functions for global optimization problems. *International Journal of Mathematical Modelling and Numerical Optimisation*, 4(2):150—-194, 2013.

J. W. Jawitz. Moments of truncated continuous univariate distributions. *Advances in water resources*, 27(3):269–281, 2004.

M. Jelali. An overview of control performance assessment technology and industrial applications. *Control engineering practice*, 14(5):441–466, 2006.

D. R. Jones, M. Schonlau, and W. J. Welch. Efficient global optimization of expensive black-box functions. *Journal of Global optimization*, 13(4):455–492, 1998.

D. Kalashnikov, A. Irpan, P. Pastor, J. Ibarz, A. Herzog, E. Jang, D. Quillen, E. Holly, M. Kalakrishnan, V. Vanhoucke, et al. Scalable deep reinforcement learning for vision-based robotic manipulation. In *Conference on Robot Learning (CoRL)*, pages 651–673, 2018.

R. E. Kalman. When is a linear control system optimal? *Journal of Fluids Engineering*, 86(1):51–60, 1964.

S. Kamthe and M. Deisenroth. Data-efficient reinforcement learning with probabilistic model predictive control. In A. Storkey and F. Perez-Cruz, editors, *International Conference on Artificial Intelligence and Statistics (AISTATS)*, volume 84 of *Proceedings of Machine Learning Research*, pages 1701–1710, 2018.

M. Kanagawa and P. Hennig. Convergence guarantees for adaptive Bayesian quadrature methods. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 6234–6245, 2019.

K. Kandasamy, G. Dasarathy, J. Oliva, J. Schneider, and B. Poczos. Multi-fidelity gaussian process bandit optimisation. *Journal of Artificial Intelligence Research*, 66:151–196, 2019.

M. C. Kennedy and A. O'Hagan. Predicting the output from a complex computer code when fast approximations are available. *Biometrika*, 87(1):1–13, 2000.

J. Kober, J. A. Bagnell, and J. Peters. Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research (IJRR)*, 32(11):1238–1274, 2013.

H. Kong, G. Goodwin, and M. Seron. A revisit to inverse optimality of linear systems. *International Journal of Control*, 85(10):1506–1514, 2012.

A. Krause and C. S. Ong. Contextual Gaussian process bandit optimization. In *Neural Information Processing Systems (NeurIPS)*, pages 2447–2455, 2011.

A. Kucukelbir, D. Tran, R. Ranganath, A. Gelman, and D. M. Blei. Automatic differentiation variational inference. *The Journal of Machine Learning Research*, 18(1):430–474, 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.

S. Levine, P. Pastor, A. Krizhevsky, and D. Quillen. Learning hand-eye coordination for robotic grasping with large-scale data collection. In *International symposium on experimental robotics*, pages 173–184. Springer, 2016.

S. Levine, P. Pastor, A. Krizhevsky, J. Ibarz, and D. Quillen. Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection. *The International Journal of Robotics Research (IJRR)*, 37(4-5):421–436, 2018.

W. Li and E. Todorov. Iterative linear quadratic regulator design for nonlinear biological movement systems. In *International Conference on Informatics in Control, Automation and Robotics*, pages 222–229, 2004.

G. Lindgren. Lectures on stationary stochastic processes. *PhD course of Lund's University*, 2006.

D. J. Lizotte, T. Wang, M. H. Bowling, and D. Schuurmans. Automatic gait optimization with Gaussian process regression. In *International Joint Conference on Artificial Intelligence*, volume 7, pages 944–949, 2007.

A. Marco. Gaussian process optimization for self-tuning control. Master's thesis, Polytechnic University of Catalonia (BarcelonaTech), 2015.

A. Marco, P. Hennig, J. Bohg, S. Schaal, and S. Trimpe. Automatic LQR tuning based on Gaussian process optimization: Early experimental results. In *Second Machine Learning in Planning and Control of Robot Motion Workshop (at IROS)*, 2015.

A. Marco, P. Hennig, J. Bohg, S. Schaal, and S. Trimpe. Automatic LQR tuning based on Gaussian process global optimization. In *international conference on robotics and automation (ICRA)*, pages 270–277. IEEE, 2016.

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 *IEEE International Conference on Robotics and Automation (ICRA)*, pages 1557–1563. IEEE, 2017.

A. Marco, A. von Rohr, D. Baumann, J. M. Hernández-Lobato, and S. Trimpe. Excursion search for constrained Bayesian optimization under a limited budget of failures. In *International Conference on Machine Learning (ICML)*, 2020. URL https://arxiv.org/abs/1602.04450.

S. Mason, L. Righetti, and S. Schaal. Full dynamics LQR control of a humanoid robot: An experimental study on balancing and squatting. In *International Conference on Humanoid Robots (Humanoids)*, pages 374–379. IEEE-RAS, 2014.

J. C. Maxwell. I. on governors. *Proceedings of the Royal Society of London*, (16): 270–283, 1868.

J. R. Medina, S. Endo, and S. Hirche. Impedance-based Gaussian processes for predicting human behavior during physical interaction. In *International Conference on Robotics and Automation (ICRA)*, pages 3055–3061. IEEE, 2016.

J. H. Metzen. Active contextual entropy search. In *BayesOpt Workshop at 29th Annual Conference on Neural Information Processing Systems*, 2015.

N. Minorsky. Directional stability of automatically steered bodies. *Journal of the American Society for Naval Engineers*, 34(2):280–309, 1922.

B. T. Mirletz, I.-W. Park, R. D. Quinn, and V. SunSpiral. Towards bridging the reality gap between tensegrity simulation and robotic hardware. In *International Conference on Intelligent Robots and Systems (IROS)*, pages 5357–5363. IEEE, 2015.

J. Močkus. On Bayesian methods for seeking the extremum. In *Optimization Techniques IFIP Technical Conference*, pages 400–404. Springer, 1975.

J. Mockus, V. Tiesis, and A. Zilinskas. Toward global optimization: Bayesian methods for seeking the extremum, 1978.

K. Mülling, J. Kober, O. Kroemer, and J. Peters. Learning to select and generalize striking movements in robot table tennis. *The International Journal of Robotics Research*, 32(3):263–279, 2013.

I. Murray, R. Adams, and D. MacKay. Elliptical slice sampling. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, pages 541–548, 2010.

A. Nagabandi, K. Konoglie, S. Levine, and V. Kumar. Deep dynamics models for learning dexterous manipulation. *arXiv preprint arXiv:1909.11652*, 2019.

M. Neumann-Brosig, A. Marco, D. Schwarzmann, and S. Trimpe. Data-efficient autotuning with Bayesian optimization: An industrial control study. *IEEE Transactions on Control Systems Technology*, 2019.

K. Ogata and Y. Yang. *Modern control engineering*, volume 5. Prentice Hall, NJ, 2010.

J. Peters and S. Schaal. Policy gradient methods for robotics. In *International Conference on Intelligent Robots and Systems (IROS)*, pages 2219–2225, 2006.

V. Picheny. A stepwise uncertainty reduction approach to constrained global optimization. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, pages 787–795, 2014.

M. Poloczek, J. Wang, and P. Frazier. Multi-information source optimization. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 4288–4298, 2017.

M. J. Powell. A direct search optimization method that models the objective and constraint functions by linear interpolation. In *Advances in optimization and numerical analysis*, pages 51–67. Springer, 1994.

Quanser. Self-erecting single inverted pendulum – Instructor manual". Technical Report 516, rev. 4.1, 2015.

A. Rai, R. Antonova, S. Song, W. Martin, H. Geyer, and C. Atkeson. Bayesian optimization using domain knowledge on the ATRIAS biped. In *International Conference on Robotics and Automation (ICRA)*, pages 1771–1778. IEEE, 2018.

A. Rai, R. Antonova, F. Meier, and C. G. Atkeson. Using simulation to improve sample-efficiency of Bayesian optimization for bipedal robots. *Journal of machine learning research (JMLR)*, 20(49):1–24, 2019.

C. E. Rasmussen and C. K. Williams. *Gaussian Processes for Machine Learning*. MIT Press, 2006.

L. Righetti, M. Kalakrishnan, P. Pastor, J. Binney, J. Kelly, R. C. Voorhies, G. S. Sukhatme, and S. Schaal. An autonomous manipulation system based on force control and optimization. *Autonomous Robots*, 36(1-2):11–30, 2014.

J. Roberts, I. Manchester, and R. Tedrake. Feedback controller parameterizations for reinforcement learning. In *IEEE Symposium on Adaptive Dynamic Programming And Reinforcement Learning*, pages 310–317, 2011.

D. Rodriguez, A. Brandenburger, and S. Behnke. Combining simulations and real-robot experiments for bayesian optimization of bipedal gait stabilization. In *Robot World Cup*, pages 70–82. Springer, 2018.

S. Schaal. Learning from demonstration. *Advances in Neural Information Processing Systems (NeurIPS)*, pages 1040–1046, 1997.

S. Schaal. The SL simulation and real-time control software package. Technical report, 2009. URL http://www-clmc.usc.edu/publications/S/schaal-TRSL.pdf.

S. Schaal and C. G. Atkeson. Learning control in robotics. *IEEE Robotics & Automation Magazine*, 17(2):20–29, 2010.

B. Schölkopf and A. J. Smola. *Learning with kernels*. MIT Press, 2002.

M. Schonlau, W. J. Welch, and D. R. Jones. Global versus local search in constrained optimization of computer models. *Lecture Notes-Monograph Series*, pages 11–25, 1998.

J. Schreiter, D. Nguyen-Tuong, M. Eberts, B. Bischoff, H. Markert, and M. Toussaint. Safe exploration for active learning with Gaussian processes. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 133–149. Springer, 2015.

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. *IEEE*, 104(1):148–175, 2016.

J. C. Spall. Simultaneous perturbation stochastic approximation. *Introduction to Stochastic Search and Optimization: Estimation, Simulation, and Control*, pages 176–207, 2003.

N. Srinivas, A. Krause, S. Kakade, and M. Seeger. Gaussian process optimization in the bandit setting: No regret and experimental design. In *International Conference on Machine Learning (ICML)*, pages 1015–1022, 2010.

Y. Sui, A. Gotovos, J. Burdick, and A. Krause. Safe exploration for optimization with Gaussian processes. In *International Conference on Machine Learning (ICML)*, pages 997–1005, 2015.

R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. MIT press, 1998.

K. Swersky, J. Snoek, and R. P. Adams. Multi-Task Bayesian Optimization. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 2004–2012, 2013.

M. E. Taylor and P. Stone. Transfer learning for reinforcement learning domains: A survey. *Journal of Machine Learning Research (JMLR)*, 10:1633–1685, 2009.

M. Tesch, J. Schneider, and H. Choset. Using response surfaces and expected improvement to optimize snake robot gait parameters. In *International Conference on Intelligent Robots and Systems*, pages 1069–1074. IEEE, 2011.

M. Titsias and N. D. Lawrence. Bayesian gaussian process latent variable model. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, pages 844–851, 2010.

E. Todorov, T. Erez, and Y. Tassa. MuJoCo: A physics engine for model-based control. In *International Conference on Intelligent Robots and Systems (IROS)*, pages 5026–5033. IEEE, 2012.

S. Trimpe and R. D'Andrea. The Balancing Cube: A dynamic sculpture as test bed for distributed estimation and control. *IEEE Control Systems Magazine*, 32(6): 48–75, 2012.

S. Trimpe, A. Millane, S. Doessegger, and R. D'Andrea. A self-tuning LQR approach demonstrated on an inverted pendulum. In *IFAC World Congress*, pages 11281–11287, 2014.

H. Vanchinathan. *Learning to Recommend: Interactive Learning with Limited Feedback*. PhD thesis, ETH Zurich, 2015.

J. Villemonteix, E. Vazquez, and E. Walter. An informational approach to the global optimization of expensive-to-evaluate functions. *Journal of Global Optimization*, 44(4):509–534, 2009.

A. von Rohr, S. Trimpe, A. Marco, P. Fischer, and S. Palagi. Gait learning for soft microrobots controlled by light fields. In *IEEE International Conference on Intelligent Robots and Systems (IROS)*, pages 6199–6206, 2018.

Z. Wang and S. Jegelka. Max-value entropy search for efficient Bayesian optimization. In *International Conference on Machine Learning (ICML)*, pages 3627–3635, 2017.

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.

Z. Wang, C. Li, S. Jegelka, and P. Kohli. Batched high-dimensional Bayesian optimization via structural kernel learning. In *International Conference on Machine Learning (ICML)*, pages 3656–3664. JMLR. org, 2017.

Z. Wang, C. Gehring, P. Kohli, and S. Jegelka. Batched large-scale Bayesian optimization in high-dimensional spaces. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, pages 745–754, 2018a.

Z. Wang, B. Kim, and L. P. Kaelbling. Regret bounds for meta Bayesian optimization with an unknown Gaussian process prior. In *Neural Information Processing Systems (NeurIPS)*, 2018b.

A. W. Winkler, F. Farshidian, M. Neunert, D. Pardo, and J. Buchli. Online walking motion and foothold optimization for quadruped locomotion. In *International Conference on Robotics and Automation (ICRA)*, pages 5308–5313. IEEE, 2017.

D. H. Wolpert and W. G. Macready. No free lunch theorems for optimization. *IEEE transactions on evolutionary computation*, 1(1):67–82, 1997.

J. Wu, M. Poloczek, A. G. Wilson, and P. Frazier. Bayesian optimization with gradients. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 5267–5278, 2017.

J. Wu, S. Toscano-Palmerin, P. I. Frazier, and A. G. Wilson. Practical multi-fidelity Bayesian optimization for hyperparameter tuning. In *Uncertainty in Artificial Intelligence(UAI)*, 2019.

M. H. Yeganegi, M. Khadiv, S. A. A. Moosavian, J. Zhu, A. Del Prete, and L. Righetti. Robust humanoid locomotion using trajectory optimization and sample-efficient learning. In *International Conference on Humanoid Robots (Humanoids)*, pages 170–177, 2019.

K. Yuan, I. Chatzinikolaidis, and Z. Li. Bayesian optimization for whole-body control of high-degree-of-freedom robots through reduction of dimensionality. *Robotics and Automation Letters (RAL)*, 4(3):2268–2275, 2019.

K. Zhou, J. Doyle, and K. Glover. *Robust and optimal control*, volume 40. Prentice Hall International, 1996.

S. Zhu, D. Surovik, K. Bekris, and A. Boularias. Efficient model identification for tensegrity locomotion. In *International Conference on Intelligent Robots and Systems (IROS)*, pages 2985–2990. IEEE, 2018.

Appendix A

Automatic LQR tuning of a humanoid robot

In this appendix, we discuss further details about the implementation choices and the experimental setup used in Sec. 3.5.

A.1 Implementation choices

The robot Hermes performs a squatting task by tracking a given sinusoidal reference. Such reference $\boldsymbol{s}_k^{\text{ref}} \in \mathbb{R}^S$ is only non-zero in the vertical component of the center of mass (COM) of the robot. The corresponding COM part $r_k = \left[r_k^{\text{X}}, r_k^{\text{Y}}, r_k^{\text{Z}}\right]$ and its velocity $\dot{r}_k$ are given as

$$\begin{aligned} r_k^{\text{Z}} &= s_0^{\text{Z}} - a(1 + \cos(2\pi b \Delta\tau k)) \\ \dot{r}_k^{\text{Z}} &= a 2\pi b \Delta\tau \sin(2\pi b \Delta\tau k)) \\ r_k^{\text{X}} &= \dot{r}_k^{\text{X}} = r_k^{\text{Y}} = \dot{r}_k^{\text{Y}} = 0, \end{aligned} \tag{A.1}$$

where $a = 0.1$ m is the amplitude, $b = 0.8$ Hz is the frequency, $\Delta\tau = 10^{-3}$ s is sampling time, k is the time step and s_0^{Z} is the vertical component of the position of the center of mass at the initial time.

For simplicity, the performance matrices are such that $R = I$, where I is the identity matrix. Furthermore, we penalize the joint positions and the position and orientations of the center of mass in matrix Q as follows

$$\begin{aligned} Q_{18,18} = Q_{19,19} = Q_{20,20} = 10^2; Q_{3,3} = Q_{5,5} = Q_{10,10} = Q_{12,12} = 10^3 \\ Q_{6,6} = Q_{13,13} = Q_{1,1} = Q_{8,8} = Q_{9,9} = Q_{2,2} = Q_{17,17} = 10^5 \\ Q_{15,15} = Q_{16,16} = 10^4; Q_{4,4} = Q_{11,11} = 10^7, \end{aligned} \tag{A.2}$$

and $Q_{i,i} = 1$ for the rest of the indices, with $i = \{1, \ldots, 40\}$.

A.2 Experimental setup

The robot Hermes was cable-communicated with a desktop PC, running Ubuntu 16.04 and fully controlled using SL (Schaal, 2009) at 1kHz. Hermes is a hydraulically-actuated torque-controlled humanoid, manufactured by Sarcos. Its joints need from an external high-pressure oil pump to function. More details about Hermes can be found in (Mason et al., 2014).

We used ROS to communicate the robot and Entropy Search (ES) by sending/receiving data via publishers/subscribers. The c++ code for the communication framework is publicly available at `https://github.com/alonrot/userES_pubsub_lqr`. In the following, we briefly explain the main aspects of such communication scheme.

First, we start ES (see Alg. 2). When CostEvaluation() is called, the communication between ES and Hermes begins. Therein, ES publishes the controller parameters to Hermes, and a new squatting experiment is started. Once the experiment has terminated, the corresponding cost value is listened by ES, which continues executing until a pre-established stopping criterion is met.
Once a new experiment has started, the three following steps are conducted:

1) Assuming the robot is up and running, bring the robot to the *home position*, regardless of its current state.
2) Perform squatting task for 30 seconds, while checking for unstability flags.
3) Report "unstable" if the robot surpasses pre-established joint limits, or report the cost value in case the task is successfully finished.

Step 1 is needed to ensure all experiments start from the same initial joint configuration, i.e., the home position. There exist many technical issues can easily break the ES optimization loop. For example, loosing communication with the robot or an aggressive controller destabilizing the robot. The former is ensured by checking a "alive/dead" flag published from the robot side, while the latter is achieved by making sure that certain joint limits are not exceeded. If exceeded, a stopping flag was triggered to safely freeze the robot. However, in many cases, too aggressive controllers would cause sudden violent movements in the robot, or high vibrations (due to the controller amplifying measurement noise). Then, an emergency stop had to be pressed to stop the oil pump, after which the robot was not operative anymore and had to be manually restarted. At the same time, ES was informed about the standby situation, to wait indefinitely for the robot to be restarted.

Appendix B

Controller learning under limited budget of failures

This appendix presents supplementary material to Chap. 6, such as derivations, clarifications and additional results.

B.1 Additional details to the derivation of XSF

Herein, the derivation of (6.6), shown in Sec. 6.2.2, is complemented with two additional insights. First, in App. B.1.1, we show how the integral from (6.6) resolves into an analytical expression. Then, in App. B.1.2, we reason about adding $\{x, u\}$ to the dataset $\mathcal{D}_t^f$ as a *virtual* observation.

B.1.1 Analytical expression for the integral in (6.6)

The integral from (6.6) can be split in two parts

$$\int_{-\infty}^{+\infty} |f'| p(f'|\tilde{\mathcal{D}}) \mathrm{d}f' = -\int_{-\infty}^{0} f' p(f'|\tilde{\mathcal{D}}) \mathrm{d}f' + \int_{0}^{+\infty} f' p(f'|\tilde{\mathcal{D}}) \mathrm{d}f',$$

where the placeholder $\tilde{\mathcal{D}} = \mathcal{D}_t^f \cup \{x, u\}$ is used for simplicity, and the dependency of f' on x is implicit, and also omitted. Since $f' \sim \mathcal{N}(f'; \mu'(x), \nu^2(x))$ is Gaussian distributed, each of the integrals above can be seen as the expected value of an unnormalized truncated normal distribution with support $[-\infty, 0]$, and $[0, +\infty]$, respectively (Jawitz, 2004). These expectations are given by

$$\begin{aligned}
\int_{-\infty}^{0} f' p(f'|\tilde{\mathcal{D}}) \mathrm{d}f' &= \mu'(x) Z_u(x) - \nu(x) \phi\left(-\tfrac{\mu'(x)}{\nu(x)}\right) \\
\int_{0}^{+\infty} f' p(f'|\tilde{\mathcal{D}}) \mathrm{d}f' &= \mu'(x) Z_l(x) + \nu(x) \phi\left(-\tfrac{\mu'(x)}{\nu(x)}\right),
\end{aligned}$$

where $Z_l(x) = \Phi\left(\frac{\mu'(x)}{\nu(x)}\right)$, $Z_u(x) = \Phi\left(\frac{-\mu'(x)}{\nu(x)}\right)$, ϕ is the density of a standard normal distribution and Φ is its cumulative density function. We make use of the definition

$\Phi(a) = \frac{1}{2}(1 + \operatorname{erf}\left(a/\sqrt{2}\right))$, where $\operatorname{erf}(\cdot)$ is the error function, to compute $\Phi(a) - \Phi(-a) = \operatorname{erf}\left(a/\sqrt{2}\right)$. Then, $Z_l(x) - Z_u(x) = \operatorname{erf}\left(\frac{\mu'(x)}{\sqrt{2}\nu(x)}\right)$, and the integral can be solved analytically as

$$\begin{aligned}\int_{-\infty}^{+\infty} |f'| p(f'|\tilde{\mathcal{D}}) \mathrm{d}f' &= \mu'(x)(Z_l(x) - Z_u(x)) + 2\nu(x)\phi\left(\tfrac{\mu'(x)}{\nu(x)}\right) \\ &= \mu'(x)\operatorname{erf}\left(\tfrac{\mu'(x)}{\sqrt{2}\nu(x)}\right) + 2\nu(x)\phi\left(\tfrac{\mu'(x)}{\nu(x)}\right).\end{aligned}$$

Then, (6.6) follows.

B.1.2 Virtual observation $\{x, u\}$

The posterior of the process derivative $p(f'|x, u, \mathcal{D}_t^f)$ is a Gaussian density and can be seen as conditioning $f'(x)$ on an extended dataset that includes $\{x, u\}$ as a virtual observation. In the following, we briefly discuss this.

Since differentiation is a linear operation, the derivative of a GP remains a GP (Rasmussen and Williams, 2006, Sec. 9.4). Furthermore, the joint density between a process value $f(x)$, its derivative $f'(x)$ and the dataset $\{X, y\}$ is Gaussian (Wu et al., 2017)

$$p(y, f, f'|x, X) = \mathcal{N}\left(\begin{bmatrix} y \\ f \\ f' \end{bmatrix}; \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} \tilde{K}(X,X) & K(X,x) & K'(X,x) \\ K(x,X) & K(x,x) & K'(x,x) \\ K'(x,X) & K'(x,x) & K''(x,x) \end{bmatrix}\right),$$

where $\tilde{K}(X,X) = K(X,X) + \sigma_{\mathrm{n}}^2 I$, $K'(X,x) = \partial K(X,x)/\partial x$, $K''(x,x) = \partial^2 K(x,x)/\partial x^2$, and the prior mean of the GP is assumed to be zero. Then, the conditional $p(f'|f, x, \mathcal{D}_t^f) = \mathcal{N}(f'; \mu'(x; f), \nu^2(x))$ is also Gaussian, and can be obtained using Gaussian algebra (Rasmussen and Williams, 2006, A. 2). The mean $\mu'(x; f)$ depends on the random variable f as

$$\mu'(x; f) = \begin{bmatrix} K'(x,X) & K'(x,x) \end{bmatrix} \begin{bmatrix} \tilde{K}(X,X) & K(X,x) \\ K(x,X) & K(x,x) \end{bmatrix}^{-1} \begin{bmatrix} y \\ f \end{bmatrix}. \tag{B.1}$$

The desired Gaussian density $\mathcal{N}(f'; \mu'(x; u), \nu^2(x))$ is obtained by replacing the value f in the expression for the mean (B.1). Thereby, $\{x, u\}$ appears in (B.1) as an additional *virtual* observation at location x added to the existing dataset $\{X, y\}$, in shorthand notation: $p(f'|u, x, \mathcal{D}_t^f) = p(f'|\mathcal{D}_t^f \cup \{x, u\})$.

B.2 Fréchet distribution

In this section, we present a brief analysis on why assuming a Fréchet distribution is less error prone in practice than using the Gumbel distribution, when it comes to

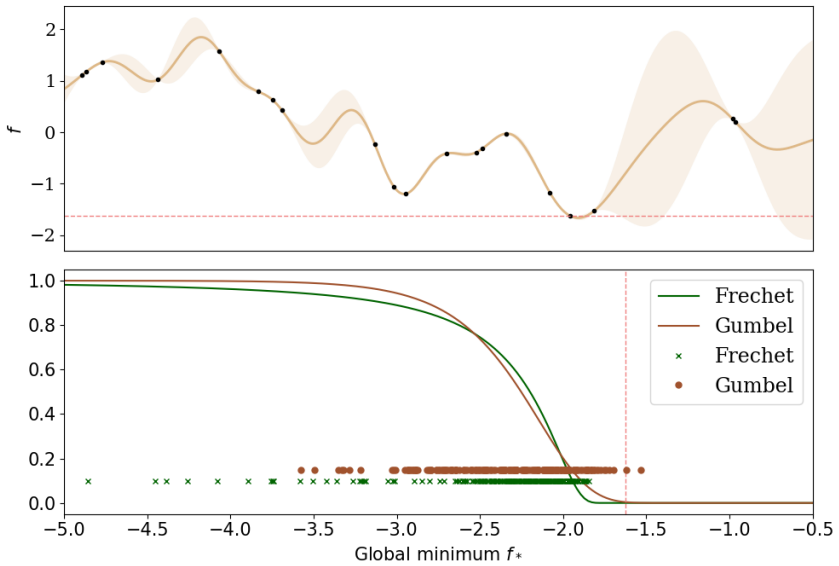


Figure B.1: (top) Gaussian process, and η (red dashed line). (bottom) Survival functions for both Gumbel, and Fréchet distributions. Samples from the Gumbel (crosses) and from the Fréchet (circles) distribution are shown.

model the distribution over the global minimum $p(f_*)$. This analysis complements the methodology presented in Sec. 6.3.1.

When modeling $p(f_*)$ with the Gumbel distribution and sampling from it, some samples of the global minimum can lie above the lowest observation so far $\eta = \min(y(x_1), \ldots, y(x_t))$, with non-zero probability, which is unrealistic. This can be explicitly avoided by using the Fréchet distribution which, contrary to Gumbel, has zero probability mass near η. We illustrate this with an example, in which a GP with zero mean, unit variance, and squared exponential kernel is considered, conditioned on 20 observations sampled from the GP prior. We discretize the domain in 200 points and sample the resulting GP posterior at them. In Fig. B.1, we see that a portion of the Gumbel samples lie above η. To show consistency, we sample the posterior GP 100 times and average the number of times that Gumbel exceeds η, i.e., $1.60 \pm 1.22\%$ of the cases, while the Fréchet distribution exceeds η in 0% of the cases.

B.3 Algorithm and complexity

Herein, we discuss pseudocode for XsF and its computational complexity.

B.3.1 XsF algorithm

Pseudocode for XsF is shown in Alg. 4. The decision boundary ρ_{b} is used to switch between safe search (cf. (6.14)) and risky search (cf. (6.14)). The algorithm returns the location where the mean of the posterior GP is minimized without violating the probabilistic constraints. To abbreviate, we have used the placeholder $\zeta(x) = \prod_{i=1}^{K} \Pr(g_j(x) \leq 0)$.

We do not explicitly discuss Xs, as it simply comprises a standard Bayesian optimization loop, which involves (i) computing samples of the global minimum, and (ii) maximizing the acquisition function (6.9).

B.3.2 Complexity

At each iteration, the most expensive operations required to obtain (6.14) and (6.14) are: (a) obtaining samples from the global minimum $p(f_*)$ and (b) maximizing the acquisition function using local optimization with random restarts.

As explained in (Wang and Jegelka, 2017), obtaining S samples from $p(f_*)$ involves discretizing the input domain and performing a binary search, which has a total cost of $\mathcal{O}(S + N_d \log(1/\kappa))$, where N_d is the size of the discretization grid, and κ is the accuracy of the binary search.

Each call to the acquisition function α_{X} (6.9), has a cost of $\mathcal{O}(SD)$ where D is the dimensionality of the input space. Then, assuming R random restarts, and M maximum number of function calls, the total cost of XsF in per iteration the worst case scenario is given by $\mathcal{O}(MRD(S+1) + N_d \log(1/\kappa) + (G+1)(N_{\text{obs}}+1)^3)$. The last term is the cost of inverting the Gram matrix, needed for GP predictions (cf. (B.1)), after having collected N_{obs} observations, and having G constraints. When setting $G = 0$, we obtain the computational cost of Xs, as it also requires gathering samples from $p(f_*)$ and local optimization with random restarts.

B.4 Implementation details

Both, Xs and XsF are developed using BoTorch[1], a Python library for Bayesian optimization that serves as a low-level API for building and optimizing new acquisition functions and fitting GP models. It makes use of SciPy Python optimizers[2] for estimating the GP hyperparameters and optimizing the acquisition function through local optimization with random restarts. In all cases we allow 10 random restarts and use L-BFGS-B (Byrd et al., 1995) as local optimization algorithm. Currently,

[1] botorch.org/docs/introduction.html
[2] docs.scipy.org/doc/scipy/reference/tutorial/optimize.html

Algorithm 4 Failures-aware Excursion Search (XsF)

Input: $T, B, S, \mathcal{D}_0^f, \mathcal{D}_0^g, \rho_{\text{safe}}, \rho_{\text{risk}}, \rho_{\text{b}}, \rho_0$
for $t = 1$ **to** T **do**
  $\rho_t \leftarrow$ UpdateDecisionBoundary(ρ_{t-1})
  $f_* \leftarrow$ SampleGlobalMinimum(S)
  if $\rho_t > \rho_{\text{b}}$ **then**
    $x_t \leftarrow \arg\max_{x \in \mathcal{X}} \ \alpha_{\text{X}}(x; f_*)$ s.t. $\zeta(x) \geq \rho_t$ (6.14)
  else
    $x_t \leftarrow \arg\max_{x \in \mathcal{X}} \ \alpha_{\text{X}}(x; f_*)\zeta(x)$ (6.14)
  end if
  EvaluateAndUpdateGPs(x_t)
end for
$x_*^{\text{c}} \leftarrow \arg\min_{x \in \mathcal{X}} \ \mu(x)$ s.t. $\zeta(x) \geq \rho_{\text{safe}}$
Return: x_*^{c}

procedure UpdateDecisionBoundary(ρ_t)
  $z_t \leftarrow \Phi^{-1}(\rho_t)$
  $u_t \leftarrow u_t(\Delta B_t, \Delta T_t)$ Controller update (6.13)
  $z_t \leftarrow z_t + u_t$ Process update
  Return: $\Phi(z_t)$
end procedure

procedure SampleGlobalMinimum(S)
  Estimate Fréchet distribution $\mathcal{F}_{s,q}$ following Sec. 6.3.1
  for $l = 1$ **to** S **do**
    $f_*^l = \mathcal{F}_{s,q}^{-1}(\xi^l)$. $\xi^l \sim U(0, 1)$
  end for
  Return: $f_*^1, \ldots, f_*^S$
end procedure

procedure EvaluateAndUpdateGPs(x_t)
  $y = f(x_t)$, $y_j = g_j(x_t)$ $j = \{1, \ldots, G\}$
  $\mathcal{D}_t^f \leftarrow \{y, x_t\}$, $\mathcal{D}_t^{g_j} \leftarrow \{y_j, x_t\}$ $j = \{1, \ldots, G\}$
  Update hyperparameters of GP models
end procedure

BoTorch does not support optimization under non-linear constraints, which is needed to solve (6.12). To overcome this, we use the implementation of COBYLA (Powell, 1994) from nlopt[3].

In all experiments, the noise of the likelihood is fixed to $\sigma_{\text{n}} = 0.01$ for all GPs. The chosen hyperpriors on the lengthscales and the signal variance are reported in Table B.1, where $\mathcal{U}(a, b)$ refers to a uniform prior on the interval $[a, b]$, $\mathcal{G}(a, b)$ refers

[3] nlopt.readthedocs.io/en/latest/

to a Gamma prior with concentration a and rate b, and $\mathcal{N}(a, b^2)$ refers to a normal distribution with mean a and standard deviation b.

In Sec. 5.2., both, the Michalewicz and the Hartman functions are normalized to have zero mean and unit variance. The true minimum is known for both functions, which allows to compute the regret.

In Sec. 5.4, the goal is to find the state feedback gain $x \in \mathbb{R}^{4\times 1}$ for the cart-pole problem that minimizes a quadratic cost $f(x)$, which penalizes deviations of the pendulum states $s_k = [\varphi_k, \dot{\varphi}_k, l_k, \dot{l}_k]^\top$ from an equilibrium point s^*. The pole angle is φ_k, the pole angular velocity is $\dot{\varphi}_k$, the cart displacement is l_k, and the cart velocity is $\dot{l}_k$. The input to the system is the cart acceleration a_k, which is given by $a_k = x^\top(s_k - s^*) + 0.01 \sum_1^{N_{\text{simu}}}(l_k - l^*)$, where an integrator, with gain 0.01, is added to eliminate the steady-state the error. For each parametrization x, the constraint value is computed as the maximum displacement of the cart over a simulation of $N_{\text{simu}} = 800$ steps, i.e., $g(x) = \max(l_k),\ k = \{1, \ldots, N_{\text{simu}}\}$. Constraint violation is quantified as $g(x) > l_{\max}$, where $l_{\max}$ is the physical limit of the rail in which the cart moves. To allow the system to dissipate energy, the damping value of the simulated cart-pole in MuJoCo was increased from 1.0 to 1.5.

Table B.1: Hyperprior choices for the GP model hyperparameters for all experiments.

		Lengthscale λ	Variance σ^2
Michalewicz 10D	f	$\mathcal{U}(0.01, 0.3)$	$\mathcal{N}(0.5, 0.25^2)$
	g	$\mathcal{U}(0.01, 0.3)$	$\mathcal{N}(0.5, 0.25^2)$
Hartman 6D	f	$\mathcal{G}(1.0, 5.0)$	$\mathcal{N}(0.5, 0.25^2)$
	g	$\mathcal{G}(1.0, 5.0)$	$\mathcal{N}(0.5, 0.25^2)$
NN compression	f	$\mathcal{U}(0.01, 0.3)$	$\mathcal{N}(0.5, 0.2^2)$
	g	$\mathcal{U}(0.01, 0.3)$	$\mathcal{N}(7.5, 2.0^2)$
Pendulum	f	$\mathcal{U}(0.01, 0.3)$	$\mathcal{N}(1.0, 0.25^2)$
	g	$\mathcal{U}(0.01, 0.3)$	$\mathcal{N}(0.5, 0.25^2)$

B.5 Additional results

In this section, we present complementary results to those presented in Sec. 6.5.2.

To decouple the influence of the hyperparameter learning from the performance of the acquisition function itself, we fix the GP hyperparameters and sample the true objective f and the true constraint g from the corresponding GP priors. To obtain such samples we follow the same approach as in (Hernández-Lobato et al., 2016): First, the input domain is discretized to an irregular grid of 8000 points.

Table B.2: Constrained (top) and unconstrained in-model comparisons (bottom). Simple regret r_T (mean $\pm$ std) and percentage of safe evaluations Ω.

	3D Synthetic function	
	r_T	
EI	1.03 ± 0.50	
MES	1.03 ± 0.43	
PI	0.86 ± 0.41	
UCB	1.00 ± 0.43	
Xs	$\mathbf{0.19 \pm 0.34}$	
	r_T	Ω (%)
EIC	0.71 ± 0.61	21 ± 19
PESC	1.32 ± 0.62	14 ± 6
XsF	$\mathbf{0.30 \pm 0.51}$	$\mathbf{52 \pm 15}$

Second, function evaluations are randomly sampled from the corresponding GP prior at such locations. Finally, the GP is conditioned on those evaluations and the resulting posterior mean is used as true objective. The lengthscales where fixed to 0.1 and the signal variance to 1.0.

The simple regret cannot be computed because the true minimum of the GP sample is unknown a priori. Instead, we report results assuming a very conservative lower bound on all the possible sampled functions, i.e., $\min_{x \in \mathcal{X}} f(x) = -4.0$. We allow a maximum of $T = 100$ iterations, and a budget of failures $B = 15$ in the constrained case. The experiments were repeated 50 times for all algorithms. At each repetition, a new function is sampled from the GP priors.

In Table B.2, we show a performance comparison of both, Xs and XsF in optimizing a 3D input space. Without the influence of hyperparameter optimization, the proposed methods reach lower observations than state-of-the-art methods.